SPRINGER REFERENCE

Claude Sammut
Geoffrey I. Webb
Editors

Encyclopedia of Machine Learning

Springer

Encyclopedia of Machine Learning

Claude Sammut, Geoffrey I. Webb (Eds.)

Encyclopedia of Machine Learning

With 293 Figures and 78 Tables

Springer

Editors
Claude Sammut
School of Computer Science and Engineering
University of New South Wales
Sydney
Australia 2052
claude@cse.unsw.edu.au

Geoffrey I. Webb
Faculty of Information Technology
Clayton School of Information Technology
Monash University
P.O. Box 63
Victoria
Australia 3800
Geoff.Webb@monash.edu

ISBN 978-0-387-30768-8 e-ISBN 978-0-387-30164-8
Print and electronic bundle ISBN 978-0-387-34558-1
DOI 10.1007/978-0-387-30164-8
Springer New York

Library of Congress Control Number: 2010935441

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The term "Machine Learning" came into wide-spread use following the first workshop by that name, held at Carnegie-Mellon University in 1980. The papers from that workshop were published as Machine Learning: An Artificial Intelligence Approach, edited by Ryszard Michalski, Jaime Carbonell and Tom Mitchell. Machine Learning came to be identified as a research field in its own right as the workshops evolved into international conferences and journals of machine learning appeared.

Although the field coalesced in the 1980s, research on what we now call machine learning has a long history. In his 1950 paper on "Computing Machinery and Intelligence", Alan Turing introduced his imitation game as a means of determining if a machine could be considered intelligent. In the same paper he speculates that programming the computer to have adult level intelligence would be too difficult. "Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain". Investigations into induction, a fundamental operation in learning, go back much further to Francis Bacon and David Hume in the 17th and 18th centuries.

Early approaches followed the classical AI tradition of symbolic representation and logical inference. As machine learning began to be used in a wide variety of areas, the range of techniques expanded to incorporate ideas from psychology, information theory, statistics, neuroscience, genetics, operations research and more. Because of this diversity, it is not always easy for a new researcher to find his or her way around the machine learning landscape. The purpose of this encyclopedia is to guide enquiries into the field as a whole and to serve as an entry point to specific topics, providing overviews and, most importantly, references to source material. All the entries have been written by experts in their field and have been refereed and revised by an international editorial board consisting of leading machine learning researchers.

Putting together an encyclopedia for such a diverse field has been a major undertaking. We thank all the authors, without whom this would not have been possible. They have devoted their expertise and patience to the project because of their desire to contribute to this dynamic and still growing field. A project as large as this could only succeed with the help of the area editors whose specialised knowledge was essential in defining the range and structure of the entries.

The encyclopedia was started by the enthusiasm of Springer editors Jennifer Evans and Oona Schmidt and continued with the support of Melissa Fearon. Special thanks to Andrew Spencer, who oversaw production and kept everyone, including the editors on track.

Claude Sammut and Geoffrey I. Webb

Editors-in-Chief

Claude Sammut
School of Computer Science and Engineering
University of New South Wales
Sydney, Australia 2052
claude@cse.unsw.edu.au

Geoffrey I. Webb
Faculty of Information Technology
Clayton School of Information Technology
Monash University
P.O. Box 63
Victoria, Australia 3800
Geoff.Webb@monash.edu

Area Editors

Charu Aggarwal
IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne
NY 10532
USA
charu@us.ibm.com

Wray Buntine
NICTA
Locked Bag 8001
Canberra ACT 2601
Australia
wray.buntine@nicta.com.au

James Cussens
Department of Biology (Area 17)
York Centre for Complex Systems Analysis
University of York
PO Box 373
York YO10 5YW
UK
jc@cs.york.ac.uk

Luc De Raedt
Dept. of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
3001 Heverlee
Belgium
luc.deraedt@cs.kuleuven.be

Peter A. Flach
Department of Computer Science
University of Bristol
Woodland Road
Bristol BS8 1UB
UK
Peter.Flach@bristol.ac.uk

Russ Greiner
Department of Computing Science
University of Alberta
Athabasca Hall 359
Edmonton
Alberta T6G 2E8
Canada
greiner@cs.ualberta.ca

Eamonn Keogh
Computer Science & Engineering Department
University of California
Riverside
California
CA 92521
USA
eamonn@cs.ucr.edu

Michael L. Littman
Department of Computer Science
Rutgers, the State University of New Jersey
110 Frelinghuysen Road
Piscataway
New Jersey 08854-8019
USA
mlittman@cs.rutgers.edu

Sridhar Mahadevan
Department of Computer Science
University of Massachusetts
140 Governor's Drive
Amherst
MA 01003
USA
mahadeva@cs.umass.edu

Stan Matwin
School of Information Technology and Engineering
University of Ottawa
800 King Edward Ave., P.O. Box 450 Stn A
Ottawa
Ontario K1N 6N5
Canada
stan@site.uottawa.ca

Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500

Austin
Texas
TX 78712-0233
USA
risto@cs.utexas.edu

Dunja Mladenic
Department for Intelligent Systems
J. Stefan Institute
Jamova 39
1000 Ljubljana
Slovenia
Dunja.Mladenic@ijs.si

C. David Page
Department of Biostatistics and Medical Informatics
University of Wisconsin Medical School
1300 University Avenue
Wisconsin
Madison WI 53706
USA
page@biostat.wisc.edu

Bernhard Pfahringer
Department of Computer Science
University of Waikato
Private Bag 3105
Hamilton
New Zealand
bernhard@cs.waikato.ac.nz

Michail Prokopenko
CSIRO
Macquarie University
Building E6B,
Campus Herring Road
North Ryde
NSW
Australia 2113

Frank Stephan
Department of Mathematics
National University of Singapore
2 Science Drive 2
S14, Singapore 117543
Singapore
fstephan@comp.nus.edu.sg

Peter Stone
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin
Texas
TX 78712-0233
USA
pstone@cs.utexas.edu

Prasad Tadepalli
School of Electrical Engineering and Computer Science
Oregon State University
1148 Kelley Engineering Center
Corvallis
Oregon
OR 97331-5501
USA
tadepall@eecs.oregonstate.edu

Takashi Washio
The Institute of Scientific and Industrial Research
Osaka University
8-1 Mihogaoka
Osaka
Ibaraki 567
Japan
washio@ar.sanken.osaka-u.ac.jp

List of Contributors

Pieter Abbeel
Department of Electrical Engineering
and Computer Sciences
University of California
746 Sutardja Dai Hall #1758
CA 94720-1758, Berkeley
California
USA
pabbeel@stanford.edu

Charu C. Aggarwal
IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne
NY 10532
USA
charu@us.ibm.com

Biliana Alexandrova-Kabadjova
General Directorate of Central Bank Operations
Central Banking Operations Division
Bank of Mexico
Av. 5 de Mayo No. 6
Col. Centro, C.P. 06059
Mexico, D.F
balexandrova@banxico.org.mx

Periklis Andritsos
Thoora Inc.
Toronto, ON
Canada
periklis@thoora.com

Peter Auer
Institute of Computer Science
University of Leoben
Franz-Josef-Strasse 18
8700 Leoben
Austria
auer@unileoben.ac.at

J. Andrew Bagnell
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
USA
dbagnell@ri.cmu.edu

Michael Bain
University of New South Wales
Sydney
Australia
mike@cse.unsw.edu.au

Arindam Banerjee
Department of Computer Science and
Engineering
University of Minnesota
Minneapolis, MN
USA
banerjee@cs.umn.edu

Andrew G. Barto
Department of Computer Science
University of Massachusetts Amherst
272 Computer Science Building
Amherst, MA 01003
USA
barto@cs.umass.edu

Rohan A. Baxter
Analytics, Intelligence and Risk
Australian Taxation Office
PO Box 900
Civic Square, ACT 2608
Australia
r.baxter@computer.org

Bettina Berendt
Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A
3001 Heverlee
Belgium
Bettina.Berendt@cs.kuleuven.be

Indrajit Bhattacharya
IBM India Research Laboratory
New Delhi
India

Mustafa Bilgic
University of Maryland
AV Williams Bldg
Rm 3217
College Park, MD 20742
USA

Mauro Birattari
IRIDIA
Université Libre de Bruxelles
Brussels
Belgium
mbiro@ulb.ac.be

Hendrik Blockeel
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
3001 Heverlee
Belgium
Hendrik.Blockeel@cs.kuleuven.be

Shawn Bohn
Pacific Northwest National Laboratory

Antal van den Bosch
Tilburg centre for Creative Computing
Tilburg University
P.O. Box 90153
5000 LE, Tilburg
The Netherlands
Antal.vdnBosch@uvt.nl

Janez Brank
Department for Intelligent Systems
Jožef Stefan Institute
Jamova 39
1000 Ljubljana
Slovenia
janez.brank@ijs.si

Jürgen Branke
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
Universität Karlsruhe (TH)
76128 Karlsruhe
Germany
branke@aifb.uni-karlsruhe.de

Pavel Brazdil
LIAAD-INESC Porto L.A./Faculdade de Economia
Laboratory of Artificial
Intelligence and Computer Science
University of Porto
Rua de Ceuta n. 118
6.piso
Porto 4050-190
Portugal
pbrazdil@liaad.up.pt

Gavin Brown
The University of Manchester
School of Computer Science
Kilburn Building
Oxford Road
Manchester, M13 9PL
UK
Gavin.Brown@manchester.ac.uk

Ivan Bruha
Department of Computing & Software
McMaster University
Hamilton, ON
Canada
bruha@cas.mcmaster.ca

M.D. Buhmann
Numerische Mathematik
Justus-Liebig University
Mathematisches Institut
Heinrich-Buff-Ring 44
35392 Giessen
Germany
Martin.Buhmann@math.uni-giessen.de

Wray L. Buntine
NICTA
Locked Bag 8001
Canberra ACT 2601
Australia
wray.buntine@nicta.com.au

Tibério Caetano
Research School of Information Sciences
and Engineering
Australian National University
Canberra ACT 0200
Australia
tibério.caetano@nicta.com.au

Nicola Cancedda
Xerox Research Centre Europe
6, chemin de Maupertuis
38240 Meylan
France
nicola.cancedda@xrce.xerox.com

Gail A. Carpenter
Department of Cognitive and Neural Systems
Center for Adaptive Systems
Boston University
Boston, MA
USA

John Case
Department of Computer and Information
Sciences
University of Delaware
Newark DE 19716-2586
USA
case@cis.udel.edu

Tonatiuh Peña Centeno
Economic Research Division
Bank of Mexico
Av. 5 de Mayo # 18
Col. Centro, C.P. 06059
Mexico, D.F.

Deepayan Chakrabarti
Yahoo! Research
701 1st Avenue
Sunnyvale, CA 94089
USA
deepay@yahoo-inc.com

Philip K. Chan
Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL
USA
pkc@cs.fit.edu

Massimiliano Ciaramita
Yahoo! Research Barcelona
Ocata 1
Barcelona 8003
Spain
massi@yahoo-inc.com

Adam Coates
Department of Computer Science
Stanford University
Stanford, CA
USA

David Cohn
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
USA
david.cohn@somerandom.com

David Corne
Heriot-Watt University
Earl Mountbatten Building
Edinburgh EH14 4AS
UK
dwcorne@macs.hw.ac.uk

Susan Craw
IDEAS Research Institute
School of Computing
The Robert Gordon University
St. Andrew Street
Aberdeen AB25 1HG
Scotland
UK
s.crow@comp.rgu.ac.uk

Artur Czumaj
Department of Computer Science
University of Warwick
Coventry CV4 7AL
UK
aczumaj@acm.org

Walter Daelemans
Department of Linguistics
CLIPS University of Antwerp
Prinsstraat 13
Antwerpen
Belgium
walter.daelemans@ua.ac.be

Sanjoy Dasgupta
Department of Computer Science and Engineering
University of California
San Diego
9500 Gilman Drive
Mail Code 0404
La Jolla, California 92093-0404
USA
dasgupta@cs.ucsd.edu

Gerald DeJong
Department of Computer Science
University of Illinois at Urbana
Urbana, IL
USA
mrebl@uiuc.edu

Marco Dorigo
IRIDIA
Université Libre de Bruxelles
Avenue Franklin Roosevelt 50
1050 Brussels
Belgium
mdorigo@ulb.ac.be

Kurt Driessens
Departement Computerwetenschappen
Katholieke Universiteit Leuven
Celestijnenlaan 200 A
3001 Heverlee
Belgium
kurt.driessens@cs.kuleuven.be

Christopher Drummond
Integrated Reasoning
National Research Council Institute
for Information Technology
1200 Montreal Road
Building M-50, Room 374
Ottawa, ON K1A 0R6
Canada
Christopher.Drummond@nrc-cnrc.gc.ca

Yaakov Engel
AICML, Department of Computing Science
University of Alberta
2-21 Athabasca Hall
Edmonton
Alberta T6G 2E8
Canada
yakiengel@gmail.com

Scott E. Fahlman
Language Technologies Institute
Carnegie Mellon University GHC 6417
5000 Forbes Avenue
Pittsburgh, PA 15213
USA
sef@cs.cmu.edu

Alan Fern
School of Electrical Engineering and
Computer Science
Oregon State University
2071 Kelley Engineering Center
Corvallis, OR 97330-5501
USA
afern@eecs.orst.edu

Peter A. Flach
Department of Computer Science
University of Bristol
Woodland Road
Bristol, BS8 1UB
UK
Peter.Flach@bristol.ac.uk

Pierre Flener
Department of Information Technology
Uppsala University
Box 337
SE-751 05 Uppsala
Sweden
Pierre.Flener@it.uu.se

Johannes Fürnkranz
TU Darmstadt
Fachbereich Informatik
Hochschulstraße 10
64289 Darmstadt
Germany
juffi@ke.informatik.tu-darmstadt.de

Thomas Gärtner
Knowledge Discovery
Fraunhofer Institute for Intelligent Analysis and
Information Systems
Schloss Birlinghoven
53754 Sankt Augustin
Germany
thomas.gaertner@iais.fraunhofer.de

João Gama
Laboratory of Artificial Intelligence
and Decision Support
University of Porto
Porto
Portugal
jgama@fep.up.pt

Alma Lilia García-Almanza
General Directorate of Information Technology
Bank of Mexico
Av. 5 de Mayo No. 1
Col. Centro, C.P. 06059
Mexico, D.F.
algarcia@banxico.org.mx

Gemma C. Garriga
Laboratoire d'Informatique de Paris 6
Universite Pierre et Marie Curie
4 place Jussieu
Paris 75005
France
gemma.garriga@hut.fi

Wulfram Gerstner
Laboratory of Computational Neuroscience
Brain Mind Institute
Ecole Polytechnique Fédérale de Lausanne
Station 15
1015 Lausanne EPFL
Switzerland
wulfram.gerstner@ep.ch

Lise Getoor
Department of Computer Science
University of Maryland
AV Williams Bldg, Rm 3217
College Park, MD 20742
USA
getoor@cs.umd.edu

Christophe Giraud-Carrier
Department of Computer Science
Brigham Young University
3361 TMCB Provo
UT 84602
USA

Marko Grobelnik
Department for Intelligent Systems
Jožef Stefan Institute
Jamova 39
1000, Ljubljana
Slovenia
Marko.Grobelnik@ijs.si

Stephen Grossberg
Department of Cognitive
Boston University
677 Beacon Street
Boston, MA 02215
USA
steve@bu.edu

Jiawei Han
Department of Computer Science
University of Illinois
at Urbana Champaign
201 N. Goodwin Avenue
Urbana, IL 61801
USA
hanj@cs.uiuc.edu

Julia Handl
Faculty of Life Sciences in Manchester
University of Manchester
UK
j.handl@manchester.ac.uk

Michael Harries
Technology Strategy Division
Advanced Products Group, Citrix Labs
North Ryde
NSW 2113
Australia

Jun He
Department of Computer Science
Aberystwyth University
Aberystwyth SY23 3DB
Wales
UK
j.he@cs.bham.ac.uk

Bernhard Hengst
School of Computer Science & Engineering
University of New South Wales
Sydney
NSW 2052
Australia
bernhardh@cse.unsw.edu.au

Tom Heskes
Radboud University Nijmegen
Toernooiveld 1
6525 ED
Nijmegen
The Netherlands
t.heskes@science.ru.nl

Geoffrey Hinton
Department of Computer Science Office PT 290 G
University of Toronto
6 King's College Road
M5S 3G4, Toronto
Ontario
Canada
hinton@cs.toronto.edu

Lawrence Holder
School of Electrical Engineering and Computer Science
Box 642752
Washington State University
Pullman, WA 99164
USA
holder@wsu.edu

Tamás Horváth
Department of Computer Science III
University of Bonn and Fraunhofer IAIS
Fraunhofer Institute for Intelligent
Analysis and Information Systems
Schloss Birlinghoven
53754 Sankt Augustin
Germany
tamas.horvath@ais.fraunhofer.de

Eyke Hüllermeier
Knowledge Engineering & Bioinformatics
Head of the KEBI Lab
Department of Mathematics and Computer Science
Philipps-Universität Marburg
Mehrzweckgebäude
Hans-Meerwein-Straße
35032 Marburg
Germany
eyke@informatik.uni-marburg.de

Phil Husbands
Department of Informatics
University of Sussex
Brighton BN19QH
UK
philh@sussex.ac.uk

Marcus Hutter
Australian National University
RSIS Room B259
Building 115
Corner of North and Daley Road
ACT 0200
Canberra
Australia
marcus.hutter@anu.edu.au

Christian Igel
Institut für Neuroinformatik
Ruhr-Universität Bochum
Universitstr. 150
44780 Bochum
Germany
Christian.Igel@neuroinformatik.ruhr-uni-bochum.de

Sanjay Jain
Department of Computer Science
National University of Singapore
13 Computing Drive
Singapore 117417
Republic of Singapore
sanjay@comp.nus.edu.sg

Tommy R. Jensen
Institut für Mathematik
Alpen-Adria-Universität Klagenfurt
Universitässtr. 65-67
9020 Klagenfurt
Austria
tjensen@uni-klu.ac.at

Xin Jin
University of Illinois at Urbana-Champaign
Toernooiveld 1
6525 ED
Urbana, IL
USA

Antonis C. Kakas
Department of Computer Science
University of Cyprus
75 Kallipoleos Str., P.O. Box 537
Nicosia 1678
Cyprus
antonis@ucy.ac.cy

Subbarao Kambhampati
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ
USA
rao@asu.edu

Anne Kao
The Boeing Company
P.O. Box 3707 MC 7L-43
Seattle, WA 98124-2207
USA
akao@thumper.rt.cs.boeing.com

George Karypis
Department of Computer Science and Engineering
Digital Technology Center
and Army HPC Research Center
University of Minnesota
Minneapolis, MN 55455
USA
karypis@cs.umn.edu

Samuel Kaski
Laboratory of Computer
and Information Science
Helsinki University of Technology
P.O. Box 5400
02015 TKK
Finland
samuel.kaski@tkk.fi

Carlos Kavka
Istituto Nazionale di Fisica Nucleare
University of Trieste
Trieste 34127
Italy
Carlos.Kavka@ts.infn.it

James Kennedy
U.S. Bureau of Labor Statistics
Postal Square Building
2 Massachusetts Ave., NE
Washington, DC 20212-0001
USA
Kennedy.Jim@bls.gov

Eamonn Keogh
Computer Science & Engineering Department
University of California
Riverside, CA 92521
USA
eamonn@cs.ucr.edu

Kristian Kersting
Knowledge Discovery
Fraunhofer IAIS
Schloß Birlinghoven
53754 Sankt Augustin
Germany
kristian.kersting@iais.fraunhofer.de

Joshua Knowles
University of Manchester

Aleksander Kołcz
Microsoft One Microsoft Way
Redmond, WA 98052
USA
alek@ir.iit.edu

Kevin B. Korb
School of Information Technology
Monash University
Room 205, Bldg 63, 3800
Clayton, Victoria
Australia
kbkorb@gmail.com

Stefan Kramer
Institut für Informatik/I12
Technische Universität München
Boltzmannstr. 3
85748 Garching b. München
Germany
kramer@in.tum.de

Krzysztof Krawiec
Institute of Computing Science
Poznan University of Technology
Piotrowo
60-695 Poznan
Poland
krawiec@cs.put.poznan.pl

Nicolas Lachiche
Image Sciences, Computer Sciences and Remote Sensing Laboratory
64, bld Brant
67400 Illkirch-Graffenstaden
France
Nicolas.Lachiche@urs.u-strasbg.fr

Michail G. Lagoudakis
Department of Electronic and Computer Engineering
Technical University of Crete
73100 Chania
Crete
Greece
lagoudakie@intelligence.tuc.gr

John Langford
Yahoo Research
New York, NY 10011
USA
jl@yahoo-inc.com

Pier Luca Lanzi
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano 20133
Italy
lanzi@elet.polimi.it

Nada Lavrač
Department of Knowledge Technologies
Jožef Stefan Institute
Jamova 39
Ljubljana
Slovenia
Faculty of Information Technology
University of Nova Gorica
Vipavska 13
5000 Nova Gorica
Slovenia

Christina Leslie
Computational Biology Program
Sloan-Kettering Institute
Memorial Sloan-Kettering Cancer Center
1275 York Ave
Mail Box #460
New York, NY 10065
cleslie@cbio.mskcc.org

Shiau Hong Lim
University of Illinois
IL
USA
shonglim@uiuc.edu

Charles X. Ling
The University of Western Ontario
Canada
dr_charles_ling@yahoo.com

Huan Liu
Computer Science and Engineering
Ira Fulton School of Engineering
Arizona State University
Brickyard Suite 501
699 South Mill Avenue
Tempe, AZ 85287-8809
USA
huan.liu@asu.edu

Bin Liu
Faculty of Information Technology
Monash University
Melbourne 2052
Australia
bin.liu@infotech.monash.edu.au

John Lloyd
College of Engineering and Computer Science
The Australian National University
0200, Canberra ACT
Australia
jwl@mail.rise.anu.edu.au

Shie Mannor
Department of Electrical Engineering
Israel Institute of Technology
Technion
Technion City
32000 Haifa
Israel
shie@ee.technion.ac.il

Eric Martin
Department of Artificial Intelligence
School of Computer Science and Engineering
University of New South Wales
NSW 2052
Sydney
Australia
emartin@cse.unsw.edu.au

Serafín Martínez-Jaramillo
General Directorate of Financial System Analysis
Financial System Analysis Division
Bank of Mexico
Av. 5 de Mayo No. 1
Col. Centro, C.P. 06059
Mexico, D.F
smartin@banxico.org.mx

Stan Matwin
School of Information Technology and Engineering
University of Ottawa
Ottawa, ON
Canada
stan@site.uottawa.ca

Julian McAuley
Statistical Machine Learning Program
Department of Engineering and
Computer Science
National University of Australia
NICTA, Locked Bag 8001
Canberra ACT 2601
Australia
julian.mcauley@nicta.com.au

Prem Melville
Machine Learning
IBM T. J. Watson Research Center
Route 134/P.O. Box 218
1101 Kitchawan Rd
Yorktown Heights, NY 10598
USA
pmelvil@us.ibm.com

Pietro Michelucci
Strategic Analysis, Inc.
4075 Wilson Blvd
Suite 200
Arlington, VA 22203
USA
pmichelucci@sainc.com

Rada Mihalcea
Department of Computer Science
and Engineering
University of North Texas
Denton, TX 76203-6886
USA
rada@cs.unt.edu

Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-0233
USA
risto@cs.utexas.edu

Dunja Mladenić
Department of Knowledge Technologies
Jožef Stefan Insitute
Jamova 39
1000, Ljubljana
Slovenia
Dunja.Mladenic@ijs.si

Katharina Morik
Department of Computer Science
Technische Universität Dortmund
Dortmund
Germany
katharina.morik@tu-dortmund.de

Jun Morimoto
Advanced Telecommunication
Research Institute International ATR
Kyoto
Japan

Abdullah Mueen
Department of Computer Science and Engineering
University California-Riverside
Riverside, CA 92521
USA

Paul Munro
School of Information Sciences
University of Pittsburgh
Pittsburgh, PA
USA
pmunro@mail.sis.pitt.edu

Ion Muslea
Language Weaver, Inc.
4640 Admiralty Way, Suite 1210
Marina del Rey, CA 90292
USA
imuslea@languageweaver.com

Galileo Namata
Department of Computer Science
University of Maryland
College Park, MD 20742
USA

Sriraam Natarajan
Department of Computer Sciences
University of Wisconsin Medical School
1300 University Avenue
Madison, WI 53706
USA
natarasr@biostat.wisc.edu

Andrew Y. Ng
Stanford AI Laboratory
Stanford University
353 Serra Mall, Gates Building 1A
Stanford, CA 94305-9010
USA
ang@cs.stanford.edu

Siegfried Nijssen
Institut für Informatik
Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, Gebäude 079
79110 Freiburg i. Br.
Germany
snijssen@informatik.uni-freiburg.de

William Stafford Noble
Department of Genome Sciences
University of Washington
Seattle, WA
USA
noble@gs.washington.edu

Petra Kralj Novak
Department of Knowledge Technologies
Jožef Stefan Institute
Jamova 39
Ljubljana
Slovenia
Petra.Kralj.Novak@ijs.si

Daniel Oblinger
DARPA/IPTO
3701 Fairfax Drive
Arlington, VA 22203
USA
oblinger@pobox.com

Peter Orbanz
Department of Engineering
Cambridge University
Trumpington Street
Cambridge, CB2 1PZ
UK

Miles Osborne
Institute for Communicating and
Collaborative Systems
University of Edinburgh
2 Buccleuch Place
Edinburgh EH8 9LW
Scotland
UK
miles@inf.ed.ac.uk

C. David page
Department of Biostatistics and Medical Informatics
University of Wisconsin Medical School
1300 University Avenue
Madison, WI 53706
USA
page@biostat.wisc.edu

Jonathan Patrick
Telfer School of Management
University of Ottawa
55 Laurier avenue
Ottawa, ON K1N 6N5
Canada
patrick@telfer.uottawa.ca

Claudia Perlich
Data Analytics Research Group
IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
USA
perlich@us.ibm.com

Jan Peters
Department of Empirical
Inference and Machine Learning
Max Planck Institute for Biological Cybernetics
Spemannstr. 38
72076 Tuebingen
Germany
mail@jan-peters.net

Bernhard Pfahringer
Department of Computer Science
University of Waikato
Private Bag 3105
Hamilton
New Zealand
bernhard@cs.waikato.ac.nz

Steve Poteet
Boeing Phantom Works
P.O. Box 3707 MC 7L-43
Seattle, WA
USA

Pascal Poupart
School of Computer Science
University of Waterloo
200 University Avenue West
Waterloo
ON N2L 3G1
Canada
pponpart@cs.uwaterloo.ca

Rob Powers
Computer Science Department
Stanford University
353 Serra Mall
Stanford, CA 94305
USA
powers@cs.stanford.edu

Cecilia M. Procopiuc
AT&T Labs
Florham Park, NJ
USA
magda@research.att.com

Martin L. Puterman
Centre for Health Care Management
Sauder School of Business
University of British Columbia
2053 Main Mall
Vancouver, BC V6T 1Z2
Canada
marty@chem.ubc.ca

Lesley Quach
Boeing Phantom Works
P.O. Box 3707 MC 7L-43
Seattle, WA
USA

Novi Quadrianto
Department of Engineering and Computer Science
Australian National University NICTA London Circuit
Canberra ACT 0200
Australia
novi.quadrianto@nicta.com.au

Luc De Raedt
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
BE - 3001 Heverlee
Belgium
luc.deraedt@cs.kuleuven.be

Dev Rajnarayan
NASA Ames Research Center
Mail Stop 269-1
Moffett Field, CA 94035
USA

Adwait Ratnaparkhi
Yahoo! Labs
Santa Clara
California
USA
adwait_ratnaparkhi@yahoo.com

Soumya Ray
School of EECS
Oregon State University
1148 Kelley Engineering Center
97331
Corvallis, OR
USA
sray@eecs.oregonstate.edu

Mark Reid
Research School of Information Sciences and Engineering
The Australian National University
Canberra, ACT 0200
Australia
mark.reid@anu.edu.au

Jean-Michel Renders
Xerox Research Centre Europe
6, chemin de Maupertuis
38240 Meylan
France

John Risch
Pacific Northwest National Laboratory

Jorma Rissanen
Complex Systems Computation Group
Department of Computer Science
Helsinki Institute of Information Technology
Helsinki 00014
Finland
Jorma.Rissanen@hiit.fi

Nicholas Roy
Massachusetts Institute of Technology
Cambridge, MA
USA

Lorenza Saitta
Università del Piemonte Orientale
Alessandria
Italy
Michele.Sebag@lri.fr

Yasubumi Sakakibara
Department of Biosciences and Informatics
Keio University
yasu@bio.keio.ac.jp
Hiyoshi
Kohoku-ku
Japan

Claude Sammut
School of Computer Science and Engineering
The University of New South Wales
Sydney
NSW 2052
Australia
claude@cse.unsw.edu.au

Joerg Sander
Department of Computing Science
University of Alberta
Edmonton, AB
Canada
joerg@cs.ualberta.ca

Scott Sanner
Statistical Machine Learning Group
NICTA, 7 London Circuit, Tower A
ACT 2601
Canberra
Australia
scott.sanner@nicta.com.au

Stefan Schaal
Department of Computer Science
University of Southern California
ATR Computational Neuroscience Labs
3641 Watt Way
Los Angeles, CA 90089-2520
USA
sschaal@usc.edu

Ute Schmid
Department of Information Systems
and Applied Computer Science
University of Bamberg
Feldkirchenstr. 21
96045 Bamberg
Germany
ute.schmid@uni-bamberg.de

Stephen Scott
University of Nebraska
Lincoln, NE
USA

Michele Sebag
Laboratoire de Recherche en Informatique
Université Paris-Sud
Bât 490
91405 Orsay
France
Michele.Sebag@lri.fr

Prithviraj Sen
University of Maryland
AV Williams Bldg, Rm 3217
College Park, MD 20742
USA

Hanhuai Shan
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN
USA
shan@cs.umn.edu

Hossam Sharara
Department of Computer Science
University of Maryland
College Park, MD 20742
Maryland
USA

Victor S. Sheng
The University of Western Ontario
Canada

Jelber Sayyad Shirabad
School of Information Technology
and Engineering
University of Ottawa
800 King Edward
P.O. Box 450
Stn A, K1N 6N5
Ottawa, Ontario
Canada
jsayyad@site.uottawa.ca

Yoav Shoham
Computer Science Department
Stanford University
353 Serra Mall
Stanford, CA 94305
USA
shoham@standford.edn

Thomas R. Shultz
Department of Psychology and
School of Computer Science
McGill University
1205 Dr. Penfield Avenue
Montréal
QC H3A 1B1
Canada
thomas.shultz@mcgill.ca

Ricardo Silva
Gatsby Computational Neuroscience Unit
University College London
Alexandra House
17 Queen Square
London WC1N 3AR
UK
rbas@gatsby.ucl.ac.uk

Vikas Sindhwani
IBM T. J. Watson Research Center
Route 134/P.O. Box 218
1101 Kitchawan Rd
Yorktown Heights, NY 10598
USA

Moshe Sipper
Department of Computer Science
Ben-Gurion University
P.O. Box 653
Beer-Sheva 84105
Israel
sipper@cs.bgu.ac.il

William D. Smart
Associate Professor
Department of Computer Science and Engineering
Washington University in St. Louis
Campus Box 1045
One Brookings Drive
St. Louis, MO 63130
USA
wds@cse.wustl.edu

Carlos Soares
LIAAD-INESC Porto L.A./Faculdade de Economia
Laboratory of Artificial Intelligence
and Computer Science
University of Porto
Rua de Ceuta n. 118
6.piso, 4050-190
Porto
Portugal

Christian Sohler
Heinz Nixdorf Institute & Computer Science Department
University of Paderborn
Fuerstenallee 11
33102 Paderborn
Germany
csohler@upb.de

Frank Stephan
Department of Computer Science
and Department of Mathematics
National University of Singapore
Singapore 119076
Republic of Singapore
fstephan@comp.nus.edu.sg

Peter Stone
Department of Computer Sciences
The University of Texas at Austin
Austin, TX
USA
pstone@cs.utexas.edu

Alexander L. Strehl
Department of Computer Science
Rutgers University
110 Frelinghuysen Road
Piscataway, NJ 08854
USA
strehl@cs.rutgers.edu

Prasad Tadepalli
School of Electrical Engineering and Computer Science
Oregon State University
1148 Kelley Engineering Center
Corvallis, OR 97331-5501
USA
tadepall@eecs.oregonstate.edu

Russ Tedrake
Department of Computer Science
Massachusetts Institute of Technology
32 Vassar Street
Cambridge, MA 02139
USA
russt@mit.edu

Yee Whye Teh
Gatsby Computational Neuroscience Unit
University College London
17 Queen Square
London WC1N 3AR
UK
yeewhye@gmail.com

Jon Timmis
Department of Computer Science
and Department of Electronics
University of York
Heslington
York 10 5DD
UK
jtimmis@cs.york.ac.uk

Jo-Anne Ting
University of Edinburgh

Kai Ming Ting
Gippsland School of Information Technology
Monash University
Gippsland Campus Churchill
3842, Victoria
Australia
kaiming.ting@infotech.monash.edu.au

Ljupčo Todorovski
Faculty of Administration
University of Ljubljana
Gosarjeva 5
1000 Ljubljana
Slovenia
Ljupco.Todorovski@fu.uni-lj.si

Hannu Toivonen
Department of Computer Science
University of Helsinki
P.O. Box 68 (Gustaf Hällströmin katu 2b)
00014 Helsinki
Finland
hannu.toivonen@cs.helsinki.fi

Luís Torgo
Department of Computer Science
Faculty of Sciences
University of Porto
Rua Campo Alegre
1021/1055, 4169–007
Porto
Portugal
ltorgo@dcc.fc.up.pt

Panayiotis Tsaparas
Microsoft Research
Microsoft
Mountain View, CA
USA
panayiotis.tsaparas@micorsoft.com

Paul E. Utgoff
Department of Computer Science
University of Massachusetts
140 Governor's Drive
Amherst, MA 01003–4610
USA

William Uther
NICTA and the University of New South Wales
William.Uther@nicta.com.au

Sethu Vijayakumar
University of Edinburgh
University of Southern California

Ricardo Vilalta
Department of Computer Science
University of Houston
4800 Calhoun Rd
Houston, TX 77204-3010
USA

Michail Vlachos
IBM Zürich Research Laboratory
Säumerstrasse 4
8803 Rüschlikon
Switzerland
michaliso@gmail.com

Kiri L. Wagstaff
Machine Learning Systems
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA
USA
kiri.wagstaff@jpl.nasa.gov1

Geoffrey I. Webb
Faculty of Information Technology
Clayton School of Information Technology
Monash University
P.O. Box 63
Victoria 3800
Australia
Geoff.Webb@monash.edu

R. Paul Wiegand
Institute for Simulation and Training
University of Central Florida
Orlando, FL
USA
paul@tesseraet.org
wiegand@ist.ucf.edu

Eric Wiewiora
University of California
San Diego
ewiewior@cs.ucsd.edu

Anthony Wirth
Department of Computer Science
and Software Engineering
The University of Melbourne
Victoria 3010
Australia
awirth@csse.unimelb.edu.au

Michael Witbrock
Cycorp, Inc.
3721 Executive Center Drive
Austin, TX 78731
USA
witbrock@cyc.com

David Wolpert
NASA Ames Research Center
Moffett Field, CA
USA
dhw@ptolemy.arc.nasa.gov

Stefan Wrobel
Department of Computer Science
University of Bonn, and Fraunhofer IAIS
(Institute for Intelligent Analysis and
Information Systems) Fraunhofer IAIS
Schloss Birlinghoven
53754 Sankt Augustin
Germany

Jason Wu
Boeing Phantom Works
P.O. Box 3707 MC 7L-43
Seattle, WA
USA

Zhao Xu
Knowledge Discovery
Fraunhofer IAIS
Schloß Birlinghoven
53754 Sankt Augustin
Germany

Ying Yang
Australian Taxation Office
990 White Horse Road
Box Hill
VIC 3128
Australia
Ying.Yang@ato.gov.au

Sungwook Yoon
PARC Labs
Coyote Hill Road
Palo Alto, CA
USA

Thomas Zeugmann
Division of Computer Science
Graduate School of
Information Science and Technology
Hokkaido University
Sapparo
Japan
thomas@ist.hokudai.ac.jp

Xinhua Zhang
School of Computer Science
Australian National University
NICTA London Circuit
Canberra
Australia
xinhua.zhang@anu.edu.au

Ying Zhao
Department of Computer Science and Technology
Tsinghua University
Beijing 100084
China

Fei Zheng
Faculty of Information Technology
Monash University
Clayton School of I.T.
Room 217, Bldg 63
Wellington Road
Clayton
Melbourne
Victoria 3800
Australia
fei.zheng@infotech.monash.edu.au

Xiaojin Zhu
Department of Computer Sciences
University of Wisconsin-Madison
1210 West Dayton Street, 53706
Madison, WI
USA
jerryzhu@cs.wisc.edu

0–9

1-Norm Distance

►Manhattan Distance

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

A

Abduction

Antonis C. Kakas
University of Cyprus, Nicosia, Cyprus

Definition

Abduction is a form of reasoning, sometimes described as "deduction in reverse," whereby given a rule that "*A follows from B*" and the observed result of "*A*" we infer the condition "*B*" of the rule. More generally, given a theory, *T*, modeling a domain of interest and an observation, "*A*," we infer a hypothesis "*B*" such that the observation follows deductively from *T* augmented with "*B*." We think of "*B*" as a possible explanation for the observation according to the given theory that contains our rule. This new information and its consequences (or ramifications) according to the given theory can be considered as the result of a (or part of a) learning process based on the given theory and driven by the observations that are explained by abduction. Abduction can be combined with ►induction in different ways to enhance this learning process.

Motivation and Background

Abduction is, along with induction, a *synthetic* form of reasoning whereby it generates, in its explanations, new information not hitherto contained in the current theory with which the reasoning is performed. As such, it has a natural relation to learning, and in particular to *knowledge intensive learning*, where the new information generated aims to complete, at least partially, the current knowledge (or model) of the problem domain as described in the given theory.

Early uses of abduction in the context of machine learning concentrated on how abduction can be used as a theory revision operator for identifying where the current theory could be revised in order to accommodate the new learning data. This includes the work of Michalski (1993), Ourston and Mooney (1994), and Ade, Malfait, and Raedt (1994). Another early link of abduction to learning was given by the ►explanation based learning method (DeJong & Mooney, 1986), where the abductive explanations of the learning data (training examples) are generalized to all cases.

Following this, it was realized (Flach & Kakas, 2000) that the role of abduction in learning could be strengthened by linking it to induction, culminating in a hybrid integrated approach to learning where abduction and induction are tightly integrated to provide powerful learning frameworks such as the ones of Progol 5.0 (Muggleton & Bryant, 2000) and HAIL (Ray, Broda, & Russo, 2003). On the other hand, from the point of view of abduction as "inference to the best explanation" (Josephson & Josephson, 1994) the link with induction provides a way to distinguish between different explanations and to select those explanations that give a better inductive generalization result.

A recent application of abduction, on its own or in combination with induction, is in Systems Biology where we try to model biological processes and pathways at different levels. This challenging domain provides an important development test-bed for these methods of knowledge intensive learning (see e.g., King et al., 2004; Papatheodorou, Kakas, & Sergot, 2005; Ray, Antoniades, Kakas, & Demetriades, 2006; Tamaddoni-Nezhad, Kakas, Muggleton, & Pazos, 2004; Zupan et al., 2003).

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Structure of the Learning Task

Abduction contributes to the learning task by first explaining, and thus rationalizing, the training data according to a given and current model of the domain to be learned. These abductive explanations either form on their own the result of learning or they feed into a subsequent phase to generate the final result of learning.

Abduction in Artificial Intelligence

Abduction as studied in the area of Artificial Intelligence and the perspective of learning is mainly defined in a logic-based approach (Other approaches to abduction include the set covering approach See, e.g., Reggia (1983) or case-based explanation, e.g., Leake (1995).) as follows.

Given a set of sentences T (a theory or model), and a sentence O (observation), the abductive task is the problem of finding a set of sentences H (abductive explanation for O) such that:

1. $T \cup H \models O$,
2. $T \cup H$ is consistent,

where $\models$ denotes the deductive entailment relation of the formal logic used in the representation of our theory and consistency refers also to the corresponding notion in this logic. The particular choice of this underlying formal framework of logic is in general a matter that depends on the problem or phenomena that we are trying to model. In many cases, this is based on ▶first order predicate calculus, as, for example, in the approach of theory completion in Muggleton and Bryant (2000). But other logics can be used, e.g., the nonmonotonic logics of default logic or logic programming with negation as failure when the modeling of our problem requires this level of expressivity.

This basic formalization as it stands, does not fully capture the explanatory nature of the abductive explanation H in the sense that it necessarily conveys some reason why the observations hold. It would, for example, allow an observation O to be explained by itself or in terms of some other observations rather than in terms of some "deeper" reason for which the observation must hold according to the theory T. Also, as the above specification stands, the observation can be abductively explained by generating in H some new (general) theory completely unrelated to the given theory T. In this case, H does not account for the observations O according to the given theory T and in this sense it may not be considered as an explanation for O relative to T. For these reasons, in order to specify a "level" at which the explanations are required and to understand these relative to the given general theory about the domain of interest, the members of an explanation are normally restricted to belong to a special preassigned, domain-specific class of sentences called *abducible*.

Hence abduction, is typically applied on a model, T, in which we can separate two disjoint sets of predicates: the *observable* predicates and the *abducible* (*or open*) predicates. The basic assumption then is that our model T has reached a sufficient level of comprehension of the domain such that all the incompleteness of the model can be isolated (under some working hypotheses) in its abducible predicates. The observable predicates are assumed to be completely defined (in T) in terms of the abducible predicates and other background auxiliary predicates; any incompleteness in their representation comes from the incompleteness in the abducible predicates. In practice, the empirical observations that drive the learning task are described using the observable predicates. Observations are represented by formulae that refer only to the observable predicates (and possibly some background auxiliary predicates) typically by ground atomic facts on these observable predicates. The abducible predicates describe underlying (theoretical) relations in our model that are not observable directly but can, through the model T, bring about observable information.

The assumptions on the abducible predicates used for building up the explanations may be subject to restrictions that are expressed through *integrity constraints*. These represent additional knowledge that we have on our domain expressing general properties of the domain that remain valid no matter how the theory is to be extended in the process of abduction and associated learning. Therefore, in general, an *abductive theory* is a triple, denoted by $\langle T, A, \mathrm{IC}\rangle$, where T is the background theory, A is a set of abducible predicates, and IC is a set of integrity constraints. Then, in the definition of an abductive explanation given above, one more requirement is added:

3. $T \cup H$ satisfies IC.

The satisfaction of integrity constraints can be formally understood in several ways (see Kakas, Kowalski, & Toni, 1992 and references therein). Note that the integrity constraints reduce the number of explanations for a set of observations filtering out those explanations that do not satisfy them. Based on this notion of abductive explanation a *credulous* form of abductive entailment is defined. Given an abductive theory, $T = \langle T, A, \text{IC} \rangle$, and an observation O then, O is *abductively entailed* by T, denoted by $T \models_A O$, if there exists an abductive explanation of O in T.

This notion of abductive entailment can then form the basis of a coverage relation for learning in the face of incomplete information.

Abductive Concept Learning

Abduction allows us to reason in the face of incomplete information. As such when we have learning problems where the background data on the training examples is incomplete the use of abduction can enhance the learning capabilities.

Abductive concept learning (ACL) (Kakas & Riguzzi, 2000) is a learning framework that allows us to learn from incomplete information and to later be able to classify new cases that again could be incompletely specified. Under ACL, we learn abductive theories, $\langle T, A, \text{IC} \rangle$ with abduction playing a central role in the covering relation of the learning problem. The abductive theories learned in ACL contain both rules, in T, for the concept(s) to be learned as well as general clauses acting as integrity constraints in IC.

Practical problems that can be addressed with ACL: (1) concept learning from incomplete background data where some of the background predicates are incompletely specified and (2) concept learning from incomplete background data together with given integrity constraints that provide some information on the incompleteness of the data. The treatment of incompleteness through abduction is integrated within the learning process. This allows the possibility of learning more compact theories that can alleviate the problem of over fitting due to the incompleteness in the data. A specific subcase of these two problems and important third application problem of ACL is that of (3) multiple predicate learning, where each predicate is required to be learned from the incomplete data for the other predicates. Here the abductive reasoning can be used to suitably connect and integrate the learning of the different predicates. This can help to overcome some of the nonlocality difficulties of multiple predicate learning, such as order-dependence and global consistency of the learned theory.

ACL is defined as an extension of ▶Inductive Logic Programming (ILP) where both the background knowledge and the learned theory are abductive theories. The central formal definition of ACL is given as follows where examples are atomic ground facts on the target predicate(s) to be learned.

Definition 1 (**Abductive Concept Learning**)
Given

- *A set of positive examples* E^+
- *A set of negative examples* E^-
- *An abductive theory* $T = \langle P, A, I \rangle$ *as background theory*
- *An hypothesis space* $\mathcal{T} = \langle \mathcal{P}, \mathcal{I} \rangle$ *consisting of a space of possible programs* $\mathcal{P}$ *and a space of possible constraints* $\mathcal{I}$

Find
A set of rules $P' \in \mathcal{P}$ *and a set of constraints* $I' \in \mathcal{I}$ *such that the new abductive theory* $T' = \langle P \cup P', A, I \cup I' \rangle$ *satisfies the following conditions*

- $T' \models_A E^+$
- $\forall e^- \in E^-,\ T' \not\models_A e^-$

where E^+ *stands for the conjunction of all positive examples.*

An individual example e is said to be *covered* by a theory T' if $T' \models_A e$. In effect, this definition replaces the deductive entailment as the example coverage relation in the ILP problem with abductive entailment to define the ACL learning problem.

The fact that the conjunction of positive examples must be covered means that, for every positive example, there must exist an abductive explanation and the explanations for all the positive examples must be consistent with each other. For negative examples, it is required that no abductive explanation exists for any of them. ACL can be illustrated as follows.

Example 2 *Suppose we want to learn the concept father. Let the background theory be* $T = \langle P, A, \varnothing \rangle$ *where*:

$P = \{parent(john, mary), male(john),$
$parent(david, steve),$
$parent(kathy, ellen), female(kathy)\},$
$A = \{male, female\}.$

Let the training examples be:

$E^+ = \{father(john, mary), father(david, steve)\},$
$E^- = \{father(kathy, ellen), father(john, steve)\}.$

In this case, a possible hypotheses $T' = \langle P \cup P', A, I' \rangle$ *learned by ACL would consist of*

$P' = \{father(X, Y) \leftarrow parent(X, Y), male(X)\},$
$I' = \{ \leftarrow male(X), female(X)\}.$

This hypothesis satisfies the definition of ACL because:

1. $T' \models_A father(john, mary), father(david, steve)$ *with* $\Delta = \{male(david)\}$.
2. $T' \not\models_A father(kathy, ellen)$, *as the only possible explanation for this goal, namely* $\{male(kathy)\}$ *is made inconsistent by the learned integrity constraint in* I'.
3. $T' \not\models_A father(john, steve)$, *as this has no possible abductive explanations.*

Hence, despite the fact that the background theory is incomplete (in its abducible predicates), ACL can find an appropriate solution to the learning problem by suitably extending the background theory with abducible assumptions. Note that the learned theory without the integrity constraint would not satisfy the definition of ACL, because there would exist an abductive explanation for the negative example $father(kathy, ellen)$, *namely* $\Delta^- = \{male(kathy)\}$. *This explanation is prohibited in the complete theory by the learned constraint together with the fact* $female(kathy)$.

The algorithm and learning system for ACL is based on a decomposition of this problem into two subproblems: (1) learning the rules in P' together with appropriate explanations for the training examples and (2) learning integrity constraints driven by the explanations generated in the first part. This decomposition allows ACL to be developed by combining the two IPL settings of explanatory (predictive) learning and confirmatory (descriptive) learning. In fact, the first subproblem can be seen as a problem of learning from entailment, while the second subproblem as a problem of learning from interpretations.

Abduction and Induction

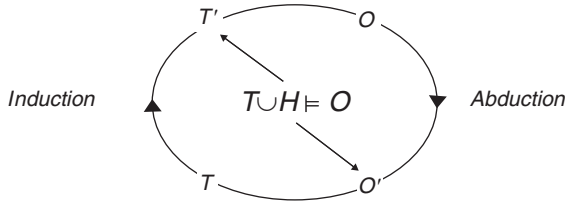
The utility of abduction in learning can be enhanced significantly when this is integrated with *induction*. Several approaches for synthesizing abduction and induction in learning have been developed, e.g., Ade and Denecker (1995), Muggleton and Bryant (2000), Yamamoto (1997), and Flach and Kakas (2000). These approaches aim to develop techniques for knowledge intensive learning with complex background theories. One problem to be faced by purely inductive techniques, is that the training data on which the inductive process operates, often contain gaps and inconsistencies. The general idea is that abductive reasoning can feed information into the inductive process by using the background theory for inserting new hypotheses and removing inconsistent data. Stated differently, abductive inference is used to complete the training data with hypotheses about missing or inconsistent data that explain the example or training data, using the background theory. This process gives alternative possibilities for assimilating and generalizing this data.

Induction is a form of synthetic reasoning that typically generates knowledge in the form of new general rules that can provide, either directly, or indirectly through the current theory T that they extend, new interrelationships between the predicates of our theory that can include, unlike abduction, the observable predicates and even in some cases new predicates. The inductive hypothesis thus introduces new, hitherto unknown, links between the relations that we are studying thus allowing new predictions on the observable predicates that would not have been possible before from the original theory under any abductive explanation.

An inductive hypothesis, H, extends, like in abduction, the existing theory T to a new theory $T' = T \cup H$, but now H provides new links between observables and nonobservables that was missing or incomplete in the original theory T. This is particularly evident from the fact that induction can be performed even with an empty given theory T, using just the set of observations. The observations specify incomplete (usually extensional) knowledge about the observable

predicates, which we try to *generalize* into new knowledge. In contrast, the generalizing effect of abduction, if at all present, is much more limited. With the given current theory T, that abduction always needs to refer to, we implicitly restrict the generalizing power of abduction as we require that the basic model of our domain remains that of T. Induction has a stronger and genuinely new generalizing effect on the observable predicates than abduction. While the purpose of abduction is to extend the theory with an explanation and then reason with it, thus enabling the generalizing potential of the given theory T, in induction the purpose is to extend the given theory to a new theory, which can provide new possible observable consequences.

This complementarity of abduction and induction – abduction providing explanations from the theory while induction generalizes to form new parts of the theory – suggests a basis for their integration within the context of theory formation and theory development. A *cycle of integration* of abduction and induction (Flach & Kakas, 2000) emerges that is suitable for the task of incremental modeling (Fig. 1). Abduction is used to transform (and in some sense normalize) the observations to information on the abducible predicates. Then, induction takes this as input and tries to generalize this information to general rules for the abducible predicates now treating these as observable predicates for its own purposes. The cycle can then be repeated by adding the learned information on the abducibles back in the model as new partial information



Abduction. Figure 1. The cycle of abductive and inductive knowledge development. The cycle is governed by the "equation" $T \cup H \models O$, where T is the current theory, O the observations triggering theory development, and H the new knowledge generated. On the left-hand side we have induction, its output feeding into the theory T for later use by abduction on the right; the abductive output in turn feeds into the observational data O' for later use by induction, and so on

on the incomplete abducible predicates. This will affect the abductive explanations of new observations to be used again in a subsequent phase of induction. Hence, through this cycle of integration the abductive explanations of the observations are added to the theory, not in the (simple) form in which they have been generated, but in a generalized form given by a process of induction on these.

A simple example, adapted from Ray et al. (2003), that illustrates this cycle of integration of abduction and induction is as follows. Suppose that our current model, T, contains the following rule and background facts:

sad(X) ← *tired(X), poor(X),*
tired(oli), tired(ale), tired(kr),
academic(oli), academic(ale), academic(kr),
student(oli), lecturer(ale), lecturer(kr),

where the only observable predicate is *sad*/1.

Given the observations $O = \{sad(ale), sad(kr),$ *not* $sad(oli)\}$ can we improve our model? The incompleteness of our model resides in the predicate *poor*. This is the only abducible predicate in our model. Using abduction we can explain the observations O via the explanation:

E = *{poor(ale), poor(kr), not poor(oli)}.*

Subsequently, treating this explanation as training data for inductive generalization we can generalize this to get the rule:

poor(X) ← *lecturer(X)*

thus (partially) defining the abducible predicate *poor* when we extend our theory with this rule.

This combination of abduction and induction has recently been studied and deployed in several ways within the context of ILP. In particular, *inverse entailment* (Muggleton and Bryant, 2000) can be seen as a particular case of integration of abductive inference for constructing a "bottom" clause and inductive inference to generalize it. This is realized in Progol 5.0 and applied to several problems including the discovery of the function of genes in a network of metabolic pathways (King et al., 2004), and more recently to the study of

inhibition in metabolic networks (Tamaddoni-Nezhad, Chaleil, Kakas, & Muggleton, 2006; Tamaddoni-Nezhad et al., 2004). In Moyle (2000), an ILP system called ALECTO, integrates a phase of *extraction-case abduction* to transform each case of a training example to an abductive hypothesis with a phase of induction that generalizes these abductive hypotheses. It has been used to learn robot navigation control programs by completing the specific domain knowledge required, within a general theory of planning that the robot uses for its navigation (Moyle, 2002).

The development of these initial frameworks that realize the cycle of integration of abduction and induction prompted the study of the problem of *completeness* for finding any hypotheses H that satisfies the basic task of finding a consistent hypothesis H such that $T \cup H \models O$ for a given theory T, and observations O. Progol was found to be incomplete (Yamamoto, 1997) and several new frameworks of integration of abduction and induction have been proposed such as SOLDR (Ito & Yamamoto, 1998), CF-induction (Inoue, 2001), and HAIL (Ray et al., 2003). In particular, HAIL has demonstrated that one of the main reasons for the incompleteness of Progol is that in its cycle of integration of abduction and induction, it uses a very restricted form of abduction. Lifting some of these restrictions, through the employment of methods from abductive logic programming (Kakas et al., 1992), has allowed HAIL to solve a wider class of problems. HAIL has been extended to a framework, called XHAIL (Ray, 2009), for learning nonmonotonic ILP, allowing it to be applied to learn Event Calculus theories for action description (Alrajeh, Ray, Russo, & Uchitel, 2009) and complex scientific theories for systems biology (Ray & Bryant, 2008).

Applications of this integration of abduction and induction and the cycle of knowledge development can be found in the recent proceedings of the Abduction and Induction in Artificial Intelligence workshops in 2007 (Flach & Kakas, 2009) and 2009 (Ray, Flach, & Kakas, 2009).

Abduction in Systems Biology

Abduction has found a rich field of application in the domain of systems biology and the declarative modeling of computational biology. In a project called, Robot scientist (King et al., 2004), Progol 5.0 was used to generate abductive hypotheses about the function of genes. Similarly, learning the function of genes using abduction has been studied in GenePath (Zupan et al., 2003) where experimental genetic data is explained in order to facilitate the analysis of genetic networks. Also in Papatheodorou et al. (2005) abduction is used to learn gene interactions and genetic pathways from microarray experimental data. Abduction and its integration with induction has been used in the study of inhibitory effect of toxins in metabolic networks (Tamaddoni-Nezhad et al., 2004, 2006) taking into account also the temporal variation that the inhibitory effect can have. Another bioinformatics application of abduction (Ray et al., 2006) concerns the modeling of human immunodeficiency virus (HIV) drug resistance and using this in order to assist medical practitioners in the selection of antiretroviral drugs for patients infected with HIV. Also, the recently developed frameworks of XHAIL and CF-induction have been applied to several problems in systems biology, see e.g., Ray (2009), Ray and Bryant (2008), and Doncescu, Inoue, and Yamamoto (2007), respectively.

Cross References

►Explanation-Based Learning
►Inductive Logic Programming

Recommended Reading

Ade, H., & Denecker, M. (1995). AILP: Abductive inductive logic programming. In C. S. Mellish (Ed.), *IJCAI* (pp. 1201–1209). San Francisco: Morgan Kaufmann.

Ade, H., Malfait, B., & Raedt, L. D. (1994). Ruth: An ILP theory revision system. In *ISMIS94*. Berlin: Springer.

Alrajeh, D., Ray, O., Russo, A., & Uchitel, S. (2009). Using abduction and induction for operational requirements elaboration. *Journal of Applied Logic*, *7*(3), 275–288.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternate view. *Machine Learning*, *1*, 145–176.

Doncescu, A., Inoue, K., & Yamamoto, Y. (2007). Knowledge based discovery in systems biology using cf-induction. In H. G. Okuno & M. Ali (Eds.), *IEA/AIE* (pp. 395–404). Heidelberg: Springer.

Flach, P., & Kakas, A. (2000). Abductive and inductive reasoning: Background and issues. In P. A. Flach & A. C. Kakas (Eds.), *Abductive and inductive reasoning. Pure and applied logic*. Dordrecht: Kluwer.

Flach, P. A., & Kakas, A. C. (Eds.). (2009). Abduction and induction in artificial intelligence [Special issue]. *Journal of Applied Logic, 7*(3).

Inoue, K. (2001). Inverse entailment for full clausal theories. In *LICS-2001 workshop on logic and learning*.

Ito, K., & Yamamoto, A. (1998). Finding hypotheses from examples by computing the least generlisation of bottom clauses. In *Proceedings of discovery science '98* (pp. 303–314). Berlin: Springer.

Josephson, J., & Josephson, S. (Eds.). (1994). *Abductive inference: Computation, philosophy, technology*. New York: Cambridge University Press.

Kakas, A., Kowalski, R., & Toni, F. (1992). Abductive logic programming. *Journal of Logic and Computation*, *2*(6), 719–770.

Kakas, A., & Riguzzi, F. (2000). Abductive concept learning. *New Generation Computing*, *18*, 243–294.

King, R., Whelan, K., Jones, F., Reiser, P., Bryant, C., Muggleton, S., et al. (2004). Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, *427*, 247–252.

Leake, D. (1995). Abduction, experience and goals: A model for everyday abductive explanation. *The Journal of Experimental and Theoretical Artificial Intelligence*, *7*, 407–428.

Michalski, R. S. (1993). Inferential theory of learning as a conceptual basis for multistrategy learning. *Machine Learning*, *11*, 111–151.

Moyle, S. (2002). Using theory completion to learn a robot navigation control program. In *Proceedings of the 12th international conference on inductive logic programming* (pp. 182–197). Berlin: Springer.

Moyle, S. A. (2000). *An investigation into theory completion techniques in inductive logic programming*. PhD thesis, Oxford University Computing Laboratory, University of Oxford.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, *13*, 245–286.

Muggleton, S., & Bryant, C. (2000). Theory completion using inverse entailment. In *Proceedings of the tenth international workshop on inductive logic programming (ILP-00)* (pp. 130–146). Berlin: Springer.

Ourston, D., & Mooney, R. J. (1994). Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, *66*, 311–344.

Papatheodorou, I., Kakas, A., & Sergot, M. (2005). Inference of gene relations from microarray data by abduction. In *Proceedings of the eighth international conference on logic programming and non-monotonic reasoning (LPNMR'05)* (Vol. 3662, pp. 389–393). Berlin: Springer.

Ray, O. (2009). Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, *7*(3), 329–340.

Ray, O., Antoniades, A., Kakas, A., & Demetriades, I. (2006). Abductive logic programming in the clinical management of HIV/AIDS. In G. Brewka, S. Coradeschi, A. Perini, & P. Traverso (Eds.), *Proceedings of the 17th European conference on artificial intelligence. Frontiers in artificial intelligence and applications* (Vol. 141, pp. 437–441). Amsterdam: IOS Press.

Ray, O., Broda, K., & Russo, A. (2003). Hybrid abductive inductive learning: A generalisation of Progol. In *Proceedings of the 13th international conference on inductive logic programming. Lecture notes in artificial intelligence* (Vol. 2835, pp. 311–328). Berlin: Springer.

Ray, O., & Bryant, C. (2008). Inferring the function of genes from synthetic lethal mutations. In *Proceedings of the second international conference on complex, intelligent and software intensive systems* (pp. 667–671). Washington, DC: IEEE Computer Society.

Ray, O., Flach, P. A., & Kakas, A. C. (Eds.). (2009). Abduction and induction in artificial intelligence. *Proceedings of IJCAI 2009 workshop*.

Reggia, J. (1983). Diagnostic experts systems based on a set-covering model. *International Journal of Man-Machine Studies*, *19*(5), 437–460.

Tamaddoni-Nezhad, A., Chaleil, R., Kakas, A., & Muggleton, S. (2006). Application of abductive ILP to learning metabolic network inhibition from temporal data. *Machine Learning*, *64*(1–3), 209–230.

Tamaddoni-Nezhad, A., Kakas, A., Muggleton, S., & Pazos, F. (2004). Modelling inhibition in metabolic pathways through abduction and induction. In *Proceedings of the 14th international conference on inductive logic programming* (pp. 305–322). Berlin: Springer.

Yamamoto, A. (1997). Which hypotheses can be found with inverse entailment? In *Proceedings of the seventh international workshop on inductive logic programming. Lecture notes in artificial intelligence* (Vol. 1297, pp. 296–308). Berlin: Springer.

Zupan, B., Bratko, I., Demsar, J., Juvan, P., Halter, J., Kuspa, A., et al. (2003). Genepath: A system for automated construction of genetic networks from mutant data. *Bioinformatics*, *19*(3), 383–389.

Absolute Error Loss

►Mean Absolute Error

Accuracy

Definition

Accuracy refers to a measure of the degree to which the predictions of a ►model match the reality being modeled. The term *accuracy* is often applied in the context of ►classification models. In this context, *accuracy* = $P(\lambda(X) = Y)$, where XY is a ►joint distribution and the classification model λ is a function $X \rightarrow Y$. Sometimes, this quantity is expressed as a percentage rather than a value between 0.0 and 1.0.

The accuracy of a model is often assessed or estimated by applying it to test data for which the ►labels (Y values) are known. The accuracy of a classifier on test data may be calculated as *number of correctly classified objects/total number of objects*. Alternatively, a smoothing function may be applied, such as a ►Laplace estimate or an ►*m*-estimate.

Accuracy is directly related to ►error rate, such that *accuracy* = 1.0 − *error rate* (or when expressed as a percentage, *accuracy* = 100 − *error rate*).

Cross References

►Confusion Matrix
►Resubstitution Accuracy

ACO

►Ant Colony Optimization

Actions

In a ►Markov decision process, *actions* are the available choices for the decision-maker at any given *decision epoch*, in any given *state*.

Active Learning

David Cohn
Mountain View, CA, USA

Definition

The term *Active Learning* is generally used to refer to a learning problem or system where the learner has some role in determining on what data it will be trained. This is in contrast to *Passive Learning*, where the learner is simply presented with a ►training set over which it has no control. Active learning is often used in settings where obtaining ►labeled data is expensive or time-consuming; by sequentially identifying which examples are most likely to be useful, an active learner can sometimes achieve good performance, using far less ►training data than would otherwise be required.

Structure of Learning System

In many machine learning problems, the training data are treated as a fixed and given part of the problem definition. In practice, however, the training data are often not fixed beforehand. Rather, the learner has an opportunity to play a role in deciding what data will be acquired for training. This process is usually referred to as "active learning," recognizing that the learner is an active participant in the training process.

The typical goal in active learning is to select training examples that best enable the learner to minimize its loss on future test cases. There are many theoretical and practical results demonstrating that, when applied properly, active learning can greatly reduce the number of training examples, and even the computational effort required for a learner to achieve good generalization.

A toy example that is often used to illustrate the utility of active learning is that of learning a threshold function over a one-dimensional interval. Given +/− labels for N points drawn uniformly over the interval, the expected error between the true value of the threshold and any learner's best guess is bounded by $O(1/N)$. Given the opportunity to sequentially select the position of points to be labeled, however, a learner can pursue a binary search strategy, obtaining a best guess that is within $O(1/2^N)$ of the true threshold value.

This toy example illustrates the sequential nature of example selection that is a component of most (but not all) active learning strategies: the learner makes use of initial information to discard parts of the solution space, and to focus future data acquisition on distinguishing parts that are still viable.

Related Problems

The term "active learning" is usually applied in supervised learning settings, though there are many related problems in other branches of machine learning and beyond. The "exploration" component of the "exploration/exploitation" strategy in reinforcement learning is one such example. The learner *must* take actions to gain information, and must decide what actions will give him/her the information that will best minimize future loss. A number of fields of Operations Research predate and parallel machine learning work on active learning, including Decision Theory (North, 1968), Value of Information Computation, Bandit problems (Robbins, 1952), and Optimal Experiment Design (Fedorov, 1972; Box & Draper, 1987).

Active Learning Scenarios

When active learning is used for classification or regression, there are three common settings: *constructive* active learning, *pool-based* active learning, and *stream-based* active learning (also called *selective sampling*).

Constructive Active Learning

In constructive active learning, the learner is allowed to propose arbitrary points in the input space as examples to be labeled. While this in theory gives the learner the most power to explore, it is often not practical. One obstacle is the observation that most learning systems train on only a reduced representation of the instances they are presented with: text classifiers on bags of words (rather than fully-structured text) and speech recognizers on formants (rather than raw audio). While a learning system may be able to identify what pattern of formants would be most informative to label, there is no reliable way to generate audio that a human could recognize (and label) from the desired formants alone.

Pool-Based Active Learning

Pool-based active learning (McCallum & Nigam, 1998) is popular in domains such as text classification and speech recognition where unlabeled data are plentiful and cheap, but labels are expensive and slow to acquire. In pool-based active learning, the learner may not propose arbitrary points to label, but instead has access to a set of unlabeled examples, and is allowed to select which of them to request labels for.

A special case of pool-based learning is transductive active learning, where the test distribution is exactly the set of unlabeled examples. The goal then is to sequentially select and label a small number of examples that will best allow predicting the labels of those points that remain unlabeled.

A theme that is common to both constructive and pool-based active learning is the principle of sequential experimentation. Information gained from early queries allows the learner to focus its search on portions of the domain that are most likely to give it additional information on subsequent queries.

Stream-Based Active Learning

Stream-based active learning resembles pool-based learning in many ways, except that the learner only has access to the unlabeled instances as a stream; when an instance arrives, the learner must decide whether to ask for its label or let it go.

Other Forms of Active Learning

By virtue of the broad definition of active learning, there is no real limit on the possible settings for framing it. Angluin's early work on learning regular sets (Angluin, 1987) employed a "counterexample" oracle: the learner would propose a solution, and the oracle would either declare it correct, or divulge a counterexample – an instance on which the proposed and true solutions disagreed. Jin and Si (2003) describe a Bayesian method for selecting informative items to recommend when learning a collaborative filtering model, and Steck and Jaakkola (2002) describe a method best described as *unsupervised* active learning to build Bayesian networks in large domains.

While most active learning work assumes that the cost of obtaining a label is independent of the instance to be labeled, there are many scenarios where this is not the case. A mobile robot taking surface measurements must first travel to the point it wishes to sample, making distant points more expensive than nearby ones. In some cases, the cost of a query (e.g., the difficulty of traveling to a remote point to sample it) may not even be known until it is made, requiring the learner to learn a model of that as well. In these situations, the sequential nature of active learning is greatly accentuated, and a learner faces the additional challenges of planning under uncertainty (see "Greedy vs. Batch Active Learning," below).

Common Active Learning Strategies

1. *Version space partitioning.* The earliest practical active learning work (Ruff & Dietterich, 1989; Mitchell, 1982) explicitly relied on ►version space partitioning. These approaches tried to select examples on which there was maximal disagreement between hypotheses in the current version space. When such examples were labeled, they would invalidate as large a portion of the version space as possible. A limitation of explicit version space approaches is that, in underconstrained domains, a learner may waste its effort differentiating portions of the version space that have little

effect on the classifier's predictions, and thus on its error.

2. *Query by Committee* (Seung, Opper, & Sompolinsky 1992). In query by committee, the experimenter trains an ensemble of models, either by selecting randomized starting points (e.g., in the case of a neural network) or by bootstrapping the training set. Candidate examples are scored based on disagreement among the ensemble models – examples with high disagreement indicate areas in the sample space that are underdetermined by the training data, and therefore potentially valuable to label. Models in the ensemble may be looked at as samples from the version space; picking examples where these models disagree is a way of splitting the version space.
3. *Uncertainty sampling* (Lewis & Gail, 1994). Uncertainty sampling is a heuristic form of statistical active learning. Rather than sampling different points in the version space by training multiple learners, the learner itself maintains an explicit model of uncertainty over its input space. It then selects for labeling those examples on which it is least confident. In classification and regression problems, uncertainty contributes directly to expected loss (as the variance component of the "error = bias + variance" decomposition), so that gathering examples where the learner has greatest uncertainty is often an effective loss-minimization heuristic. This approach has also been found effective for non-probabilistic models, by simply selecting examples that lie near the current decision boundary. For some learners, such as support vector machines, this heuristic can be shown to be an approximate partitioning of the learner's version space (Tong & Koller, 2001).
4. *Loss minimization* (Cohn, Ghahramani, & Jordan, 1996). Uncertainty sampling can stumble when parts of the learner's domain are inherently noisy. It may be that, regardless of the number of samples labeled in some neighborhood, it will remain impossible to accurately predict these. In these cases, it would be desirable to not only model the learner's uncertainty over arbitrary parts of its domain, but also to model what effect labeling any future example is expected to have on that uncertainty. For some learning algorithms it is feasible to explicitly compute such estimates (e.g., for locally-weighted regression and mixture models, these estimates may be computed in closed form). It is, therefore, practical to select examples that directly minimize the expected loss to the learner, as discussed below under "Statistical Active Learning."

Statistical Active Learning

Uncertainty sampling and direct loss minimization are two examples of *statistical* active learning. Both rely on the learner's ability to statistically model its own uncertainty. When learning with a statistical model, such as a linear regressor or a mixture of Gaussians (Dasgupta, 1999), the objective is usually to find model parameters that minimize some form of expected loss. When active learning is applied to such models, it is natural to also select training data so as to minimize that same objective. As statistical models usually give us estimates on the probability of (as yet) unknown values, it is often straightforward to turn this machinery upon itself to assist in the active learning process (Cohn et al., 1996). The process is usually as follows:

1. Begin by requesting labels for a small random subsample of the examples $x_1, x_2, \mathrm{K}, x_n x$ and fit our model to the labeled data.
2. For any x in our domain, a statistical model lets us estimate both the conditional expectation $\hat{y}(x)$ and $\sigma^2_{\hat{y}(x)}$, the variance of that expectation. We estimate our current loss by drawing a new random sample of unlabeled data, and computing the averaged $\sigma^2_{\hat{y}(x)}$.
3. We now consider a candidate point $\tilde{x}$, and ask what reduction in loss we would obtain if we had labeled it $\tilde{y}$. If we knew its label with certainty, we could simply add the point to the training set, retrain, and compute the new expected loss. While we do not know the true $\tilde{y}$, we could, in theory, compute the new expected loss for every possible $\tilde{y}$ and average those losses, weighting them by our model's estimate of $p(\tilde{y}|\tilde{x})$. In practice, this is normally unfeasible; however, for some statistical models, such as locally-weighted regression and mixtures of Gaussians, we can compute the distribution of $p(\tilde{y}|\tilde{x})$ and its effect on $\sigma^2_{\hat{y}(x)}$ in closed form (Cohn et al., 1996).

4. Given the ability to estimate the expected effect of obtaining label $\tilde{y}$ for candidate $\tilde{x}$, we repeat this computation for a sample of Mcandidates, and then request a label for the candidate with the largest expected decrease in loss. We add the newly-labeled example to our training set, retrain, and begin looking at candidate points to add on the next iteration.

The Need for Reference Distributions

Step (2) above illustrates a complication that is unique to active learning approaches. Traditional "passive" learning usually relies on the assumption that the distribution over which the learner will be tested is the same as the one from which the training data were drawn. When the learner is allowed to select its own training data, it still needs some form of access to the distribution of data on which it will be tested. A pool-based or stream-based learner can use the pool or stream as a proxy for that distribution, but if the learner is allowed (or required) to construct its own examples, it risks wasting all its effort on resolving portions of the solution space that are of no interest to the problem at hand.

A Detailed Example: Statistical Active Learning with LOESS

LOESS (Cleveland, Devlin, & Gross, 1988) is a simple form of locally-weighted regression using a kernel function. When asked to predict the unknown output y corresponding to a given input x, LOESS computes a ►linear regression over known (x, y) pairs, in which it gives pair (x_i, y_i) weight according to the proximity of x_i to x. We will write this weighting as a kernel function, $K(x_i, x)$, or simplify it to k_i when there is no chance of confusion.

In the active learning setting, we will assume that we have a large supply of unlabeled examples drawn from the test distribution, along with labels for a small number of them. We wish to label a small number more so as to minimize the mean squared error (MSE) of our model. MSE can be decomposed into two terms: squared ►bias and variance. If we make the (inaccurate but simplifying) assumption that LOESS is approximately unbiased for the problem at hand, minimizing MSE reduces to minimizing the variance of our estimates.

Given n labeled pairs, and a prediction to make for input x, LOESS computes the following covariance statistics around x:

$$\mu_x = \frac{\sum_i k_i x_i}{n}, \quad \sigma_x^2 = \frac{\sum_i k_i (x_i - \mu_x)^2}{n},$$
$$\sigma_{xy} = \frac{\sum_i k_i (x_i - \mu_x)(y_i - \mu_y)}{n}$$
$$\mu_y = \frac{\sum_i k_i y_i}{n}, \quad \sigma_y^2 = \frac{\sum_i k_i (y_i - \mu_y)^2}{n},$$
$$\sigma_{y|x}^2 = \sigma_y^2 - \frac{\sigma_{xy}}{\sigma_x^2}$$

We can combine these to express the conditional expectation of y (our estimate) and its variance as:

$$\hat{y} = \mu_y + \frac{\sigma_{xy}}{\sigma_x^2}(x - \mu_x),$$
$$\sigma_{\hat{y}}^2 = \frac{\sigma_{y|x}^2}{n^2}\left(\sum_i k_i^2 + \frac{(x-\mu_x)^2}{\sigma_x^2}\sum_i k_i^2 \frac{(x_i - \mu_x)^2}{\sigma_x^2}\right).$$

Our proxy for model error is the variance of our prediction, integrated over the test distribution $\left\langle \sigma_{\hat{y}}^2 \right\rangle$. As we have assumed a pool-based setting in which we have a large number of unlabeled examples from that distribution, we can simply compute the above variance over a sample from the pool, and use the resulting average as our estimate.

To perform statistical active learning, we want to compute how our estimated variance will change if we add an (as yet unknown) label $\tilde{y}$ for an arbitrary $\tilde{x}$. We will write this new expected variance as $\left\langle \tilde{\sigma}_{\hat{y}}^2 \right\rangle$. While we do not *know* what value $\tilde{y}$ will take, our model gives us an estimated mean $\hat{y}(\tilde{x})$ and variance $\sigma_{\hat{y}(\tilde{x})}^2$ for the value, as above. We can add this "distributed" y value to LOESS just as though it were a discrete one, and compute the resulting expectation $\left\langle \tilde{\sigma}_{\hat{y}}^2 \right\rangle$ in closed form. Defining $\tilde{k}$ as $K(\tilde{x}, x)$, we write:

$$\left\langle \tilde{\sigma}_{\hat{y}}^2 \right\rangle = \frac{\left\langle \tilde{\sigma}_{y|x}^2 \right\rangle}{(n+\tilde{k})^2}\left(\sum_i k_i^2 + \tilde{k}^2 + \frac{(x-\tilde{\mu}_x)^2}{\tilde{\sigma}_x^2} \times \left(\sum_i k_i^2 \frac{(x_i - \tilde{\mu}_x)^2}{\tilde{\sigma}_x^2} + \tilde{k}^2 \frac{(\tilde{x} - \tilde{\mu}_x)^2}{\tilde{\sigma}_x^2}\right)\right),$$

where the component expectations are computed as follows:

$$\left\langle \tilde{\sigma}^2_{y|x} \right\rangle = \left\langle \tilde{\sigma}^2_{y} \right\rangle - \frac{\left\langle \tilde{\sigma}^2_{xy} \right\rangle}{\tilde{\sigma}^2_{x}},$$

$$\left\langle \tilde{\sigma}^2_{y} \right\rangle = \frac{n\sigma^2_y}{n+\tilde{k}} + \frac{n\tilde{k}(\sigma^2_{y|\tilde{x}} + (\hat{y}(\tilde{x}) - \mu_y)^2)}{(n+\tilde{k})^2},$$

$$\tilde{\mu}_x = \frac{n\mu_x + \tilde{k}\tilde{x}}{n+\tilde{k}},$$

$$\left\langle \tilde{\sigma}_{xy} \right\rangle = \frac{n\sigma_{xy}}{n+\tilde{k}} + \frac{n\tilde{k}(\tilde{x} - \mu_x)(\hat{y}(\tilde{x}) - \mu_y)}{(n+\tilde{k})^2},$$

$$\tilde{\sigma}^2_{x} = \frac{n\sigma^2_x}{n+\tilde{k}} + \frac{n\tilde{k}(\tilde{x} - \mu_x)^2}{(n+\tilde{k})^2},$$

$$\left\langle \tilde{\sigma}^2_{xy} \right\rangle = \left\langle \tilde{\sigma}_{xy} \right\rangle^2 + \frac{n^2\tilde{k}^2\sigma^2_{y|\tilde{x}}(\tilde{x} - \mu_x)^2}{(n+\tilde{k})^4}.$$

Greedy Versus Batch Active Learning

It is also worth pointing out that virtually all active learning work relies on greedy strategies – the learner estimates what single example best achieves its objective, requests that one, retrains, and repeats. In theory, it is possible to plan some number of queries ahead, asking what point is best to label now, given that N-1 more labeling opportunities remain. While such strategies have been explored in Operations Research for very small problem domains, their computational requirements make this approach unfeasible for problems of the size typically encountered in machine learning.

There are cases where retraining the learner after every new label would be prohibitively expensive, or where access to labels is limited by the number of iterations as well as by the total number of labels (e.g., for a finite number of clinical trials). In this case, the learner may select a set of examples to be labeled on each iteration. This batch approach, however, is only useful if the learner is able to identify a set of examples whose expected contributions are non-redundant, which substantially complicates the process.

Cross References

►Active Learning Theory

Recommended Reading

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2), 87–106.

Angluin, D. (1988). Queries and concept learning. *Machine Learning*, 2, 319–342.

Box, G. E. P., & Draper, N. (1987). *Empirical model-building and response surfaces*. New York: Wiley.

Cleveland, W., Devlin, S., & Gross, E. (1988). Regression by local fitting. *Journal of Econometrics*, 37, 87–114.

Cohn, D., Atlas, L., & Ladner, R. (1990). Training connectionist networks with queries and selective sampling. In D. Touretzky (Ed.)., *Advances in neural information processing systems*. Morgan Kaufmann.

Cohn, D., Ghahramani, Z., & Jordan, M. I. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4, 129–145. http://citeseer.ist.psu.edu/321503.html

Dasgupta, S. (1999). Learning mixtures of Gaussians. *Foundations of Computer Science*, 634–644.

Fedorov, V. (1972). *Theory of optimal experiments*. New York: Academic Press.

Kearns, M., Li, M., Pitt, L., & Valiant, L. (1987). *On the learnability of Boolean formulae*, *Proceedings of the 19th annual ACM conference on theory of computing* (pp. 285–295). New York: ACM Press.

Lewis, D. D., & Gail, W. A. (1994). *A sequential algorithm for training text classifiers. Proceedings of the 17th annual international ACM SIGIR conference* (pp. 3–12). Dublin.

McCallum, A., & Nigam, K. (1998). Employing EM and pool-based active learning for text classification. In *Machine learning: Proceedings of the fifteenth international conference* (ICML'98) (pp. 359–367).

North, D. W. (1968). A tutorial introduction to decision theory. *IEEE Transactions Systems Science and Cybernetics*, 4(3).

Pitt, L., & Valiant, L. G. (1988). Computational limitations on learning from examples. *Journal of the ACM (JACM)*, 35(4), 965–984.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 55, 527–535.

Ruff, R., & Dietterich, T. (1989). What good are experiments? *Proceedings of the sixth international workshop on machine learning*. Ithaca, NY.

Seung, H. S., Opper, M., & Sompolinsky, H. (1992). Query by committee. In *Proceedings of the fifth workshop on computational learning theory* (pp. 287–294). San Mateo, CA: Morgan Kaufmann.

Steck, H., & Jaakkola, T. (2002). Unsupervised active learning in large domains. In Proceeding of the conference on uncertainty in AI. http://citeseer.ist.psu.edu/steck02unsupervised.html

Active Learning Theory

Sanjoy Dasgupta
University of California, San Diego, La Jolla, CA, USA

Definition

The term *active learning* applies to a wide range of situations in which a learner is able to exert some control over its source of data. For instance, when fitting a

regression function, the learner may itself supply a set of data points at which to measure response values, in the hope of reducing the variance of its estimate. Such problems have been studied for many decades under the rubric of *experimental design* (Chernoff, 1972; Fedorov, 1972). More recently, there has been substantial interest within the machine learning community in the specific task of actively learning binary classifiers. This task presents several fundamental statistical and algorithmic challenges, and an understanding of its mathematical underpinnings is only gradually emerging. This brief survey will describe some of the progress that has been made so far.

Learning from Labeled and Unlabeled Data

In the machine learning literature, the task of learning a classifier has traditionally been studied in the framework of *supervised learning*. This paradigm assumes that there is a training set consisting of data points x (from some set $\mathcal{X}$) and their labels y (from some set $\mathcal{Y}$), and the goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that will accurately predict the labels of data points arising in the future. Over the past 50 years, tremendous progress has been made in resolving many of the basic questions surrounding this model, such as "how many training points are needed to learn an accurate classifier?"

Although this framework is now fairly well understood, it is a poor fit for many modern learning tasks because of its assumption that all training points automatically come labeled. In practice, it is frequently the case that the raw, abundant, easily obtained form of data is *unlabeled*, whereas labels must be explicitly procured and are expensive. In such situations, the reality is that the learner starts with a large pool of unlabeled points and must then strategically decide which ones it wants labeled: how best to spend its limited budget.

Example: Speech recognition. When building a speech recognizer, the unlabeled training data consists of raw speech samples, which are very easy to collect: just walk around with a microphone. For all practical purposes, an unlimited quantity of such samples can be obtained. On the other hand, the "label" for each speech sample is a segmentation into its constituent phonemes, and producing even one such label requires substantial human time and attention. Over the past decades, research labs and the government have expended an enormous amount of money, time, and effort on creating labeled datasets of English speech. This investment has paid off, but our ambitions are inevitably moving past what these datasets can provide: we would now like, for instance, to create recognizers for other languages, or for English in specific contexts. Is there some way to avoid more painstaking years of data labeling, to somehow leverage the easy availability of raw speech so as to significantly reduce the number of labels needed? This is the hope of active learning.

Some early results on active learning were in the *membership query* model, where the data is assumed to be *separable* (that is, some hypothesis h perfectly classifies all points) and the learner is allowed to query the label of *any* point in the input space $\mathcal{X}$ (rather than being constrained to a prespecified unlabeled set), with the goal of eventually returning the perfect hypothesis h^*. There is a significant body of beautiful theoretical work in this model (Angluin, 2001), but early experiments ran into some telling difficulties. One study (Baum & Lang, 1992) found that when training a neural network for handwritten digit recognition, the queries synthesized by the learner were such bizarre and unnatural images that they were impossible for a human to classify. In such contexts, the membership query model is of limited practical value; nonetheless, many of the insights obtained from this model carry over to other settings (Hanneke, 2007a).

We will fix as our standard model one in which the learner is *given* a source of unlabeled data, rather than being able to generate these points himself. Each point has an associated label, but the label is initially *hidden*, and there is a cost for revealing it. The hope is that an accurate classifier can be found by querying just a few labels, much fewer than would be required by regular supervised learning.

How can the learner decide which labels to probe? One option is to select the query points at random, but it is not hard to show that this yields the same label complexity as supervised learning. A better idea is to choose the query points *adaptively*: for instance, start by querying some random data points to get a rough sense of where the decision boundary lies, and then gradually refine the estimate of the boundary by specifically querying points in its immediate vicinity. In other

words, ask for the labels of data points whose particular positioning makes them especially informative. Such strategies certainly sound good, but can they be fleshed out into practical algorithms? And if so, do these algorithms work well in the sense of producing good classifiers with fewer labels than would be required by supervised learning?

On account of the enormous practical importance of active learning, there are a wide range of algorithms and techniques already available, most of which resemble the aggressive, adaptive sampling strategy just outlined, and many of which show promise in experimental studies. However, a big problem with this kind of sampling is that very quickly the set of labeled points no longer reflects the underlying data distribution. This makes it hard to show that the classifiers learned have good statistical properties (for instance, that they converge to an optimal classifier in the limit of infinitely many labels). This survey will only discuss methods that have proofs of statistical well-foundedness, and whose label complexity can be explicitly analyzed.

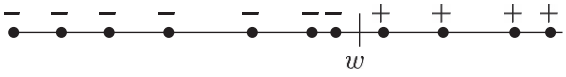
Motivating Examples

We will start by looking at a few examples that illustrate the enormous potential of active learning and that also make it clear why analyses of this new model require concepts and intuitions that are fundamentally different from those that have already been developed for supervised learning.

Example: Thresholds on the Line

Suppose the data lie on the real line, and the available classifiers are simple thresholding functions, $\mathcal{H} = \{h_w : w \in \mathbb{R}\}$:

$$h_w(x) = \begin{cases} +1 & \text{if } x \geq w \\ -1 & \text{if } x < w \end{cases}$$



To make things precise, let us denote the (unknown) underlying distribution on the data $(X, Y) \in \mathbb{R} \times \{+1, -1\}$ by $\mathbb{P}$, and let us suppose that we want a hypothesis $h \in \mathcal{H}$ whose error with respect to $\mathbb{P}$, namely $\text{err}_{\mathbb{P}}(h) = \mathbb{P}(h(X) \neq Y)$, is at most some ϵ. How many labels do we need?

In supervised learning, such issues are well understood. The standard machinery of sample complexity (using VC theory) tells us that if the data are *separable* – that is, if they can be perfectly classified by some hypothesis in $\mathcal{H}$ – then we need approximately $1/\epsilon$ random labeled examples from $\mathbb{P}$, and it is enough to return any classifier consistent with them.

Now suppose we instead draw $1/\epsilon$ *unlabeled* samples from $\mathbb{P}$. If we lay these points down on the line, their hidden labels are a sequence of −s followed by a sequence of +s, and the goal is to discover the point w at which the transition occurs. This can be accomplished with a simple binary search which asks for just $\log 1/\epsilon$ labels: first ask for the label of the median point; if it is +, move to the 25th percentile point, otherwise move to the 75th percentile point; and so on. Thus, for this hypothesis class, active learning gives an *exponential* improvement in the number of labels needed, from $1/\epsilon$ to just $\log 1/\epsilon$. For instance, if supervised learning requires a million labels, active learning requires just $\log 1{,}000{,}000 \approx 20$, literally!

It is a tantalizing possibility that even for more complicated hypothesis classes $\mathcal{H}$, a sort of generalized binary search is possible. A natural next step is to consider linear separators in *two* dimensions.

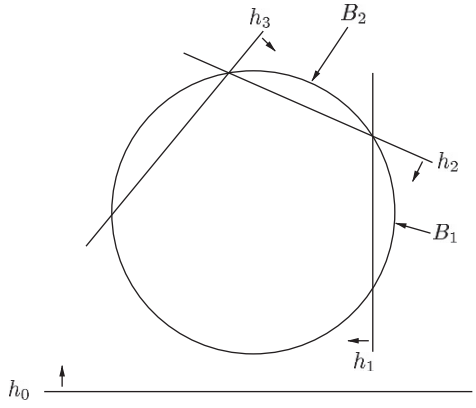
Example: Linear Separators in $\mathbb{R}^2$

Let $\mathcal{H}$ be the hypothesis class of linear separators in $\mathbb{R}^2$, and suppose the data is distributed according to some density supported on the perimeter of the unit circle. It turns out that the positive results of the one-dimensional case do not generalize: there are some target hypotheses in $\mathcal{H}$ for which $\Omega(1/\epsilon)$ labels are needed to find a classifier with error rate less than ϵ, no matter what active learning scheme is used.

To see this, consider the following possible target hypotheses (Fig. 1):

- h_0: all points are positive.
- h_i $(1 \leq i \leq 1/\epsilon)$: all points are positive except for a small slice B_i of probability mass ϵ.

The slices B_i are explicitly chosen to be disjoint, with the result that $\Omega(1/\epsilon)$ labels are needed to distinguish between these hypotheses. For instance, suppose nature chooses a target hypothesis at random from among the $h_i, 1 \leq i \leq 1/\epsilon$. Then, to identify this target with probability at least $1/2$, it is necessary to query points in at least (about) half the B_is.



Active Learning Theory. Figure 1. $\mathbb{P}$ **is supported on the circumference of a circle. Each** B_i **is an arc of probability mass** ϵ

Thus for these particular target hypotheses, active learning offers little improvement in sample complexity over regular supervised learning. What about other target hypotheses in $\mathcal{H}$, for instance those in which the positive and negative regions are more evenly balanced? It is quite easy (Dasgupta, 2005) to devise an active learning scheme which asks for $O(\min\{1/i(h), 1/\epsilon\}) + O(\log 1/\epsilon)$ labels, where $i(h) = \min\{\text{positive mass of } h, \text{negative mass of } h\}$. Thus even within this simple hypothesis class, the label complexity can run anywhere from $O(\log 1/\epsilon)$ to $\Omega(1/\epsilon)$, depending on the specific target hypothesis!

Example: An Overabundance of Unlabeled Data

In our two previous examples, the amount of unlabeled data needed was $O(1/\epsilon)$, exactly the usual sample complexity of supervised learning. But it is sometimes helpful to have significantly more unlabeled data than this. In Dasgupta (2005), a distribution $\mathbb{P}$ is described for which if the amount of unlabeled data is small (below any prespecified threshold), then the number of labels needed to learn the target linear separator is $\Omega(1/\epsilon)$; whereas if the amount of unlabeled data is much larger, then only $O(\log 1/\epsilon)$ labels are needed. This is a situation where most of the data distribution is fairly uninformative while a miniscule fraction is highly informative. A lot of unlabeled data is needed in order to get even a few of the informative points.

The Sample Complexity of Active Learning

We will think of the unlabeled points $x_1, \ldots, x_n$ as being drawn i.i.d. from an underlying distribution $\mathbb{P}_X$ on $\mathcal{X}$ (namely, the marginal of the distribution $\mathbb{P}$ on $\mathcal{X} \times \mathcal{Y}$), either all at once (a *pool*) or one at a time (a *stream*). The learner is only allowed to query the labels of points in the pool/stream; that is, it is restricted to "naturally occurring" data points rather than synthetic ones (Fig. 2). It returns a hypothesis $h \in \mathcal{H}$ whose quality is measured by its error rate, $\text{err}_{\mathbb{P}}(h)$.

In regular supervised learning, it is well known that if the VC dimension of $\mathcal{H}$ is d, then the number of labels that will with high probability ensure $\text{err}_{\mathbb{P}}(h) \leq \epsilon$ is roughly $O(d/\epsilon)$ if the data is separable and $O(d/\epsilon^2)$ otherwise (Haussler, 1992); various logarithmic terms are omitted here. For active learning, it is clear from the examples above that the VC dimension alone does not adequately characterize label complexity. Is there a different combinatorial parameter that does?

Generic Results for Separable Data

For separable data, it is possible to give upper and lower bounds on label complexity in terms of a special parameter known as the *splitting index* (Dasgupta, 2005). This is merely an existence result: the algorithm needed to realize the upper bound is intractable because it involves explicitly maintaining an ϵ-cover (a coarse approximation) of the hypothesis class, and the size of this cover is in general exponential in the VC dimension. Nevertheless, it does give us an idea of the kinds of label complexity we can hope to achieve.

Example. Suppose the hypothesis class consists of intervals on the real line: $\mathcal{X} = \mathbb{R}$ and $\mathcal{H} = \{h_{a,b} : a, b \in \mathbb{R}\}$, where $h_{a,b}(x) = \mathbf{1}(a \leq x \leq b)$. Using the splitting index, the label complexity of active learning is seen to be $\widetilde{\Theta}(\min\{1/\mathbb{P}_X([a,b]), 1/\epsilon\} + \log 1/\epsilon)$ when the target hypothesis is $h_{a,b}$ (Dasgupta, 2005). Here the $\widetilde{\Theta}$ notation is used to suppress logarithmic terms.

Example. Suppose $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{H}$ consists of linear separators through the origin. If $\mathbb{P}_X$ is the uniform distribution on the unit sphere, the number of labels needed to learn a hypothesis of error $\leq \epsilon$ is just $\widetilde{\Theta}(d \log 1/\epsilon)$, exponentially smaller than the $\widetilde{O}(d/\epsilon)$ label complexity of supervised learning. If $\mathbb{P}_X$ is not the uniform distribution but is everywhere within a multiplicative

Pool-based active learning

```
Get a set of unlabeled points U ⊂ 𝒳
Repeat until satisfied:
  Pick some x ∈ U to label
Return a hypothesis h ∈ ℋ
```

Stream-based active learning

```
Repeat for t = 0, 1, 2, ...:
  Choose a hypothesis h_t ∈ ℋ
  Receive an unlabeled point x ∈ 𝒳
  Decide whether to query its label
```

Active Learning Theory. Figure 2. Models of pool- and stream-based active learning. The data are draws from an underlying distribution $\mathbb{P}_X$, and hypotheses h are evaluated by $\text{err}_{\mathbb{P}}(h)$. If we want to get this error below ϵ, how many labels are needed, as a function of ϵ?

factor $\lambda > 1$ of it, then the label complexity becomes $\widetilde{O}((d \log 1/\epsilon) \log^2 \lambda)$, provided the amount of unlabeled data is increased by a factor of λ^2 (Dasgupta, 2005).

These results are very encouraging, but the question of an *efficient* active learning algorithm remains open. We now consider two approaches.

Mildly Selective Sampling

The label complexity results mentioned above are based on querying maximally informative points. A less aggressive strategy is to be *mildly selective*, to query all points except those that are quite clearly uninformative. This is the idea behind one of the earliest generic active learning schemes (Cohn, Atlas, & Ladner, 1994). Data points $x_1, x_2, \ldots$ arrive in a stream, and for each one the learner makes a spot decision about whether or not to request a label. When x_t arrives, the learner behaves as follows.

- Determine whether both possible labelings, $(x_t, +)$ and $(x_t, -)$, are consistent with the labeled examples seen so far.
- If so, ask for the label y_t. Otherwise set y_t to be the unique consistent label.

Fortunately, the check required for the first step can be performed efficiently by making two calls to a supervised learner. Thus this is a very simple and elegant active learning scheme, although as one might expect, it is suboptimal in its label complexity (Balcan et al., 2007). Interestingly, there is a parameter called the *disagreement coefficient* that characterizes the label complexity of this scheme and also of some other mildly selective learners (Friedman, 2009; Hanneke, 2007b).

In practice, the biggest limitation of the algorithm above is that it assumes the data are separable. Recent results have shown how to remove this assumption (Balcan, Beygelzimer, & Langford, 2006; Dasgupta et al., 2007) and to accommodate classification loss functions other than 0 – 1 loss (Beygelzimer et al., 2009). Variants of the disagreement coefficient continue to characterize label complexity in the agnostic setting (Beygelzimer et al., 2009; Dasgupta et al., 2007).

A Bayesian Model

The *query by committee* algorithm (Seung, Opper, & Sompolinsky, 1992) is based on a Bayesian view of active learning. The learner starts with a prior distribution on the hypothesis space, and is then exposed to a stream of unlabeled data. Upon receiving x_t, the learner performs the following steps.

- Draw two hypotheses h, h' at random from the posterior over $\mathcal{H}$.
- If $h(x_t) \neq h'(x_t)$ then ask for the label of x_t and update the posterior accordingly.

This algorithm queries points that substantially shrink the posterior, while at the same time taking account of the data distribution. Various theoretical guarantees have been shown for it (Freund, Seung, Shamir, & Tishby, 1997); in particular, in the case of linear separators with a uniform data distribution, it achieves a label complexity of $O(d \log 1/\epsilon)$, the best possible.

Sampling from the posterior over the hypothesis class is, in general, computationally prohibitive. However, for linear separators with a uniform prior, it can be implemented efficiently using random walks on convex bodies (Gilad-Bachrach, Navot, & Tishby, 2005).

Other Work

In this survey, I have touched mostly on active learning results of the greatest generality, those that apply to arbitrary hypothesis classes. There is also a significant body of more specialized results.

- Efficient active learning algorithms for specific hypothesis classes.
  This includes an online learning algorithm for linear separators that only queries some of the points and yet achieves similar regret bounds to algorithms that query all the points (Cesa-Bianchi, Gentile, & Zaniboni, 2004). The label complexity of this method is yet to be characterized.
- Algorithms and label bounds for linear separators under the uniform data distribution.
  This particular setting has been amenable to mathematical analysis. For separable data, it turns out that a variant of the perceptron algorithm achieves the optimal $O(d \log 1/\epsilon)$ label complexity (Dasgupta, Kalai, & Monteleoni, 2005). A simple algorithm is also available for the agnostic setting (Balcan et al., 2007).

Conclusion

The theoretical frontier of active learning is mostly an unexplored wilderness. Except for a few specific cases, we do not have a clear sense of how much active learning can reduce label complexity: whether by just a constant factor, or polynomially, or exponentially. The fundamental statistical and algorithmic challenges involved, together with the huge practical importance of the field, make active learning a particularly rewarding terrain for investigation.

Cross References

►Active Learning

Recommended Reading

Angluin, D. (2001). Queries revisited. In *Proceedings of the 12th international conference on algorithmic learning theory* (pp. 12–31).

Balcan, M.-F., Beygelzimer, A., & Langford, J. (2006). Agnostic active learning. In *International Conference on Machine Learning* (pp. 65–72). New York: ACM Press.

Balcan, M.-F., Broder, A., & Zhang, T. (2007). Margin based active learning. In *Conference on Learning Theory*. pp. 35–50.

Baum, E. B., & Lang, K. (1992). Query learning can work poorly when a human oracle is used. In *International Joint Conference on Neural Networks*.

Beygelzimer, A., Dasgupta, S., & Langford, J. (2009). Importance weighted active learning. In *International Conference on Machine Learning* (pp. 49–56). New York: ACM Press.

Cesa-Bianchi, N., Gentile, C., & Zaniboni, L. (2004). Worst-case analysis of selective sampling for linear-threshold algorithms. *Advances in Neural Information Processing Systems*.

Chernoff, H. (1972). Sequential analysis and optimal design. In *CBMS-NSF Regional Conference Series in Applied Mathematics 8*. SIAM.

Cohn, D., Atlas, L., & Ladner, R. (1994). Improving generalization with active learning. *Machine Learning, 15*(2),201–221.

Dasgupta, S. (2005). Coarse sample complexity bounds for active learning. *Advances in Neural Information Processing Systems*.

Dasgupta, S., Kalai, A., & Monteleoni, C. (2005). Analysis of perceptron-based active learning. In *18th Annual Conference on Learning Theory*. pp. 249–263.

Dasgupta, S., Hsu, D. J., & Monteleoni, C. (2007). A general agnostic active learning algorithm. *Advances in Neural Information Processing Systems*.

Fedorov, V. V. (1972). *Theory of optimal experiments*. (W. J. Studden & E. M. Klimko, Trans.). New York: Academic Press.

Freund, Y., Seung, S., Shamir, E., & Tishby, N. (1997). Selective sampling using the query by committee algorithm. *Machine Learning Journal, 28*,133–168.

Friedman, E. (2009). Active learning for smooth problems. In *Conference on Learning Theory*. pp. 343–352.

Gilad-Bachrach, R., Navot, A., & Tishby, N. (2005). Query by committeee made real. *Advances in Neural Information Processing Systems*.

Hanneke, S. (2007a). Teaching dimension and the complexity of active learning. In *Conference on Learning Theory*. pp. 66–81.

Hanneke, S. (2007b). A bound on the label complexity of agnostic active learning. In *International Conference on Machine Learning*. pp. 353–360.

Haussler, D. (1992). Decision-theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation, 100*(1),78–150.

Seung, H. S., Opper, M., & Sompolinsky, H. (1992). Query by committee. In *Conference on Computational Learning Theory*, pp. 287–294.

Adaboost

Adaboost is an ►ensemble learning technique, and the most well-known of the ►Boosting family of algorithms. The algorithm trains models sequentially, with a new model trained at each round. At the end of each round, mis-classified examples are identified and have their emphasis increased in a new training set which is then fed back into the start of the next round, and a new model is trained. The idea is that subsequent models

should be able to compensate for errors made by earlier models. See ►ensemble learning for full details.

Adaptive Control Processes

►Bayesian Reinforcement Learning

Adaptive Real-Time Dynamic Programming

Andrew G. Barto
University of Massachusetts, Amherst, MA, USA

Synonyms

ARTDP

Definition

Adaptive Real-Time Dynamic Programming (ARTDP) is an algorithm that allows an agent to improve its behavior while interacting over time with an incompletely known dynamic environment. It can also be viewed as a heuristic search algorithm for finding shortest paths in incompletely known stochastic domains. ARTDP is based on ►Dynamic Programming (DP), but unlike conventional DP, which consists of off-line algorithms, ARTDP is an on-line algorithm because it uses agent behavior to guide its computation. ARTDP is adaptive because it does not need a complete and accurate model of the environment but learns a model from data collected during agent-environment interaction. When a good model is available, ►Real-Time Dynamic Programming (RTDP) is applicable, which is ARTDP without the model-learning component.

Motivation and Background

RTDP combines strengths of heuristic search and DP. Like heuristic search – and unlike conventional DP – it does not have to evaluate the entire state space in order to produce an optimal solution. Like DP – and unlike most heuristic search algorithms – it is applicable to nondeterministic problems. Additionally, RTDP's performance as an ►anytime algorithm is better than conventional DP and heuristic search algorithms. ARTDP extends these strengths to problems for which a good model is not initially available.

In artificial intelligence, control engineering, and operations research, many problems require finding a policy (or control rule) that determines how an agent (or controller) should generate actions in response to the states of its environment (the controlled system). When a "cost" or a "reward" is associated with each step of the agent's behavior, policies can be compared according to how much cost or reward they are expected to accumulate over time.

The usual formulation for problems like this in the discrete-time case is the ►Markov Decision Process (MDP). The objective is to find a policy that minimizes (maximizes) a measure of the total cost (reward) over time, assuming that the agent–environment interaction can begin in any of the possible states. In other cases, there is a designated set of "start states" that is much smaller than the entire state set (e.g., the initial board configuration in a board game). In these cases, any given policy only has to be defined for the set of states that can be reached from the starting states when the agent is using that policy. The rest of the states will never arise when that policy is being followed, so the policy does not need to specify what the agent should do in those states.

ARTDP and RTDP exploit situations where the set of states reachable from the start states is a small subset of the entire state space. They can dramatically reduce the amount of computation needed to determine an optimal policy for the relevant states as compared with the amount of computation that a conventional DP algorithm would require to determine an optimal policy for all the states. These algorithms do this by focussing computation around simulated behavioral experiences (if there is a model available capable of simulating these experiences), or around real behavioral experiences (if no model is available).

RTDP and ARTDP were introduced by Barto, Bradtke, and Singh (1995). The starting point was the novel observation by Bradtke that Korf's Learning Real-Time A* heuristic search algorithm (Korf, 1990)

is closely related to DP. RTDP generalizes Learning Real-Time A* to stochastic problems. ARTDP is also closely related to Sutton's Dyna system (Sutton, 1990) and Jalali and Ferguson's (1989) Transient DP. Theoretical analysis relies on the theory of Asnychronous DP as described by Bertsekas and Tsitsiklis (1989).

ARTDP and RTDP are ►model-based reinforcement learning algorithms, so called because they take advantage of an environment model, unlike ►model-free reinforcement learning algorithms such as ►Q-Learning and ►Sarsa.

Structure of Learning System

Backup Operations

A basic step of many DP and RL algorithms is a *backup operation*. This is an operation that updates a current estimate of the *cost* of an MDP's state. (We use the cost formulation instead of reward to be consistent with the original presentation of the algorithms. In the case of rewards, this would be called the *value* of a state and we would maximize instead of minimize.) Suppose X is the set of MDP states. For each state $x \in X$, $f(x)$, the cost of state x, gives a measure (which varies with different MDP formulations) of the total cost the agent is expected to incur over the future if it starts in x. If $f_k(x)$ and $f_{k+1}(x)$, respectively, denote the estimate of $f(x)$ before and after a backup, a typical backup operation applied to x looks like this:

$$f_{k+1}(x) = min_{a \in A}[c_x(a) + \sum_{y \in X} p_{xy}(a) f_k(y)],$$

where A is the set of possible agent actions, $c_x(a)$ is the immediate cost the agent incurs for performing action a in state x, and $p_{xy}(a)$ is the probability that the environment makes a transition from state x to state y as a result of the agent's action a. This backup operation is associated with the DP algorithm known as ►value iteration. It is also the backup operation used by RTDP and ARTDP.

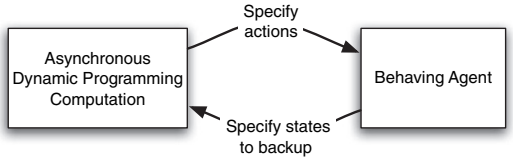
Conventional DP algorithms consist of successive "sweeps" of the state set. Each sweep consists of applying a backup operation to each state. Sweeps continue until the algorithm converges to a solution. Asynchronous DP, which underlies RTDP and ARTDP, does not use systematic sweeps. States can be chosen in any way whatsoever, and as long as backups continue to be applied to all states (and some other conditions are satisfied), the algorithm will converge. RTDP is an instance of asynchronous DP in which the states chosen for backups are determined by the agent's behavior.

The backup operation above is *model-based* because it uses known rewards and transition probabilities, and the values of all the states appear on the right-hand-side of the equation. In contrast, a *sample backup* uses the value of just one sample successor state. RTDP and ARTDP are like RL algorithms in that they rely on real or simulated behavioral experience, but unlike many (but not all) RL algorithms, they use full backups like DP.

Off-Line Versus On-Line

A conventional DP algorithm typically executes off-line. When applied to finding an optimal policy for an MDP, this means that the DP algorithm executes to completion before its result (an optimal policy) is used to control the agent's behavior. The sweeps of DP sequentially "visit" the states of the MDP, performing a backup operation on each state. But it is important not to confuse these visits with the behaving agent's visits to states: the agent is not yet behaving while the off-line DP computation is being done. Hence, the agent's behavior has no influence on the DP computation. The same is true for off-line asynchronous DP.

RTDP is an on-line, or "real-time," algorithm. It is an asynchronous DP computation that executes *concurrently* with the agent's behavior so that the agent's behavior can influence the DP computation. Further, the concurrently executing DP computation can influence the agent's behavior. The agent's visits to states directs the "visits" to states made by the concurrent asynchronous DP computation. At the same time, the action performed by the agent is the action specified by the policy corresponding to the latest results of the DP computation: it is the "greedy" action with respect to the current estimate of the cost function.



In the simplest version of RTDP, when a state is visited by the agent, the DP computation performs the

model-based backup operation given above on that same state. In general, for each step of the agent's behavior, RTDP can apply the backup operation to each of an arbitrary set of states, provided that the agent's current state is included. For example, at each step of behavior, a limited-horizon look-ahead search can be conducted from the agent's current state, with the backup operation applied to each of the states generated in the search. Essentially, RTDP is an asynchronous DP computation with the computational effort focused along simulated or actual behavioral trajectories.

Learning A Model

ARTDP is the same as RTDP except that (1) an environment model is updated using any on-line model-learning, or system identification, method, (2) the current environment model is used in performing the RTDP backup operations, and (3) the agent has to perform exploratory actions occasionally instead of always greedy actions as in RTDP. This last step is essential to ensure that the environment model eventually converges to the correct model. If the state and action sets are finite, the simplest way to learn a model is to keep counts of the number of times each transition occurs for each action and convert these frequencies to probabilities, thus forming the maximum-likelihood model.

Summary of Theoretical Results

When RTDP and ARTDP are applied to *stochastic optimal path problems*, one can prove that under certain conditions they converge to optimal policies without the need to apply backup operations to all the states. Indeed, is some problems, only a small fraction of the states need to be visited. A stochastic optimal path problem is an MDP with a nonempty set of start states and a nonempty set of goal states. Each transition until a goal state is reached has a nonnegative immediate cost, and once the agent reaches a goal state, it stays there and thereafter incurs zero cost. Each episode of agent experience begins with a start state. An optimal policy is one that minimizes the cost of every state, i.e., minimizes $f(x)$ for all states x. Under some relatively mild conditions, every optimal policy is guaranteed to eventually reach a goal state.

A state x is *relevant* if a start state s and an optimal policy exist such that x can be reached from s when the agent uses that policy. If we could somehow know which states are relevant, we could restrict DP to just these states and obtain an optimal policy. But this is not possible because knowing which states are relevant requires knowledge of optimal policies, which is what one is seeking. However, under certain conditions, without requiring repeated visits to all the irrelevant states, RTDP produces a policy that is optimal for all the relevant states. The conditions are that (1) the initial cost of every goal state is zero, (2) there exists at least one policy that guarantees that a goal state will be reached with probability one from any start state, (3) all immediate costs for transitions from non-goal states are strictly positive, and (4) none of the initial costs are larger than the actual costs. This result is proved in Barto et al. (1995) by combining aspects of Korf's (1990) proof for LRTA* with results for asynchronous DP.

Special Cases and Extensions

A number of special cases and extensions of RTDP have been developed that improve performance over the basic version. Some examples are as follows. Bonnet and Geffner's (2003) Labeled RTDP labels states that have already been "solved," allowing faster convergence than RTDP. Feng, Hansen, and Zilberstein (2003) proposed Symbolic RTDP, which selects a set of states to update at each step using symbolic model-checking techniques. The RTDP convergence theorem still applies because this is a special case of RTDP. Smith and Simmons (2006) developed Focused RTDP that maintains a priority value for each state to better direct search and produce faster convergence. Hansen and Zilberstein's (2001) LAO* uses some of the same ideas as RTDP to produce a heuristic search algorithm that can find solutions with loops to non-deterministic heuristic search problems. Many other variants are possible. Extending ARTDP instead of RTDP in all of these ways would produce analogous algorithms that could be used when a good model is not available.

Cross References

►Anytime Algorithm
►Approximate Dynamic Programming
►Reinforcement Learning
►System Identification

Recommended Reading

Barto, A., Bradtke, S., & Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 72*(1–2), 81–138.

Bertsekas, D., & Tsitsiklis, J. (1989). *Parallel and distributed computation: Numerical methods*. Englewood Cliffs, NJ: Prentice-Hall.

Bonet, B., & Geffner, H. (2003a). Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proceedings of the 13th international conference on automated planning and scheduling* (*ICAPS*-2003). Trento, Italy.

Bonet, B., & Geffner, H. (2003b). Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proceedings of the international joint conference on artificial intelligence* (*IJCAI*-2003). Acapulco, Mexico.

Feng, Z., Hansen, E., & Zilberstein, S. (2003). Symbolic generalization for on-line planning. In *Proceedings of the 19th conference on uncertainty in artificial intelligence.* Acapulco, Mexico.

Hansen. E., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence, 129*, 35–62.

Jalali, A., & Ferguson, M. (1989). Computationally efficient control algorithms for Markov chains. In *Proceedings of the 28th conference on decision and control* (pp.1283–1288), Tampa, FL.

Korf, R. (1990). Real-time heuristic search. *Artificial Intelligence, 42*(2–3), 189–211.

Smith, T., & Simmons, R. (2006). Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *Proceedings of the national conference on artificial intelligence (AAAI)*. Boston, MA: AAAI Press.

Sutton, R. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th international conference on machine learning* (pp.216–224). San Mateo, CA: Morgan Kaufmann.

Adaptive Resonance Theory

Gail A. Carpenter, Stephen Grossberg
Boston University, Boston, MA, USA

Synonyms

ART

Definition

Adaptive resonance theory, or ART, is both a cognitive and neural theory of how the brain quickly learns to categorize, recognize, and predict objects and events in a changing world, and a set of algorithms that computationally embody ART principles and that are used in large-scale engineering and technological applications wherein fast, stable, and incremental learning about complex changing environment is needed. ART clarifies the brain processes from which conscious experiences emerge. It predicts a functional link between processes of consciousness, learning, expectation, attention, resonance, and synchrony (CLEARS), including the prediction that "all conscious states are resonant states." This connection clarifies how brain dynamics enable a behaving individual to autonomously adapt in real time to a rapidly changing world. ART predicts how top-down attention works and regulates fast stable learning of recognition categories. In particular, ART articulates a critical role for "resonant" states in driving fast stable learning; and thus the name *adaptive resonance*. These resonant states are bound together, using top-down attentive feedback in the form of learned expectations, into coherent representations of the world. ART hereby clarifies one important sense in which the brain carries out predictive computation. ART has explained and successfully predicted a wide range of behavioral and neurobiological data, including data about human cognition and the dynamics of spiking laminar cortical networks. ART algorithms have been used in large-scale applications such as medical database prediction, remote sensing, airplane design, and the control of autonomous adaptive robots.

Motivation and Background

Many current learning algorithms do not emulate the way in which humans and other animals learn. The power of human and animal learning provides high motivation to discover computational principles whereby machines can learn with similar capabilities. Humans and animals experience the world on the fly, and carry out incremental learning of sequences of episodes in real time. Often such learning is unsupervised, with the world itself as the teacher. Learning can also proceed with an unpredictable mixture of unsupervised and supervised learning trials. Such learning goes on successfully in a world that is nonstationary; that is, the rules of which can change unpredictably through time. Moreover, humans and animals can learn quickly and stably through time. A single important experience can be remembered for a long time. ART proposes a solution of this *stability–plasticity dilemma* (Grossberg, 1980) by

showing how brains learn quickly without forcing catastrophic forgetting of already learned, and still successful, memories.

Thus, ART autonomously carries out fast, yet stable, incremental learning under both unsupervised and supervised learning conditions in response to a complex nonstationary world. In contrast, many current learning algorithms use batch learning in which all the information about the world to be learned is available at a single time. Other algorithms are not defined unless all learning trials are supervised. Yet other algorithms become unstable in a nonstationary world, or become unstable if learning is fast; that is, if an event can be fully learned on a single learning trial. ART overcomes these problems.

Some machine learning algorithms are feed-forward clustering algorithms that undergo catastrophic forgetting in a nonstationary world. The ART solution of the stability–plasticity dilemma depends upon feedback, or top-down, expectations that are matched against bottom-up data and thereby focus attention upon critical feature patterns. A good enough match leads to resonance and fast learning. A big enough mismatch leads to hypothesis testing or memory search that discovers and learns a more predictive category. Thus, ART is a self-organizing expert system that avoids the brittleness of traditional expert systems.

The world is filled with uncertainty, so probability concepts seem relevant to understanding how brains learn about uncertain data. This fact has led some machine learning practitioners to assume that brains obey Bayesian laws. However, the Bayes rule is so general that it can accommodate any system in nature. Additional computational principles and mechanisms must augment Bayes to distinguish a brain from, say, a hydrogen atom or storm. Moreover, probabilistic models often use nonlocal computations. ART shows how the brain embodies a novel kind of real-time probability theory, hypothesis testing, prediction, and decision-making, the local computations of which adapt to a nonstationary world. These ART principles and mechanisms go beyond Bayesian analysis, and are embodied parsimoniously in the laminar circuits of cerebral cortex. Indeed, the cortex embodies a new kind of laminar computing that reconciles the best properties of feedforward and feedback processing, digital and analog processing, and data-driven bottom-up processing combined with hypothesis-driven top-down processing (Grossberg, 2007).

Structure of Learning System

How CLEARS Mechanisms Interact

Humans are *intentional* beings who learn expectations about the world and make predictions about what is about to happen. Humans are also *attentional* beings who focus processing resources upon a restricted amount of incoming information at any time. Why are we both intentional and attentional beings, and are these two types of processes related? The stability–plasticity dilemma and its solution using resonant states provide a unifying framework for understanding these issues.

To clarify the role of sensory or cognitive expectations, and of how a resonant state is activated, suppose you were asked to "find the yellow ball as quickly as possible, and you will win a $10,000 prize." Activating an expectation of a "yellow ball" enables its more rapid detection, and with a more energetic neural response. Sensory and cognitive top-down expectations hereby lead to *excitatory matching* with consistent bottom-up data. Mismatch between top-down expectations and bottom-up data can suppress the mismatched part of the bottom-up data, to focus attention upon the matched, or expected, part of the bottom-up data.

Excitatory matching and attentional focusing on bottom-up data using top-down expectations generates resonant brain states: When there is a good enough match between bottom-up and top-down signal patterns between two or more levels of processing, their positive feedback signals amplify and prolong their mutual activation, leading to a resonant state. Amplification and prolongation of activity triggers learning in the more slowly varying adaptive weights that control the signal flow along pathways from cell to cell. Resonance hereby provides a global context-sensitive indicator that the system is processing data worthy of learning, hence the name *adaptive* resonance theory.

In summary, ART predicts a link between the mechanisms which enable us to learn quickly and stably about a changing world, and the mechanisms that enable us to learn expectations about such a world, test hypotheses about it, and focus attention upon information that we find interesting. ART clarifies this link by asserting that to solve the stability–plasticity

dilemma, only resonant states can drive rapid new learning.

It is just a step from here to propose that those experiences which can attract our attention and guide our future lives by being learned are also among the ones that are conscious. Support for this additional assertion derives from the many modeling studies whose simulations of behavioral and brain data using resonant states map onto properties of conscious experiences in those experiments.

The type of learning within the sensory and cognitive domain that ART mechanizes is *match learning*: Match learning occurs only if a good enough match occurs between bottom-up information and a learned top-down expectation that is read out by an active recognition category, or code. When such an approximate match occurs, previously learned knowledge can be refined. Match learning raises the concern about what happens if a match is not good enough? How does such a model escape perseveration on already learned representations?

If novel information cannot form a good enough match with the expectations that are read-out by previously learned recognition categories, then a memory search or hypothesis testing is triggered, which leads to selection and learning of a new recognition category, rather than catastrophic forgetting of an old one. Figure 1 illustrates how this happens in an ART model; it is discussed in great detail below. In contrast, learning within spatial and motor processes is proposed to be *mismatch learning* that continuously updates sensory-motor maps or the gains of sensory-motor commands. As a result, we can stably learn what is happening in a changing world, thereby solving the stability–plasticity dilemma, while adaptively updating our representations of where objects are and how to act upon them using bodies whose parameters change continuously through time. Brain systems that use inhibitory matching and mismatch learning cannot generate resonances; hence, their representations are not conscious.

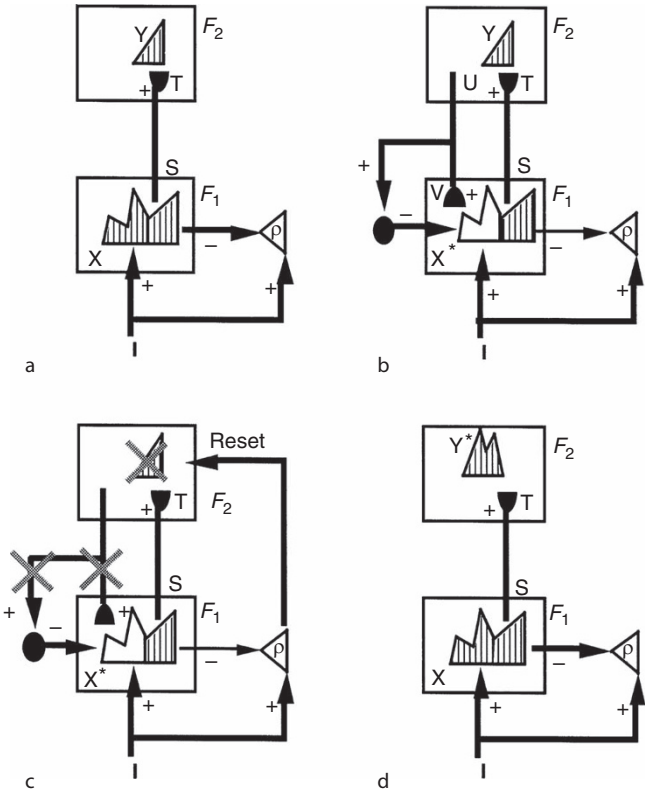
Complementary Computing in the Brain: Resonance and Reset

It has been mathematically proved that match learning within an ART model leads to stable memories in response to arbitrary list of events to be learned (e.g., Carpenter & Grossberg, 1987). However, match learning also has a serious potential weakness: If you can only learn when there is a good match between bottom-up data and learned top-down expectations, then how do you ever learn anything that you do not already know? ART proposes that this problem is solved by the brain by using an interaction between complementary processes of *resonance* and *reset,* which are predicted to control properties of attention and memory search, respectively. These complementary processes help our brains to balance between the complementary demands of processing the familiar and the unfamiliar, the expected and the unexpected.

Organization of the brain into complementary processes is predicted to be a general principle of brain design that is not just found in ART (Grossberg, 2000). A complementary process can individually compute some properties well, but cannot, by itself, process other complementary properties. In thinking intuitively about complementary properties, one can imagine puzzle pieces fitting together. Both pieces are needed to finish the puzzle. Complementary brain processes are more dynamic than any such analogy: Pairs of complementary processes interact to form emergent properties which overcome their complementary deficiencies to compute complete information with which to represent or control some aspect of intelligent behavior.

The resonance process in the complementary pair of resonance and reset is predicted to take place in the What cortical stream, notably in the inferotemporal and prefrontal cortex. Here top-down expectations are matched against bottom-up inputs. When a top-down expectation achieves a good enough match with bottom-up data, this match process focuses attention upon those feature clusters in the bottom-up input that are expected. If the expectation is close enough to the input pattern, then a state of resonance develops as the attentional focus takes hold.

Figure 1 illustrates these ART ideas in a simple two-level example. Here, a bottom-up input pattern, or vector, **I** activates a pattern X of activity across the feature detectors of the first level F_1. For example, a visual scene may be represented by the features comprising its boundary and surface representations. This feature pattern represents the relative importance of different features in the inputs pattern I. In Fig. 1a, the



Adaptive Resonance Theory. Figure 1. Search for a recognition code within an ART learning circuit: (a) The input pattern I is instated across the feature detectors at level F_1 as a short term memory (STM) activity pattern X. Input I also nonspecifically activates the orienting system with a gain that is called vigilance (ρ); that is, all the input pathways converge with gain ρ onto the orienting system and try to activate it. STM pattern X is represented by the hatched pattern across F_1. Pattern X both inhibits the orienting system and generates the output pattern *S*. Pattern *S* is multiplied by learned adaptive weights, also called long term memory (LTM) traces. These LTM-gated signals are added at F_2cells, or nodes, to form the input pattern *T*, which activates the STM pattern Y across the recognition categories coded at level F_2. (b) Pattern *Y* generates the top-down output pattern *U* which is multiplied by top-down LTM traces and added at F_1 nodes to form a *prototype* pattern *V* that encodes the learned expectation of the active F_2 nodes. Such a prototype represents the set of commonly shared features in all the input patterns capable of activating *Y*. If V mismatches I at F_1, then a new STM activity pattern X^* is selected *at* F_1. X^* is represented by the hatched pattern. It consists of the features of I that are confirmed by *V*. Mismatched features are inhibited. The inactivated nodes corresponding to unconfirmed features of X are unhatched. The reduction in total STM activity which occurs when X is transformed into X^* causes a decrease in the total inhibitionfrom F_1 to the orienting system. (c) If inhibition decreases sufficiently, the orienting system releases a nonspecific arousal wave to F_2; that is, a wave of activation that equally activates all F_2 nodes. This wave instantiates the intuition that "novel events are arousing." This arousal wave resets the STM pattern Y at F_2 by inhibiting *Y*. (d) After Y is inhibited, its top-down prototype signal is eliminated, and X can be reinstated at F_1. The prior reset event maintains inhibition of *Y* during the search cycle. As a result, X can activate a different STM pattern Y at F_2. If the top-down prototype due to this new *Y* pattern also mismatches I at F_1, then the search for an appropriate F_2 code continues until a more appropriate F_2 representation is selected. Such a search cycle represents a type of nonstationary hypothesis testing. When search ends, an attentive resonance develops and learning of the attended data is initiated (adapted with permission from Carpenter and Grossberg (1993)). The distributed ART architecture supports fast stable learning with arbitrarily distributed F_2 codes (Carpenter, 1997)

pattern peaks represent more activated feature detector cells, and the troughs, less-activated feature detectors. This feature pattern sends signals S through an adaptive filter to the second level F_2 at which a compressed representation Y (also called a recognition category, or a symbol) is activated in response to the distributed input T. Input T is computed by multiplying the signal vector S by a matrix of adaptive weights that can be altered through learning. The representation Y is compressed by competitive interactions across F_2 that allow only a small subset of its most strongly activated cells to remain active in response to T. The pattern Y in the figure indicates that a small number of category cells may be activated to different degrees. These category cells, in turn, send top-down signals U to F_1. The vector U is converted into the top-down expectation V by being multiplied by another matrix of adaptive weights. When V is received by F_1, a matching process takes place between the input vector **I** and **V** which selects that subset X* of F_1 features that were "expected" by the active F_2 category Y. The set of these selected features is the emerging "attentional focus."

Binding Distributed Feature Patterns and Symbols During Conscious Resonances

If the top-down expectation is close enough to the bottom-up input pattern, then the pattern X^* of attended features reactivates the category Y which, in turn, reactivates X^*. The network hereby locks into a resonant state through a positive feedback loop that dynamically links, or binds, the attended features across X^* with their category, or symbol, Y.

Resonance itself embodies another type of complementary processing. Indeed, there seem to be complementary processes both within and between cortical processing streams (Grossberg, 2000). This particular complementary relation occurs between distributed feature patterns and the compressed categories, or symbols, that selectively code them:

Individual features at F_1 have no meaning on their own, just like the pixels in a picture are meaningless one-by-one. The category, or symbol, in F_2 is sensitive to the global patterning of these features, and can selectively fire in response to this pattern. But it cannot represent the "contents" of the experience, including their conscious qualia, due to the very fact that a category is a compressed or "symbolic" representation. Practitioners of artificial intelligence have claimed that neural models can process distributed features, but not symbolic representations. This is not, of course, true in the brain. Nor is it true in ART.

Resonance between these two types of information converts the *pattern* of attended features into a coherent context-sensitive state that is linked to its category through feedback. This coherent state, which binds together distributed features and symbolic categories, can enter consciousness while it binds together spatially distributed features into either a stable equilibrium or a synchronous oscillation. The original ART article (Grossberg, 1976) predicted the existence of such synchronous oscillations, which were there described in terms of their mathematical properties as "order-preserving limit cycles." See Carpenter, Grossberg, Markuzon, Reynolds & Rosen (1992) and Grossberg & Versace (2008) for reviews of confirmed ART predictions, including predictions about synchronous oscillations.

Resonance Links Intentional and Attentional Information Processing to Learning

In ART, the resonant state, rather than bottom-up activation, is predicted to drive learning. This state persists long enough, and at a high enough activity level, to activate the slower learning processes in the adaptive weights that guide the flow of signals between bottom-up and top-down pathways between levels F_1 and F_2 in Fig. 1. This viewpoint helps to explain how adaptive weights that were changed through previous learning can regulate the brain's present information processing, without learning about the signals that they are currently processing unless they can initiate a resonant state. Through resonance as a mediating event, one can understand from a deeper mechanistic view why humans are intentional beings who are continually predicting what may next occur, and why we tend to learn about the events to which we pay attention.

More recent versions of ART, notably the synchronous matching ART (SMART) model (Grossberg & Versace, 2008) show how a match may lead to fast gamma oscillations that facilitate spike-timing dependent plasticity (STDP), whereas mismatch can lead to slower beta oscillations that lower the probability that mismatched events can be learned by a STDP learning law.

Complementary Attentional and Orienting Systems Control Resonance Versus Reset

A sufficiently bad mismatch between an active top-down expectation and a bottom-up input, say because the input represents an unfamiliar type of experience, can drive a memory search. Such a mismatch within the attentional system is proposed to activate a complementary *orienting system*, which is sensitive to unexpected and unfamiliar events. ART suggests that this orienting system includes the nonspecific thalamus and the hippocampal system. See Grossberg & Versace (2008) for a summary of data supporting this prediction. Output signals from the orienting system rapidly reset the recognition category that has been reading out the poorly matching top-down expectation (Figs. 1b and c). The cause of the mismatch is hereby removed, thereby freeing the system to activate a different recognition category (Fig. 1d). The reset event hereby triggers memory search, or hypothesis testing, which automatically leads to the selection of a recognition category that can better match the input.

If no such recognition category exists, say because the bottom-up input represents a truly novel experience, then the search process automatically activates an as yet uncommitted population of cells, with which to learn about the novel information. In order for a top-down expectation to match a newly discovered recognition category, its top-down adaptive weights initially have large values, which are pruned by the learning of a particular expectation.

This learning process works well under both unsupervised and supervised conditions (Carpenter et al., 1992). Unsupervised learning means that the system can learn how to categorize novel input patterns without any external feedback. Supervised learning uses predictive errors to let the system know whether it has categorized the information correctly. Supervision can force a search for new categories that may be culturally determined, and are not based on feature similarity alone. For example, separating the letters E and F that are of similar features into separate recognition categories is culturally determined. Such error-based feedback enables variants of E and F to learn their own category and top-down expectation, or prototype. The complementary, but interacting, processes of attentive-learning and orienting-search together realize a type of error correction through hypothesis testing that can build an ever-growing, self-refining internal model of a changing world.

Controlling the Content of Conscious Experiences: Exemplars and Prototypes

What combinations of features or other information are bound together into conscious object or event representations? One view is that exemplars or individual experiences are learned because humans can have very specific memories. For example, we can all recognize the particular faces of our friends. On the other hand, storing every remembered experience as exemplars can lead to a combinatorial explosion of memory, as well as to unmanageable problems of memory retrieval. A possible way out is suggested by the fact that humans can learn prototypes which represent general properties of the environment (Posner & Keele, 1968). For example, we can recognize that everyone has a face. But then how do we learn specific episodic memories? ART provides an answer that overcomes the problems faced by earlier models.

ART prototypes are not merely averages of the exemplars that are classified by a category, as is typically assumed in classical prototype models. Rather, they are the actively selected *critical feature patterns* upon which the top-down expectations of the category focus attention. In addition, the generality of the information that is codes by these critical feature patterns is controlled by a gain control process, called *vigilance* control, which can be influenced by environmental feedback or internal volition (Carpenter & Grossberg, 1987). Low vigilance permits the learning of general categories with abstract prototypes. High vigilance forces a memory search to occur for a new category when even small mismatches exist between an exemplar and the category that it activates. As a result, in the limit of high vigilance, the category prototype may encode an individual exemplar.

Vigilance is computed within the orienting system of an ART model (Fig. 1b–d). It is here that bottom-up excitation from all the active features in an input pattern I is compared with inhibition from all the active features in a distributed feature representation across F_1. If the ratio of the total activity across the active features in F_1 (i.e., the "matched" features) to the total activity of all the features in I is less than a *vigilance parameter* ρ (Fig. 1b), then a reset wave is activated (Fig. 1c), which

can drive the search for another category to classify the exemplar. In other words, the vigilance parameter controls how bad a match can be tolerated before search for a new category is initiated. If the vigilance parameter is low, then many exemplars can influence the learning of a shared prototype, by chipping away at the features that are not shared with all the exemplars. If the vigilance parameter is high, then even a small difference between a new exemplar and a known prototype (e.g., F vs. E) can drive the search for a new category with which to represent F.

One way to control vigilance is by a process of *match tracking*. Here a predictive error (e.g., D is predicted in response to F), the vigilance parameter increases until it is just higher than the ratio of active features in F_1 to total features in I. In other words, vigilance "tracks" the degree of match between input exemplar and matched prototype. This is the minimal level of vigilance that can trigger a reset wave and thus a memory search for a new category. Match tracking realizes a minimax learning rule that conjointly *maximizes* category generality while it *minimizes* predictive error. In other words, match tracking uses the least memory resources that can prevent errors from being made.

Because vigilance can vary across learning trials, recognition categories capable of encoding widely differing degrees of generalization or abstraction can be learned by a single ART system. Low vigilance leads to broad generalization and abstract prototypes. High vigilance leads to narrow generalization and to prototypes that represent fewer input exemplars, even a single exemplar. Thus a single ART system may be used, say, to learn abstract prototypes with which to recognize abstract categories of faces and dogs, as well as "exemplar prototypes" with which to recognize individual views of faces and dogs. ART models hereby try to learn the most general category that is consistent with the data. This tendency can, for example, lead to the type of overgeneralization that is seen in young children until further learning leads to category refinement.

Memory Consolidation and the Emergence of Rules: Direct Access to Globally Best Match

As sequences of inputs are practiced over learning trials, the search process eventually converges upon stable categories. It has been mathematically proved (Carpenter & Grossberg, 1987) that familiar inputs directly access the category whose prototype provides the best match globally, while unfamiliar inputs engage the orienting subsystem to trigger memory searches for better categories until they become familiar. This process continues until the memory capacity, which can be chosen arbitrarily large, is fully utilized. The process whereby search is automatically disengaged is a form of *memory consolidation* that emerges from network interactions. Emergent consolidation does not preclude structural consolidation at individual cells, since the amplified and prolonged activities that subserve a resonance may be a trigger for learning-dependent cellular processes, such as protein synthesis and transmitter production.

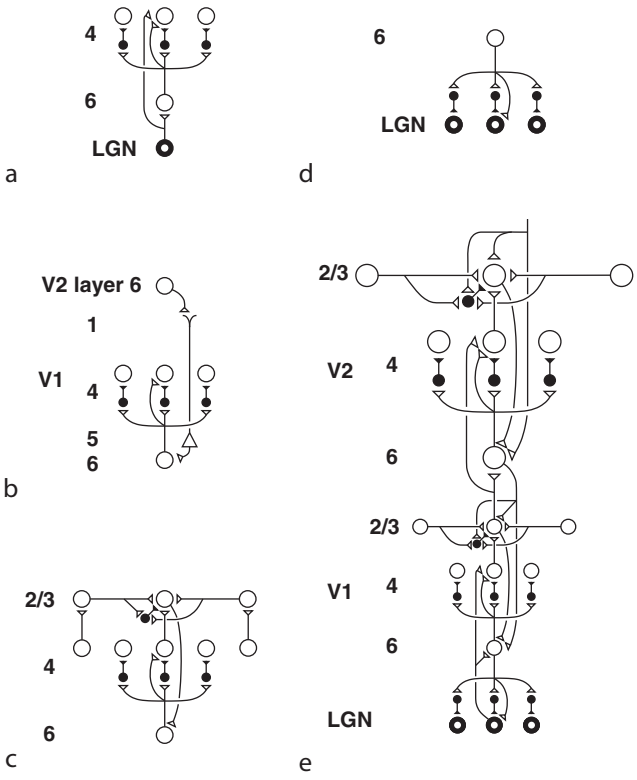
It has also been shown that the adaptive weights which are learned by some ART models can, at any stage of learning, be translated into fuzzy IF-THEN rules (Carpenter et al., 1992). Thus the ART model is a self-organizing rule-discovering production system as well as a neural network. These examples show that the claims of some cognitive scientists and AI practitioners that neural network models cannot learn rule-based behaviors are as incorrect as the claims that neural models cannot learn symbols.

How the Laminar Circuits of Cerebral Cortex Embody ART Mechanisms

More recent versions of ART have shown how predicted ART mechanisms may be embodied within known laminar microcircuits of the cerebral cortex. These include the family of LAMINART models (Fig. 2; see Raizada & Grossberg, 2003) and the synchronous matching ART, or SMART, model (Fig. 3; see Grossberg & Versace, 2008). SMART, in particular, predicts how a top-down match may lead to fast gamma oscillations that facilitate spike-timing dependent plasticity (STDP), whereas a mismatch can lead to slower beta oscillations that prevent learning by a STDP learning law. At least three neurophysiological labs have recently reported data consistent with the SMART prediction.

Review of ART and ARTMAP Algorithms

From Winner-Take-All to Distributed Coding As noted above, ART networks serve both as models of human cognitive information processing (Carpenter, 1997;



Adaptive Resonance Theory. Figure 2. LAMINART circuit clarifies how known cortical connections within and across cortical layers join the layer 6 → 4 and layer 2/3 circuits to form a laminar circuit model for the interblobs and pale stripe regions of cortical areas V1 and V2. Inhibitory interneurons are shown filled-in black. (a) The LGN provides bottom-up activation to layer 4 via two routes. First, it makes a strong connection directly into layer 4. Second, LGN axons send collaterals into layer 6, and thereby also activate layer 4 via the 6 → 4 on-center off-surround path. The combined effect of the bottom-up LGN pathways is to stimulate layer 4 via an on-center off-surround, which provides divisive contrast normalization (Grossberg, 1980) of layer 4 cell responses. (b)Folded feedback carries attentional signals from higher cortex into layer 4 of V1, via the modulatory 6 → 4 path. Corticocortical feedback axons tend preferentially to originate in layer 6 of the higher area and to terminate in layer 1 of the lower cortex, where they can excite the apical dendrites of layer 5 pyramidal cells whose axons send collaterals into layer 6. The triangle in the figure represents such a layer 5 pyramidal cell. Several other routes through which feedback can pass into V1 layer 6 exist. Having arrived in layer 6, the feedback is then "folded" back up into the feedforward stream by passing through the 6 → 4 on-center off-surround path (Bullier, Hup'e, James, & Girard, 1996). (c)Connecting the 6 → 4 on-centeroff-surround to the layer 2/3 grouping circuit: like-oriented layer 4 simple cells with opposite contrast polarities compete (not shown) before generating half-wave rectified outputs that converge onto layer 2/3 complex cells in the column above them. Just like attentional signals from higher cortex, as shown in (b), groupings that form within layer 2/3 also send activation into the folded feedback path, to enhance their own positions in layer 4 beneath them via the 6 → 4 on-center, and to suppress input to other groupings via the 6 → 4 off-surround. There exist direct layer 2/3 → 6 connections in macaque V1, as well as indirect routes via layer 5. (d) Top-down corticogeniculate feedback from V1 layer 6 to LGN also has an on-center off-surround anatomy, similar to the 6 → 4 path. The on-center feedback selectively enhances LGN cells that are consistent with the activation that they cause (Sillito, Jones, Gerstein, & West, 1994), and the off-surround contributes to length-sensitive (endstopped) responses that facilitate grouping perpendicular to line ends. (e) The entire V1/V2 circuit: V2 repeats the laminar pattern of V1 circuitry, but at a larger spatial scale. In particular, the horizontal layer 2/3 connections have a longer range in V2, allowing above-threshold perceptual groupings between more widely spaced

Grossberg, 1999, 2003) and as neural systems for technology transfer (Caudell, Smith, Escobedo, & Anderson, 1994; Parsons & Carpenter, 2003). Design principles derived from scientific analyses and design constraints imposed by targeted applications have jointly guided the development of many variants of the basic networks, including fuzzy ARTMAP (Carpenter et al., 1992), ART-EMAP, ARTMAP-IC, and Gaussian ARTMAP. Early ARTMAP systems, including fuzzy ARTMAP, employ *winner-take-all (WTA) coding*, whereby each input activates a single category node during both training and testing. When a node is first activated during training, it is mapped to its designated output class.

Starting with ART-EMAP, subsequent systems have used *distributed coding* during testing, which typically improves predictive accuracy, while avoiding the computational problems inherent in the use of distributed code representations during training. In order to address these problems, distributed ARTMAP (Carpenter, 1997; Carpenter, Milenova, & Noeske, 1998) introduced a new network configuration, in addition to new learning laws.

Comparative analysis of the performance of ARTMAP systems on a variety of benchmark problems has led to the identification of a *default ARTMAP* network, which features simplicity of design and robust performance in many application domains. Default ARTMAP employs winner-take-all coding during training and distributed coding during testing within a distributed ARTMAP network architecture. With winner-take-all coding during testing, default ARTMAP reduces to a version of fuzzy ARTMAP.

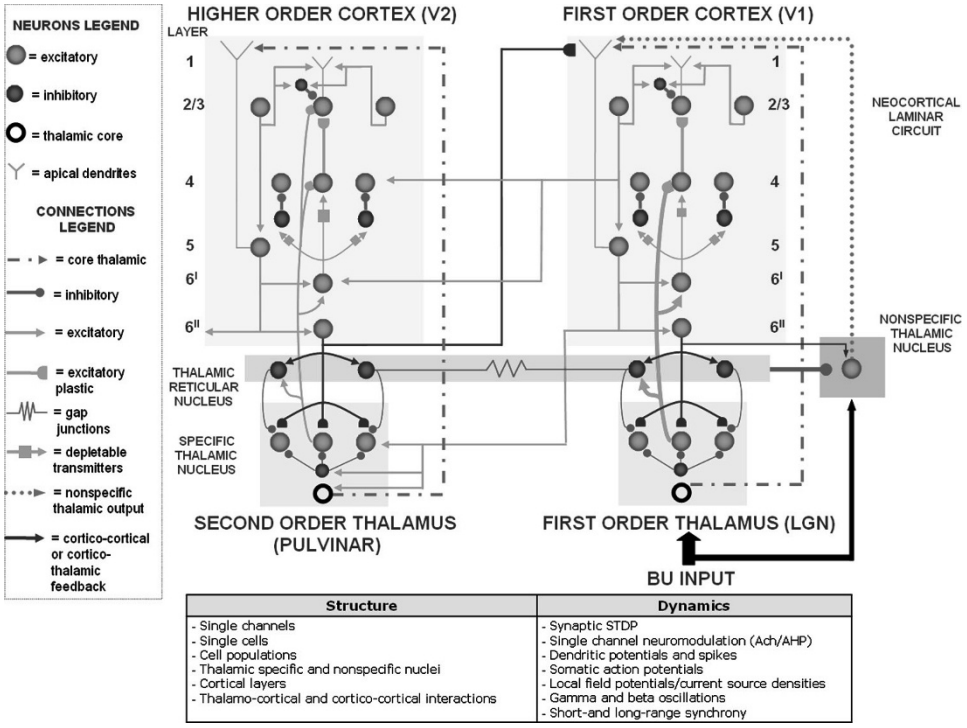
Complement Coding: Learning both Absent and Present Features ART and ARTMAP employ a preprocessing step called *complement coding* (Fig. 4), which models the nervous system's ubiquitous use of the computational design known as *opponent processing*. Balancing an entity against its opponent, as in agonist–antagonist muscle pairs, allows a system to act upon relative quantities, even as absolute magnitudes may vary unpredictably. In ART systems, complement coding is analogous to retinal ON-cells and OFF-cells. When the learning system is presented with a set of input features $\mathbf{a} \equiv (a_1 ... a_i ... a_M)$, complement coding doubles the number of input components, presenting to the network both the original feature vector and its complement.

Complement coding allows an ART system to encode within its critical feature patterns of memory features that are consistently *absent* on an equal basis with features that are consistently *present*. Features that are sometimes absent and sometimes present when a given category is learning are regarded as uninformative with respect to that category. Since its introduction, complement coding has been a standard element of ART and ARTMAP networks, where it plays multiple computational roles, including input normalization. However, this device is not particular to ART, and could, in principle, be used to preprocess the inputs to any type of system.

To implement complement coding, component activities a_i of a feature vector $\mathbf{a}$ are scaled; thus, $0 \leq a_i \leq 1$. For each feature i, the ON activity a_i determines the complementary OFF activity $(1 - a_i)$. Both a_i and $(1 - a_i)$ are represented in the $2M$-dimensional system input vector $\mathbf{A} = (\mathbf{a} \mid \mathbf{a}^c)$ (Fig. 4). Subsequent network computations then operate in this $2M$-dimensional input space. In particular, learned weight vectors $\mathbf{w}_J$ are $2M$-dimensional.

ARTMAP Search and Match Tracking in Fuzzy ARTMAP As illustrated by Fig. 1, the ART matching process triggers either learning or a parallel memory search. If search ends at an established code, the memory

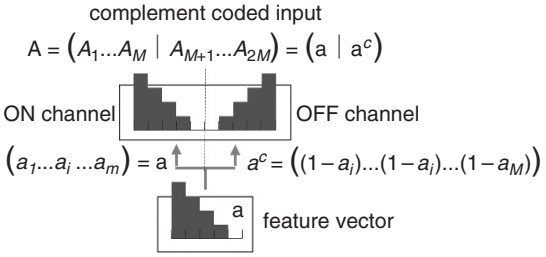
inducing stimuli to form. V1 layer 2/3 projects up to V2 layers 6 and 4, just as LGN projects to layers 6 an 4 of V1. Higher cortical areas send feedback into V2 which ultimately reaches layer 6, just as V2 feedback acts on layer 6 of V1. Feedback paths from higher cortical areas straight into V1 (not shown) can complement and enhance feedback from V2 into V1. Top-down attention can also modulate layer 2/3 pyramidal cells directly by activating both the pyramidal cells and inhibitory interneurons in that layer. The inhibition tends to balance the excitation, leading to a modulatory effect. These top-down attentional pathways tend to synapse in layer 1, as shown in Fig. 2b. Their synapses on apical dendrites in layer 1 are not shown, for simplicity. (Reprinted with permission from Raizada & Grossberg (2003))



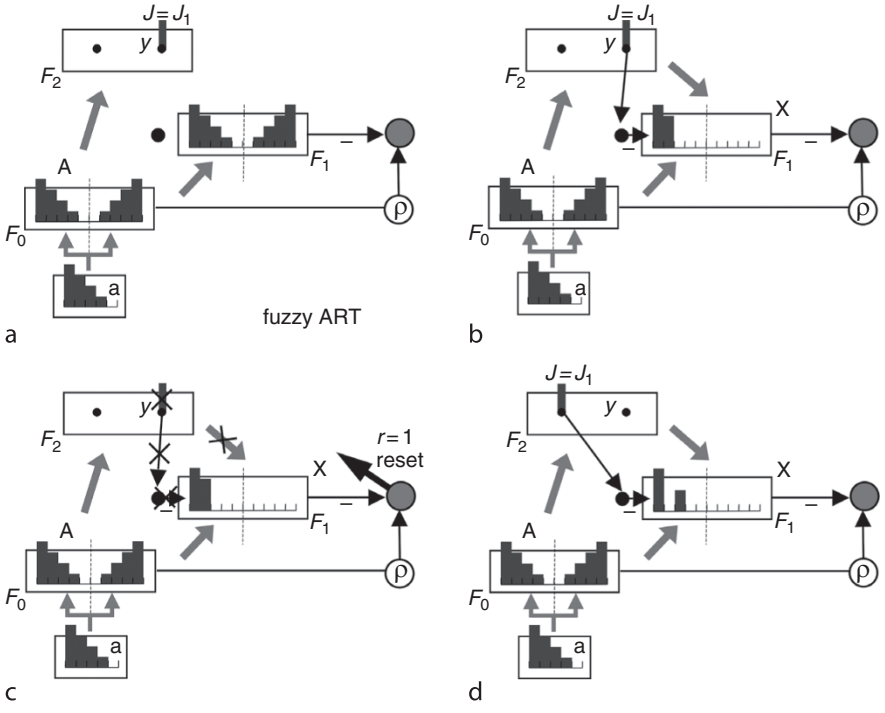
Adaptive Resonance Theory. Figure 3. SMART model overview. A first-order and higher-order cortical area are linked by corticocortical and corticothalamocortical connections. The thalamus is subdivided into specific first-order, second-order, nonspecific, and thalamic reticular nucleus (TRN). The thalamic matrix (one cell population shown as an open ring) provides priming to layer 1, where layer 5 pyramidal cell apical dendrites terminate. The specific thalamus relays sensory information (first-order thalamus) or lower-order cortical information (second-order thalamus) to the respective cortical areas via plastic connections. The nonspecific thalamic nucleus receives convergent BU input and inhibition from the TRN, and projects to layer 1 of the laminar cortical circuit, where it regulates reset and search in the cortical circuit (see text). Corticocortical feedback connections link layer 6^{II} of the higher cortical area to layer 1 of the lower cortical area, whereas thalamocortical feedback originates in layer 6^{II} and terminates in the specific thalamus after synapsing on the TRN. Layer 6^{II} corticothalamic feedback matches the BU input in the specific thalamus. V1 receives two parallel BU thalamocortical pathways. The LGN→V1 layer 4 pathway and the modulatory LGN→V1 layer 6^{I}→4 pathway provide divisive contrast normalization of layer 4 cell responses. The intracortical loop V1 layer 4→2/3→5→6^{I}→4 pathway (*folded feedback)* enhances the activity of winning layer 2/3 cells at their own positions via the 6^{I}→4 on-center, and suppresses input to other layer 2/3 cells via the 6^{I}→4 off-surround. V1 also activates the BU V1→V2 corticocortical pathways (V1 layer 2/3→V2 layers 6^{I} and 4) and the BU corticothalamocortical pathways (V1 layer 5→PULV→V2 layers 6^{I} and 4), where the layer 6^{I}→4 pathway provides divisive contrast normalization to V2 layer 4 cells analogously to V1. Corticocortical feedback from V2 layer 6^{II}→V1 layer 5→6^{I}→4 also uses the same modulatory 6^{I}→4 pathway. TRN cells of the two thalamic sectors are linked via gap junctions, which provide synchronization of the two thalamocortical sectors when processing BU stimuli (reprinted with permission from Grossberg & Versace (2008))

representation may either remain the same or incorporate new information from matched portions of the current input. While this dynamic applies to arbitrarily distributed activation patterns, the F_2 search and code for fuzzy ARTMAP (Fig. 5) describes a winner-take all system.

Before ARTMAP makes a class prediction, the bottom-up input **A** is matched against the top-down



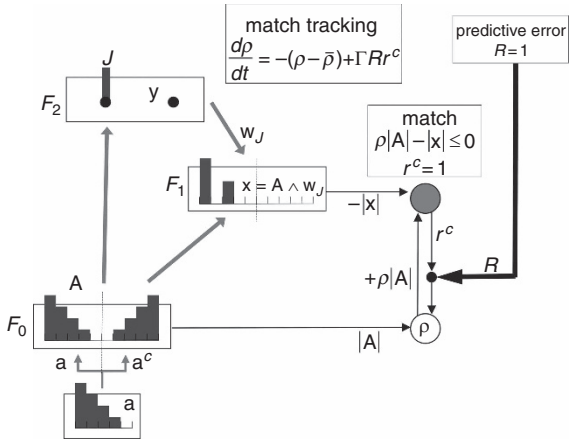
Adaptive Resonance Theory. Figure 4. Complement coding transforms an *M*-dimensional feature vector a into a 2*M*-dimensional system input vector A. A complement-coded system input represents both the degree to which a feature *i* is present (a_i) and the degree to which that feature is absent $(1 - a_i)$



Adaptive Resonance Theory. Figure 5. A fuzzy ART search cycle, with a distributed ART network configuration (Carpenter, 1997). The ART 1 search cycle (Carpenter and Grossberg, 1987) is the same, but allows only binary inputs and did not originally feature complement coding. The match field F_1 represents the matched activation pattern $\mathbf{x} = \mathbf{A} \wedge \mathbf{w}_J$, where $\wedge$ denotes the component-wise minimum, or fuzzy intersection, between the bottom-up input $\mathbf{A}$ and the top-down expectation $\mathbf{w}_J$. If the matched pattern fails to meet the matching criterion, then the active code is reset at F_2, and the system searches for another code y that better represents the input. The match/mismatch decision takes place in the ART orienting system. Each active feature in the input pattern $\mathbf{A}$ excites the orienting system with gain equal to the vigilance parameter ρ. Hence, with complement coding, the total excitatory input is $\rho|\mathbf{A}| = \rho\sum_{i=1}^{2M}\mathbf{A}_i = \rho M$. Active cells in the matched pattern x inhibit the orienting system, leading to a total inhibitory input equal to $-|\mathbf{x}| = -\sum_{i=1}^{2M} x_i$. If $\rho|\mathbf{A}| - |\mathbf{x}| \leq 0$, then the orienting system remains quiet, allowing resonance and learning to occur. If $\rho|\mathbf{A}| - |\mathbf{x}| > 0$, then the reset signal $r = 1$, initiating search for a better matching code

learned expectation, or critical feature pattern, that is read out by the active node (Fig. 5b). The matching criterion is set by a *vigilance* parameter ρ. As noted above, low vigilance permits the learning of abstract, prototype-like patterns, while high vigilance requires the learning of specific, exemplar-like patterns. When a new input arrives, vigilance equals a baseline level $\bar{\rho}$. Baseline vigilance is set equal to zero by default, in order to maximize generalization. Vigilance rises only after the system has made a predictive error. The internal control process that determines how far it must rise in order to correct the error is called *match tracking*. As vigilance rises, the network is required to pay more attention to how well top-down expectations match the current bottom-up input.

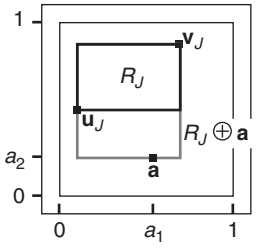
Match tracking (Fig. 6) forces an ARTMAP system not only to reset its mistakes, but to learn from them. With match tracking and fast learning, each ARTMAP network passes the *next input test*, which requires that, if a training input were re-presented immediately after a learning trial, it would directly activate the correct output class, with no predictive errors or search. Match tracking thus simultaneously implements the design goals of maximizing generalization and minimizing predictive error, without requiring the choice of a fixed matching criterion. ARTMAP memories thereby include both broad and specific pattern classes, with the latter typically formed as exceptions to the more general "rules" defined by the former. ARTMAP learning typically produces a wide variety of such mixtures, whose exact composition depends upon the order of training exemplar presentation.



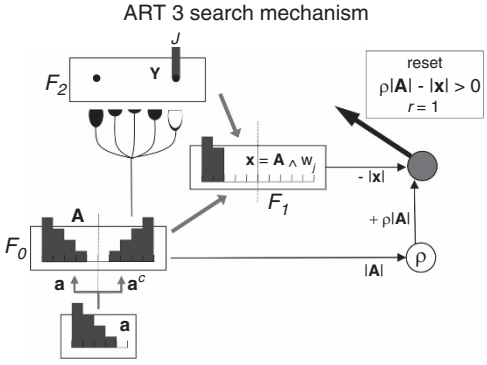
Adaptive Resonance Theory. Figure 6. ARTMAP match tracking. When an active node *J* meets the matching criterion ($\rho\,|\mathbf{A}| - |\mathbf{x}| \le 0$), the reset signal $r = 0$ and the node makes an prediction. If the predicted output is incorrect, the feedback signal $R = 1$. While $R = r^c = 1$, r increases rapidly. As soon as $\rho > \frac{|\mathbf{x}|}{|\mathbf{A}|}$, *r* switches to 1, which both halts the increase of r and resets the active F_2 node. From one chosen node to the next, *r* decays to slightly below $\frac{|\mathbf{x}|}{|\mathbf{A}|}$ (MT–). On the time scale of learning *r* returns to $\bar{\rho}$

Unless they have already activated all their coding nodes, ARTMAP systems contain a reserve of nodes that have never been activated, with weights at their initial values. These *uncommitted* nodes compete with the previously active *committed* nodes, and an uncommitted node is chosen over poorly matched committed nodes. An ARTMAP design constraint specifies that an active uncommitted node should not reset itself. Weights initially begin with $w_{iJ} = 1$. Thus, when the active node J is uncommitted, $\mathbf{x} = \mathbf{A} \wedge \mathbf{w}_J = \mathbf{A}$ at the match field. Then, $\rho\,|\mathbf{A}| - |\mathbf{x}| = \rho\,|\mathbf{A}| - |\mathbf{A}| = (\rho - 1)\,|\mathbf{A}|$. Thus $\rho\,|\mathbf{A}| - |\mathbf{x}| \le 0$ and an uncommitted node does not trigger a reset, provided $\rho \le 1$.

ART Geometry Fuzzy ART long-term memories are visualized as hyper-rectangles, called *category boxes*. The weight vector $\mathbf{w}_J$ is interpreted geometrically as a box R_J whose ON-channel corner $\mathbf{u}_J$ and OFF-channel corner $\mathbf{v}_J$ are, in the format of the complement-coded input vector, defined by $\left(\mathbf{u}_J \,\middle|\, \mathbf{v}_J^C\right) \equiv \mathbf{w}_J$ (Fig. 7). For fuzzy ART with the choice-by-difference $F_0 \to F_2$ signal function T_J, an input $\mathbf{a}$ activates the node J of the closest category box R_J, according to the L_1 (city-block) metric. In case of a tie, as when $\mathbf{a}$ lies in more than one box, the node with the smallest R_J is chosen, where $|R_J|$ is defined as the sum of the edge lengths $\sum_{i=1}^{M} |v_{iJ} - u_{iJ}|$. The chosen node J will reset if $|R_J \oplus \mathbf{a}| > M\,(1 - \rho)$, where $R_J \oplus \mathbf{a}$ is the smallest box enclosing both R_J and $\mathbf{a}$. Otherwise, R_J expands toward $R_J \oplus \mathbf{a}$ during learning. With fast learning, $R_J^{\text{new}} = R_J^{\text{old}} \oplus \mathbf{a}$.



Adaptive Resonance Theory. Figure 7. Fuzzy ART geometry. The weight of a category node *J* is represented in complement-coding form as $\mathbf{w}_J = (\mathbf{u}_J \mid \mathbf{v}_J^c)$, and the *M*-dimensional vectors $\mathbf{u}_J$ and $\mathbf{v}_J$ define the corners of the category box R_J. When $M = 2$, the size of R_J equals its width plus its height. During learning, R_J expands toward $R_J \oplus \mathbf{a}$, defined as the smallest box enclosing both R_J and a. Node *J* will reset before learning if $|R_J \oplus \mathbf{a}| > M(1 - \rho)$



Adaptive Resonance Theory. Figure 8. ART 3 search implements a medium-term memory within the F_0-to-F_2 pathways, which biases the system against choosing a category node that had just produced a reset

Biasing Against Previously Active Category Nodes and Previously Attended Features During Attentive Memory Search Activity $\mathbf{x}$ at the ART field F_1 continuously computes the match between the field's bottom-up and top-down input patterns. A reset signal r shuts off the active F_2 node J when $\mathbf{x}$ fails to meet the matching criterion determined by the value of the vigilance parameter ρ. Reset alone does not, however, trigger a search for a different F_2 node: unless the prior activation has left an enduring trace within the F_0-to-F_2 subsystem, the network will simply reactivate the same node as before. As modeled in ART 3, biasing the bottom-up input to the coding field F_2 to favor the previously inactive nodes implements search by allowing the network to activate a new node in response to a reset signal. The ART 3 search mechanism defines a medium-term memory (MTM) in the F_0-to-F_2 adaptive filter which biases the system against re-choosing a node that had just produced a reset. A presynaptic interpretation of this bias is transmitter depletion, or habituation (Fig. 8).

Medium-term memory in all ART models allows the network to shift attention among learned categories at the coding field F_2 during search. The new *biased ART* network (Carpenter & Gaddam, 2010) introduces a second medium-term memory that shifts attention among input features, as well as categories, during search.

Self-Organizing Rule Discovery This foundation of computational principles and mechanisms has enabled the development of an ART information fusion system that is capable of incrementally learning a cognitive hierarchy of rules in response to probabilistic, incomplete, and even contradictory data that are collected by multiple observers (Carpenter, Martens, & Ogas, 2005).

Cross References

►Bayes Rule
►Bayesian Methods

Recommended Reading

Bullier, J., Hupé, J. M., James, A., & Girard, P. (1996). Functional interactions between areas V1 and V2 in the monkey. *Journal of Physiology Paris*, *90*(3–4), 217–220.

Carpenter, G. A. (1997). Distributed learning, recognition, and prediction by ART and ARTMAP neural networks. *Neural Networks*, *10*, 1473–1494.

Carpenter, G. A. & Gaddam, S. C. (2010). Biased ART: A neural architecture that shifts attention towards previously disregarded features following an incorrect prediction. *Neural Networks*, 23.

Carpenter, G. A., & Grossberg, S. (1987). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37, 54–115.

Carpenter, G. A. & Grossberg, S. (1993). Normal and amnesic learning, recognition, and memory by a neural model of cortico-hippocampal interactions. *Trends in Neurosciences*, *16*, 131–137.

Carpenter, G. A., Grossberg, S., Markuzon, N., Reynolds, J. H. & Rosen, D. B. (1992). Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, *3*, 698–713.

Carpenter, G. A., Martens, S., & Ogas, O. J. (2005). Self-organizing information fusion and hierarchical knowledge discovery: A

new framework using ARTMAP neural networks. *Neural Networks*, *18*, 287–295.
Carpenter, G. A., Milenova, B. L., & Noeske, B. W. (1998). Distributed ARTMAP: A neural network for fast distributed supervised learning. *Neural Networks*, *11*, 793–813.
Caudell, T. P., Smith, S. D. G., Escobedo, R., & Anderson, M. (1994). NIRS: Large scale ART 1 neural architectures for engineering design retrieval. *Neural Networks*, *7*, 1339–1350.
Grossberg, S. (1976). Adaptive pattern classification and universal recoding, II: Feedback, expectation, olfaction, and illusions. *Biological Cybernetics*, *23*, 187–202.
Grossberg, S. (1980). How does a brain build a cognitive code? *Psychological Review*, *87*, 1–51.
Grossberg, S. (1999). The link between brain, learning, attention, and consciousness. *Consciousness and Cognition*, *8*, 1–44.
Grossberg, S. (2000). The complementary brain: Unifying brain dynamics and modularity. *Trends in Cognitive Sciences*, *4*, 233–246.
Grossberg, S. (2003). How does the cerebral cortex work? Development, learning, attention, and 3D vision by laminar circuits of visual cortex. *Behavioral and Cognitive Neuroscience Reviews, 2*, 47–76.
Grossberg, S. (2007). Consciousness CLEARS the mind. *Neural Networks*, *20*, 1040–1053.
Grossberg, S. & Versace, M. (2008). Spikes, synchrony, and attentive learning by laminar thalamocortical circuits. *Brain Research*, *1218*, 278–312.
Parsons, O., & Carpenter, G. A. (2003). ARTMAP neural networks for information fusion and data mining: Map production and target recognition methodologies. *Neural Networks*, *16*(7), 1075–1089.
Posner, M. I., & Keele, S. W. (1968). On the genesis of abstract ideas. *Journal of Experimental Psychology*, *77*, 353–363.
Raizada, R., & Grossberg, S. (2003). Towards a theory of the laminar architecture of cerebral cortex: Computational clues from the visual system. *Cerebral Cortex, 13*, 100–113.
Sillito, A. M., Jones, H. E., Gerstein, G. L., & West, D. C. (1994). Feature-linked synchronization of thalamic relay cell firing induced by feedback from the visual cortex. *Nature*, *369*, 479–482.

Adaptive System

►Complexity in Adaptive Systems

Agent

In computer science, the term "agent" usually denotes a software abstraction of a real entity which is capable of acting with a certain degree of autonomy. For example, in artificial societies, agents are software abstractions of real people, interacting in an artifical, simulated environment. Various authors have proposed different definitions of agents. Most of them would agree on the following set of agent properties:

- Persistence: Code is not executed on demand but runs continuously and decides autonomously when it should perform some activity.
- Social ability: Agents are able to interact with other agents.
- Reactivity: Agents perceive the environment and are able to react.
- Proactivity: Agents exhibit goal-directed behavior and can take the initiative.

Agent-Based Computational Models

►Artificial Societies

Agent-Based Modeling and Simulation

►Artificial Societies

Agent-Based Simulation Models

►Artificial Societies

AIS

►Artificial Immune Systems

Algorithm Evaluation

Geoffrey I. Webb
Monash University, Victoria, Australia

Definition

Algorithm evaluation is the process of assessing a property or properties of an algorithm.

Motivation and Background

It is often valuable to assess the efficacy of an algorithm. In many cases, such assessment is relative, that is,

evaluating which of several alternative algorithms is best suited to a specific application.

Processes and Techniques

Many learning algorithms have been proposed. In order to understand the relative merits of these alternatives, it is necessary to evaluate them. The primary approaches to evaluation can be characterized as either theoretical or experimental. Theoretical evaluation uses formal methods to infer properties of the algorithm, such as its computational complexity (Papadimitriou, 1994), and also employs the tools of ►computational learning theory to assess learning theoretic properties. Experimental evaluation applies the algorithm to learning tasks to study its performance in practice.

There are many different types of property that may be relevant to assess depending upon the intended application. These include algorithmic properties, such as time and space complexity. These algorithmic properties are often assessed separately with respect to performance when learning a ►model, that is, at ►training time, and performance when applying a learned model, that is, at ►test time.

Other types of property that are often studied are the properties of the models that are learned (see ►model evaluation). Strictly speaking, such properties should be assessed with respect to a specific application or class of applications. However, much machine learning research includes experimental studies in which algorithms are compared using a set of data sets with little or no consideration given to what class of applications those data sets might represent. It is dangerous to draw general conclusions about relative performance on any application from relative performance on this sample of some unknown class of applications. Such experimental evaluation has become known disparagingly as a *bake-off*.

An approach to experimental evaluation that may be less subject to the limitations of bake-offs is the use of experimental evaluation to assess a learning algorithm's ►bias and variance profile. Bias and variance measure properties of an algorithm's propensities in learning models rather than directly being properties of the models that are learned. Hence, they may provide more general insights into the relative characteristics of alternative algorithms than do assessments of the performance of learned models on a finite number of applications. One example of such use of bias–variance analysis is found in Webb (2000).

Techniques for experimental algorithm evaluation include ►bootstrap sampling, ►cross-validation, and ►holdout evaluation.

Cross References

►Computational Learning Theory
►Model Evaluation

Recommended Reading

Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The elements of statistical learning*. New York: Springer.
Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.
Papadimitriou, C. H. (1994). *Computational complexity*. Reading, MA: Addison-Wesley.
Webb, G. I. (2000). MultiBoosting: A technique for combining boosting and wagging. *Machine Learning*, *40*(2), 159–196.
Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.

Analogical Reasoning

►Instance-Based Learning

Analysis of Text

►Text Mining

Analytical Learning

►Deductive Learning
►Explanation-Based Learning

Ant Colony Optimization

Marco Dorigo, Mauro Birattari
Université Libre de Bruxelles, Brussels, Belgium

Synonyms

ACO

Definition

Ant colony optimization (ACO) is a population-based metaheuristic for the solution of difficult combinatorial

optimization problems. In ACO, each individual of the population is an artificial agent that builds incrementally and stochastically a solution to the considered problem. Agents build solutions by moving on a graph-based representation of the problem. At each step their moves define which solution components are added to the solution under construction. A probabilistic model is associated with the graph and is used to bias the agents' choices. The probabilistic model is updated online by the agents so as to increase the probability that future agents will build good solutions.

Motivation and Background

Ant colony optimization is so called because of its original inspiration: the foraging behavior of some ant species. In particular, in Beckers, Deneubourg, and Goss (1992) it was demonstrated experimentally that ants are able to find the shortest path between their nest and a food source by collectively exploiting the pheromone they deposit on the ground while walking. Similar to real ants, ACO's artificial agents, also called artificial ants, deposit artificial pheromone on the graph of the problem they are solving. The amount of pheromone each artificial ant deposits is proportional to the quality of the solution the artificial ant has built. These artificial pheromones are used to implement a probabilistic model that is exploited by the artificial ants to make decisions during their solution construction activity.

Structure of the Optimization System

Let us consider a minimization problem $(\mathcal{S}, f)$, where $\mathcal{S}$ is the *set of feasible solutions*, and f is the *objective function*, which assigns to each solution $s \in \mathcal{S}$ a cost value $f(s)$. The goal is to find an optimal solution s^*, that is, a feasible solution of minimum cost. The set of all optimal solutions is denoted by $\mathcal{S}^*$.

Ant colony optimization attempts to solve this minimization problem by repeating the following two steps:

- Candidate solutions are constructed using a parameterized probabilistic model, that is, a parameterized probability distribution over the solution space.
- The candidate solutions are used to modify the model in a way that is intended to bias future sampling toward low cost solutions.

The Ant Colony Optimization Probabilistic Model

We assume that the combinatorial optimization problem $(\mathcal{S}, f)$ is mapped on a problem that can be characterized by the following list of items:

- A finite set $\mathcal{C} = \{c_1, c_2, \ldots, c_{N_C}\}$ of *components*, where N_C is the number of components.
- A finite set $\mathcal{X}$ of *states* of the problem, where a state is a sequence $x = \langle c_i, c_j, \ldots, c_k, \ldots \rangle$ over the elements of $\mathcal{C}$. The length of a sequence x, that is, the number of components in the sequence, is expressed by $|x|$. The maximum length of a sequence is bounded by a positive constant $n < +\infty$.
- A set of (candidate) solutions $\mathcal{S}$, which is a subset of $\mathcal{X}$ (i.e., $\mathcal{S} \subseteq \mathcal{X}$).
- A set of feasible states $\tilde{\mathcal{X}}$, with $\tilde{\mathcal{X}} \subseteq \mathcal{X}$, defined via a set of *constraints* Ω.
- A nonempty set $\mathcal{S}^*$ of optimal solutions, with $\mathcal{S}^* \subseteq \tilde{\mathcal{X}}$ and $\mathcal{S}^* \subseteq \mathcal{S}$.

Given the above formulation (Note that, because this formulation is always possible, ACO can in principle be applied to any combinatorial optimization problem.) artificial ants build candidate solutions by performing randomized walks on the completely connected, weighted graph $\mathcal{G} = (\mathcal{C}, \mathcal{L}, \mathcal{T})$, where the vertices are the components $\mathcal{C}$, the set $\mathcal{L}$ fully connects the components $\mathcal{C}$, and $\mathcal{T}$ is a vector of so-called *pheromone trails* τ. Pheromone trails can be associated with components, connections, or both. Here we assume that the pheromone trails are associated with connections, so that $\tau(i,j)$ is the pheromone associated with the connection between components i and j. It is straightforward to extend the algorithm to the other cases. The graph $\mathcal{G}$ is called the *construction graph*.

To construct candidate solutions, each artificial ant is first put on a randomly chosen vertex of the graph. It then performs a randomized walk by moving at each step from vertex to vertex on the graph in such a way that the next vertex is chosen stochastically according to the strength of the pheromone currently on the arcs.

While moving from one node to another of the graph $\mathcal{G}$, constraints Ω may be used to prevent ants from building infeasible solutions. Formally, the solution construction behavior of a generic ant can be described as follows:

ANT_SOLUTION_CONSTRUCTION

- For each ant:
  - Select a start node c_1 according to some problem dependent criterion.
  - Set $k = 1$ and $x_k = \langle c_1 \rangle$.
- While $x_k = \langle c_1, c_2, \ldots, c_k \rangle \in \tilde{\mathcal{X}}, x_k \notin \mathcal{S}$, and the set J_{x_k} of components that can be appended to x_k is not empty, select the next node (component) c_{k+1} randomly according to:

$$P_{\mathcal{T}}(c_{k+1} = c | x_k) = \begin{cases} \dfrac{F_{(c_k,c)}(\tau(c_k,c))}{\sum_{(c_k,y)\in J_{x_k}} F_{(c_k,y)}(\tau(c_k,y))} & \text{if } (c_k,c) \in J_{x_k}, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where a connection (c_k, y) belongs to J_{x_k} if and only if the sequence $x_{k+1} = \langle c_1, c_2, \ldots, c_k, y \rangle$ satisfies the constraints Ω (that is, $x_{k+1} \in \tilde{\mathcal{X}}$) and $F_{(i,j)}(z)$ is some monotonic function – a common choice being $z^{\alpha}\eta(i,j)^{\beta}$, where $\alpha, \beta > 0$, and $\eta(i,j)$'s are heuristic values measuring the desirability of adding component j after i. If at some stage $x_k \notin \mathcal{S}$ and $J_{x_k} = \emptyset$, that is, the construction process has reached a dead-end, the current state x_k is discarded. However, this situation may be prevented by allowing artificial ants to build infeasible solutions as well. In such a case, an infeasibility penalty term is usually added to the cost function. Nevertheless, in most of the settings in which ACO has been applied, the dead-end situation does not occur.

For certain problems, one may find it useful to use a more general scheme, where F depends on the pheromone values of several "related" connections rather than just a single one. Moreover, instead of the *random-proportional rule* above, different selection schemes, such as the *pseudo-random-proportional rule* (Dorigo & Gambardella, 1997), may be used.

The Ant Colony Optimization Pheromone Update

Many different schemes for pheromone update have been proposed within the ACO framework. For an extensive overview, see Dorigo and Stützle (2004). Most pheromone updates can be described using the following generic scheme:

GENERIC_ACO_UPDATE

- $\forall s \in \hat{S}_t, \forall (i,j) \in s: \ \tau(i,j) \leftarrow \tau(i,j) + Q_f(s|S_1, \ldots, S_t)$,
- $\forall (i,j): \ \tau(i,j) \leftarrow (1-\rho) \cdot \tau(i,j)$,

where S_i is the sample in the ith iteration, ρ, $0 \le \rho < 1$, is the evaporation rate, and $Q_f(s|S_1, \ldots, S_t)$ is some "quality function," which is typically required to be non-increasing with respect to f and is defined over the "reference set" $\hat{S}_t$.

Different ACO algorithms may use different quality functions and reference sets. For example, in the very first ACO algorithm – Ant System (Dorigo, Maniezzo, & Colorni, 1991, 1996) – the quality function is simply $1/f(s)$ and the reference set $\hat{S}_t = S_t$. In a subsequently proposed scheme, called *iteration best update* (Dorigo & Gambardella, 1997), the reference set is a singleton containing the best solution within S_t (if there are several iteration-best solutions, one of them is chosen randomly). For the *global-best update* (Dorigo et al., 1996; Stützle & Hoos, 1997), the reference set contains the best among all the iteration-best solutions (and if there are more than one global-best solution, the earliest one is chosen). In Dorigo et al. (1996) an *elitist* strategy was introduced, in which the update is a combination of the previous two.

In case a good lower bound on the optimal solution cost is available, one may use the following quality function (Maniezzo, 1999):

$$Q_f(s|S_1, \ldots, S_t) = \tau_0 \left(1 - \frac{f(s) - \mathrm{LB}}{\bar{f} - \mathrm{LB}}\right) = \tau_0 \frac{\bar{f} - f(s)}{\bar{f} - \mathrm{LB}}, \tag{2}$$

where $\bar{f}$ is the average of the costs of the last k solutions and LB is the lower bound on the optimal solution cost. With this quality function, the solutions are evaluated by comparing their cost to the average cost of the other recent solutions, rather than by using the absolute cost values. In addition, the quality function is automatically scaled based on the proximity of the average cost to the lower bound.

A pheromone update that slightly differs from the generic update described above was used in *ant colony system* (ACS) (Dorigo & Gambardella, 1997). There the pheromone is evaporated by the ants online during the solution construction, hence only the pheromone involved in the construction evaporates.

Another modification of the generic update was introduced in $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System (Stützle & Hoos, 1997, 2000), which uses maximum and minimum pheromone trail limits. With this modification, the probability of generating any particular solution is kept above some positive threshold. This helps to prevent search stagnation and premature convergence to suboptimal solutions.

Cross References

►Swarm Intelligence

Recommended Reading

Beckers, R., Deneubourg, J. L., & Goss, S. (1992). Trails and U-turns in the selection of the shortest path by the ant Lasius Niger. *Journal of Theoretical Biology, 159*, 397–415.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy.

Dorigo M., Maniezzo V., & Colorni A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B, 26*(1), 29–41.

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. Cambridge, MA: MIT Press.

Maniezzo, V. (1999). Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing, 11*(4), 358–369.

Stützle, T., & Hoos, H. H. (1997). The $\mathcal{MAX}$–$\mathcal{MIN}$ ant system and local search for the traveling salesman problem. In *Proceedings of the 1997 Congress on Evolutionary Computation – CEC'97* (pp. 309–314). Piscataway, NJ: IEEE Press.

Stützle, T., & Hoos, H. H. (2000). $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. *Future Generation Computer Systems, 16*(8), 889–914, 2000.

Anytime Algorithm

An *anytime algorithm* is an algorithm whose output increases in quality gradually with increased running time. This is in contrast to algorithms that produce no output at all until they produce full-quality output after a sufficiently long execution time. An example of an algorithm with good anytime performance is ►Adaptive Real-Time Dynamic Programming (ARTDP).

AODE

►Averaged One-Dependence Estimators

Apprenticeship Learning

►Behavioral Cloning

Approximate Dynamic Programming

►Value Function Approximation

Apriori Algorithm

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Definition

Apriori algorithm (Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996) is a ►data mining method which outputs all ►frequent itemsets and ►association rules from given data.

Input: set $\mathcal{I}$ of items, multiset $\mathcal{D}$ of subsets of $\mathcal{I}$, frequency threshold *min_fr*, and confidence threshold *min_conf*.

Output: all frequent itemsets and all valid association rules in $\mathcal{D}$.

Method:

1: level := 1; frequent_sets := $\emptyset$;
2: candidate_sets := $\{\{i\} \mid i \in \mathcal{I}\}$;
3: while candidate_sets $\neq \emptyset$
3.1: scan data $\mathcal{D}$ to compute frequencies of all sets in candidate_sets;
3.2: frequent_sets := frequent_sets $\cup$ $\{C \in$ candidate_sets $\mid$ frequency$(C) \geq$ *min_fr*$\}$;
3.3 level := level + 1;
3.4: candidate_sets := $\{A \subset \mathcal{I} \mid |A| =$ level and $B \in$ frequent_sets for all $B \subset A, |B| =$ level $- 1\}$;

4: output frequent_sets;
5: for each $F \in$ frequent_sets
5.1: for each $E \subset F, E \neq \emptyset, E \neq F$
5.1.1: if frequency(F)/frequency$(E) \geq$ *min_conf* then output association rule $E \rightarrow (F \setminus E)$

The algorithm finds frequent itemsets (lines 1-4) by a breadth-first, general-to-specific search. It generates and tests candidate itemsets in batches, to reduce the overhead of database access. The search starts with the most general itemset patterns, the singletons, as candidate patterns (line 2). The algorithm then iteratively computes the frequencies of candidates (line 3.1) and saves those that are frequent (line 3.2). The crux of the algorithm is in the candidate generation (line 3.4): on the next level, those itemsets are pruned that have an infrequent subset. Obviously, such itemsets cannot be frequent. This allows Apriori to find all frequent itemset without spending too much time on infrequent itemsets. See ►frequent pattern and ►constraint-based mining for more details and extensions.

Finally, the algorithm tests all frequent association rules and outputs those that are also confident (lines 5-5.1.1).

Cross References

►Association Rule
►Basket Analysis
►Constraint-Based Mining
►Frequent Itemset
►Frequent Pattern

Recommended Reading

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307–s328). Menlo Park: AAAI Press.

Area Under Curve

Synonyms

AUC

Definition

The *area under curve* (AUC) statistic is an empirical measure of classification performance based on the area under an ROC curve. It evaluates the performance of a scoring classifier on a test set, but ignores the magnitude of the scores and only takes their rank order into account. AUC is expressed on a scale of 0 to 1, where 0 means that all negatives are ranked before all positives, and 1 means that all positives are ranked before all negatives. See ►ROC Analysis.

AQ

►Rule Learning

ARL

►Average-Reward Reinforcement Learning

ART

►Adaptive Resonance Theory

ARTDP

►Adaptive Real-Time Dynamic Programming

Artificial Immune Systems

Jon Timmis
University of York, Heslington, North Yorkshire, UK

Synonyms

AIS; Immune computing; Immune-inspired computing; Immunocomputing; Immunological computation

Definition

Artificial immune systems (AIS) have emerged as a computational intelligence approach that shows great promise. Inspired by the complexity of the immune system, computer scientists and engineers have created systems that in some way mimic or capture certain computationally appealing properties of the immune system, with the aim of building more robust and adaptable solutions. AIS have been defined by de Castro and Timmis (2002) as:

- adaptive systems, inspired by theoretical immunology and observed immune functions, principle and models, which are applied to problem solving

AIS are not limited to machine learning systems, there are a wide variety of other areas in which AIS are developed such as optimization, scheduling, fault tolerance, and robotics (Hart & Timmis, 2008). Within the context of machine learning, both supervised and unsupervised approaches have been developed. Immune-inspired learning approaches typically develop a memory set of detectors that are capable of classifying unseen data items (in the case of supervised learning) or a memory set of detectors that represent clusters within the data (in the case of unsupervised learning). Both static and dynamic learning systems have been developed.

Motivation and Background

The immune system is a complex system that undertakes a myriad of tasks. The abilities of the immune system have helped to inspire computer scientists to build systems that *mimic*, in some way, various properties of the immune system. This field of research, AIS, has seen the application of immune-inspired algorithms to a wide variety of areas.

The origin of AIS has its roots in the early theoretical immunology work of Farmer, Perelson, and Varela (Farmer, Packard, & Perelson, 1986; Varela, Coutinho, Dupire, & Vaz, 1988). These works investigated a number of theoretical ▶immune network models proposed to describe the maintenance of immune memory in the absence of antigen. While controversial from an immunological perspective, these models began to give rise to an interest from the computing community. The most influential people at crossing the divide between computing and immunology in the early days were Bersini and Forrest. It is fair to say that some of the early work by Bersini (1991) was very well rooted in immunology, and this is also true of the early work by Forrest (1994). It was these works that formed the basis of a solid foundation for the area of AIS. In the case of Bersini, he concentrated on the immune network theory, examining how the immune system maintained its memory and how one might build models and algorithms mimicking that property. With regard to Forrest, her work was focused on computer security (in particular, network intrusion detection) and formed the basis of a great deal of further research by the community on the application of immune-inspired techniques to computer security.

At about the same time as Forrest was undertaking her work, other researchers began to investigate the nature of learning in the immune system and how that might by used to create *machine learning* algorithms (Cook & Hunt, 1995). They had the idea that it might be possible to exploit the mechanisms of the immune system (in particular, the immune network) in learning systems, so they set about doing a proof of concept (Cook & Hunt, 1995). Initial results were very encouraging, and they built on their success by applying the immune ideas to the classification of DNA sequences as either promoter or nonpromoter classes: this work was generalized in Timmis and Neal (2001).

Similar work was carried out by de Castro and Von Zuben (2001), who developed algorithms for use in function optimization and data clustering. Work in dynamic unsupervised machine learning algorithms was also undertaken, meeting with success in works such as Neal (2002). In the supervised learning domain, very little happened until the work by Watkins (2001) (later expanded in Watkins, 2005) developed an immune-based classifier known as AIRS, and in the dynamic supervised domain, with the work in Secker, Freitas, and Timmis (2003) being one of a number of successes.

Structure of the Learning System

In an attempt to create a common basis for AIS, the work in de Castro and Timmis (2002) proposed the idea of a framework for engineering AIS. They argued that the case for such a framework as the existence of similar frameworks in other biologically inspired approaches, such as ▶artificial neural networks (ANNs) and evolutionary algorithms (EAs), has helped considerably with the understanding and construction of such systems. For example, de Castro and Timmis (2002) consider a set of artificial neurons, which can be arranged together to form an ANN. In order to acquire knowledge, these neural networks undergo an adaptive process, known as learning or training, which alters (some of) the parameters within the network. Therefore, they argued that in a simplified form, a framework to design an ANN is

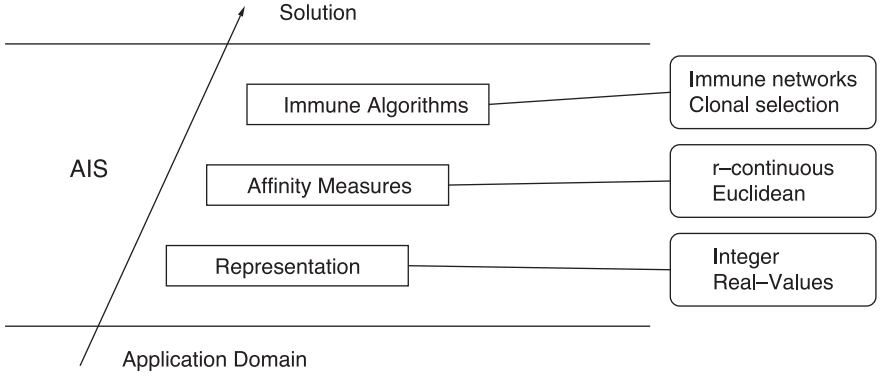
composed of a set of artificial neurons, a pattern of interconnection for these neurons, and a learning algorithm. Similarly, they argued that in evolutionary algorithms, there is a set of artificial chromosomes representing a population of individuals that iteratively suffer a process of reproduction, genetic variation, and selection. As a result of this process, a population of evolved artificial individuals arises. A framework, in this case, would correspond to the genetic representation of the individuals of the population, plus the procedures for reproduction, genetic variation, and selection. Therefore, they proposed that a framework to design a biologically inspired algorithm requires, at least, the following basic elements:

- A representation for the components of the system
- A set of mechanisms to evaluate the interaction of individuals with the environment and each other. The environment is usually stimulated by a set of input stimuli, one or more fitness function(s), or other means
- Procedures of adaptation that govern the dynamics of the system, i.e., how its behavior varies over time

This framework can be thought of as a layered approach such as the specific framework for engineering AIS of de Castro and Timmis (2002) shown in Fig. 1. This framework follows the three basic elements for designing a biologically inspired algorithm just described, where the set of mechanisms for evaluation are the affinity measures and the procedures of adaptation are the immune algorithms. In order to build a system such as an AIS, one typically requires an application domain or target function. From this basis, the way in which the components of the system will be represented is considered. For example, the representation of network traffic may well be different from the representation of a real-time embedded system. In AIS, the way in which something is represented is known as *shape space*. There are many kinds of shape space, such as Hamming, real valued, and so on, each of which carries it own bias and should be selected with care (Freitas & Timmis, 2003). Once the representation has been chosen, one or more affinity measures are used to quantify the interactions of the elements of the system. There are many possible affinity measures (which are partially dependent upon the representation adopted), such as Hamming and Euclidean distance metrics. Again, each of these has its own bias, and the affinity function must be selected with great care, as it can affect the overall performance (and ultimately the result) of the system (Freitas & Timmis, 2003).

Supervised Immune-Inspired Learning

The artificial immune recognition system (AIRS) algorithm was introduced as one of the first immune-inspired supervised learning algorithms and has subsequently gone through a period of study and refinement (Watkins, 2005). To use classifications from de Castro and Timmis (2002), for the procedures of adaptation, AIRS is a, ▶clonal selection type of immune-inspired algorithm. The representation and affinity layers of the system are standard in



Artificial Immune Systems. Figure 1. AIS layered framework adapted from de Castro and Timmis (2002)

that any number of representations such as binary, real values, etc., can be used with the appropriate affinity function. AIRS has its origin in two other immune-inspired algorithms: CLONALG (CLONAL Selection alGorithm) and Artificial Immune NEtwork (AINE) (de Castro and Timmis, 2002). AIRS resembles CLONALG in the sense that both the algorithms are concerned with developing a set of memory cells that give a representation of the learned environment.

AIRS is concerned with the development of a set of memory cells that can encapsulate the training data. This is done in a two-stage process of first evolving a candidate memory cell and then determining if this candidate cell should be added to the overall pool of memory cells. The learning process can be outlined as follows:

1. For each pattern to be recognized, do
   (a) Compare a training instance with all memory cells of the same class and find the memory cell with the best affinity for the training instance. This is referred to as a memory cell mc_{match}.
   (b) Clone and mutate mc_{match} in proportion to its affinity to create a pool of abstract B-cells.
   (c) Calculate the affinity of each B-cell with the training instance.
   (d) Allocate resources to each B-cell based on its affinity.
   (e) Remove the weakest B-cells until the number of resources returns to a preset limit.
   (f) If the average affinity of the surviving B-cells is above a certain level, continue to step 1(g). Else, clone and mutate these surviving B-cells based on their affinity and return to step 1(c).
   (g) Choose the best B-cell as a candidate memory cell (mc_{cand}).
   (h) If the affinity of mc_{cand} for the training instance is better than the affinity of mc_{match}, then add mc_{cand} to the memory cell pool. If, in addition to this, the affinity between mc_{cand} and mc_{match} is within a certain threshold, then remove mc_{match} from the memory cell pool.
2. Repeat from step 1(a) until all training instances have been presented.

Once this training routine is complete, AIRS classifies the instances using k-nearest neighbor with the developed set of memory cells.

Unsupervised Immune-Inspired Learning

The artificial immune network (aiNET) algorithm was introduced as one of the first immune-inspired unsupervised learning algorithms and has subsequently gone through a period of study and refinement (de Castro & Von Zuben, 2001). To use classifications from de Castro and Timmis (2002), for the procedures of adaptation, aiNET is an immune network type of immune-inspired algorithm. The representation and affinity layers of the system are standard (the same as in AIRS). aiNET has its origin in another immune-inspired algorithms: CLONALG (the same forerunner to AIRS), and resembles CLONALG in the sense that both algorithms (again) are concerned with developing a set of memory cells that give a representation of the learnt environment. However, within aiNET there is no error feedback into the learning process. The learning process can be outlined as follows:

1. Randomly initialize a population P
2. For each pattern to be recognized, do
   (a) Calculate the affinity of each B-cell (b) in the network for an instance of the pattern being learnt
   (b) Select a number of elements from P into a clonal pool C
   (c) Mutate each element of C proportional to affinity to the pattern being learnt (the higher the affinity, the less mutation applied)
   (d) Select the highest affinity members of C to remain in the set C and remove the remaining elements
   (e) Calculate the affinity between all members of C and remove elements in C that have an affinity below a certain threshold (user defined)
   (f) Combine the elements of C with the set P
   (g) Introduce a random number of randomly created elements into P to maintain diversity
3. Repeat from 2(a) until stopping criteria is met

Once this training routine is complete, the minimum-spanning tree algorithm is applied to the network to extract the clusters from within the network.

Recommended Reading

Bersini, H. (1991). Immune network and adaptive control. In *Proceedings of the 1st European conference on artificial life (ECAL)* (pp. 217–226). Cambridge, MA: MIT Press.

Cooke, D., & Hunt, J. (1995). Recognising promoter sequences using an artificial immune system. In *Proceedings of intelligent systems in molecular biology* (pp. 89–97). California: AAAI Press.

de Castro, L. N., & Timmis, J. (2002). *Artificial immune systems: A new computational intelligence approach*. New York: Springer.

de Castro, L. N., & Von Zuben, F. J. (2001). *aiNet: An artificial immune network for data analysis* (pp. 231–259). Hershey, PA: Idea Group Publishing.

Farmer, J. D., Packard, N. H., & Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Physica D, 22*, 187–204.

Forrest, S., Perelson, A. S., Allen, L., Cherukuri, R. (1994). Self–nonself discrimination in a computer. In *Proceedings of the IEEE symposium on research security and privacy* (pp. 202–212).

Freitas, A., & Timmis, J. (2003). *Revisiting the foundations of artificial immune systems: A problem oriented perspective*, LNCS (Vol. 2787) (pp. 229–241). New York: Springer.

Hart, E., & Timmis, J. (2008). Application Areas of AIS: The Past, Present and the Future. *Journal of Applied Soft Computing, 8*(1). pp. 191–201.

Neal, M. (2002). An artificial immune system for continuous analysis of time-varying data. In J. Timmis & P. Bentley (Eds.), *Proceedings of the 1st international conference on artificial immune system (ICARIS)* (pp. 76–85). Canterbury, UK: University of Kent Printing Unit.

Secker, A., Freitas, A., & Timmis, J. (2003). AISEC: An artificial immune system for email classification. In *Proceedings of congress on evolutionary computation (CEC)* (pp. 131–139).

Timmis, J., & Bentley (Eds.). (2002). *Proceedings of the 1st international conference on artificial immune system (ICARIS)*. Canterbury, UK: University of Kent Printing Unit.

Timmis, J., & Neal, M. (2001). A resource limited artificial immune system for data analysis. *Knowledge Based Systems, 14*(3–4), 121–130.

Varela, F., Coutinho, A., Dupire, B., & Vaz, N. (1988). Cognitive networks: Immune, neural and otherwise. *Journal of Theoretical Immunology, 2*, 359–375.

Watkins, A. (2001). AIRS: A resource limited artificial immune classifier. Master's thesis, Mississippi State University.

Watkins, A. (2005). Exploiting immunological metaphors in the development of serial, parallel and distributed learning algorithms. PhD thesis, University of Kent.

Artificial Life

Artificial Life is an interdisciplinary research area trying to reveal and understand the principles and organization of living systems. Its main goal is to artificially synthesize life-like behavior from scratch in computers or other artificial media. Important topics in artificial life include the origin of life, growth and development, evolutionary and ecological dynamics, adaptive autonomous robots, emergence and self-organization, social organization, and cultural evolution.

Artificial Neural Networks

(ANNs) is a computational model based on biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase.

Cross References

►Adaptive Resonance Theory
►Backpropagation
►Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity
►Boltzmann Machines
►Cascade Correlation
►Competitive Learning
►Deep Belief Networks
►Evolving Neural Networks
►Hypothesis Language
►Neural Network Topology
►Neuroevolution
►Radial Basis Function Networks
►Reservoir Computing
►Self-Organizing Maps
►Simple Recurrent Networks
►Weights

Artificial Societies

Jürgen Branke
University of Warwick, Coventry, UK

Synonyms

Agent-based computational models; Agent-based modeling and simulation; Agent-based simulation models

Definition

An artificial society is an agent-based, computer-implemented simulation model of a society or group of people, usually restricted to their interaction in a particular situation. Artificial societies are used in economics and social sciences to explain, understand, and analyze socioeconomic phenomena. They provide scientists with a fully controllable virtual laboratory to test hypotheses and observe complex system behavior emerging as result of the ►agents' interaction. They allow formalizing and testing social theories by using computer code, and make it possible to use experimental methods with social phenomena, or at least with their computer representations, on a large scale. Because the designer is free to choose any desired ►agent behavior as long as it can be implemented, research based on artificial societies is not restricted by assumptions typical in classical economics, such as homogeneity and full rationality of agents. Overall, artificial societies have added an all new dimension to research in economics and social sciences and have resulted in a new research field called "agent-based computational economics."

Artificial societies should be distinguished from virtual worlds and ►artificial life. The term virtual world is usually used for virtual environments to interact with, as, e.g., in computer games. In artificial life, the goal is more to learn about biological principles, understand how life could emerge, and create life within a computer.

Motivation and Background

Classical economics can be roughly divided into analytical and empirical approaches. The former uses deduction to derive theorems from assumptions. Thereby, analytical models usually include a number of simplifying assumptions in order to keep the model tractable, the most typical being full rationality and homogeneity of agents. Also, analytical economics is often limited to equilibrium calculations. Classical empirical economics collects data from the real world, and derives patterns and regularities inductively. In recent years, the tremendous increase in available computational power gave rise to a new branch of economics and sociology which uses simulation of artificial societies as a tool to generate new insights.

Artificial societies are agent-based, computer-implemented simulation models of real societies or a group of people in a specific situation. They are built from the bottom up, by specifying the behavior of the agents in different situations. The simulation then reveals the emerging global behavior of the system, and thus provides a link between micro-level behavior of the agents and macro-level characteristics of the system. Using simulation, researchers can now carry out social experiments under fully controlled and reproducible laboratory conditions, trying out different configurations and observing the consequences.

Like deduction, simulation models are based on a set of clearly specified assumptions as written down in a computer program. This is then used to generate data, from which regularities and patterns are derived inductively. As such, research based on artificial societies stands somewhere between the classical analytical and empirical social sciences.

One of the main advantages of artificial societies is that they allow to consider very complex scenarios where agents are heterogeneous, boundedly rational, or have the ability to learn. Also, they allow to observe evolution over time, instead of just the equilibrium.

Artificial societies can be used for many purposes, e.g.:

1. Verification: Test a hypothesis or theory by examining its validity in relevant, clearly defined scenarios.
2. Explanation: Construct an artificial society which shows the same behavior as the real society. Then analyze the model to explain the emergent behavior.
3. Prediction: Run a model of an existing society into the future. Also, feed the model with different input parameters and use the result as a prediction on how the society would react.
4. Optimization: Test different strategies in the simulation environment, trying to find a best possible strategy.
5. Existence proof: Demonstrate that a specific simulation model is able to generate a certain global behavior.
6. Discovery: Play around with parameter settings, discovering new interdependencies and gaining new insights.
7. Training and education: Use simulation as demonstrator.

Structure of the Learning System

Using artificial societies requires the usual steps in model building and experimental science, including

1. Developing a conceptual model
2. Building the simulation model
3. Verification (making sure the model is correct)
4. Validation (making sure the model is suitable to answer the posed questions)
5. Simulation and analysis using an appropriate experimental design.

Artificial society is an interdisciplinary research area involving, among others, computer science, psychology, economics, sociology, and biology.

Important Aspects

The modeling, simulation, and analysis process described in the previous section is rather complex and only remotely connected to machine learning. Thus, instead of a detailed description of all steps, the following focuses on aspects particularly interesting from a machine learning point of view.

Modeling Learning

One of the main advantages of artificial societies is that they can account for boundedly rational and learning agents. For that, one has to specify (in form of a program) exactly how agents decide and learn.

In principle, all the learning algorithms developed in machine learning could be used, and many have been used successfully, including ►reinforcement learning, ►artificial neural networks, and ►evolutionary algorithms. However, note that the choice of a learning algorithm is not determined by its learning speed and efficiency (as usual in machine learning), but by how well it reflects human learning in the considered scenario, at least if the goal is to construct an artificial society which allows conclusions to be transferred to the real world. As a consequence, many learning models used in artificial societies are motivated by psychology. The idea of the most suitable model depends on the simulation context, e.g., on whether the simulated learning process is conscious or nonconscious, or on the time and effort an individual may be expected to spend on a particular decision.

Besides individual learning (i.e., learning from own past experience), artificial societies usually feature social learning (where one agent learns by observing others), and cultural learning (e.g., the evolution of norms). While the latter simply emerges from the interaction of the agents, the former has to be modeled explicitly. Several different models for learning in artificial societies are discussed in Brenner (2006).

One popular learning paradigm which can be used as a model for individual as well as social learning are ►evolutionary algorithms (EAs). Several studies suggest that EAs are indeed an appropriate model for learning in artificial societies, either based on comparisons of simulations with human subject experiments or based on comparisons with other learning mechanisms such as reinforcement learning (Duffy, 2006). As EAs are successful search strategies, they seem particularly suitable if the space of possible actions or strategies is very large.

If used to model individual learning, each agent uses a separate EA to search for a better personal solution. In this case, the EA population represents the different alternative actions or strategies that an agent considers. The genetic operators crossover and mutation are clearly related to two major ingredients of human innovation: combination and variation. Crossover can be seen as deriving a new concept by combining two known concepts, and mutation corresponds to a small variation of an existing concept. So, the agent, in some sense, creatively tries out new possibilities. Selection, which favors the best solutions found so far, models the learning part. A solution's quality is usually assessed by evaluating it in a simulation assuming all other agents keep their behavior.

For modeling social learning, EAs can be used in two different ways. In both cases, the population represents the actions or strategies of the different agents in the population. From this it follows that the population size corresponds to the number of agents in the simulation. Fitness values are calculated by running the simulation and observing how the different agents perform. Crossover is now seen as a model for information exchange, or imitation, among agents. Mutation, as in the individual learning case, is seen as a small variation of an existing concept.

The first social learning model simply uses a standard EA, i.e., selection chooses agents to "reproduce,"

and the resulting new agent strategy replaces an old strategy in the population. While allowing to use standard EA libraries, this approach does not provide a direct link between agents in the simulation and individuals in the EA population. In the second social learning model, each agent directly corresponds to an individual in the EA. In every iteration, each agent creates and tests a new strategy as follows. First, it selects a "donor" individual, with preference to successful individuals. Then it performs a crossover of its own strategy and the donor's strategy, and mutates the result. This can be regarded as an agent observing other agents, and partially adopting the strategies of successful other agents. Then, the resulting new strategy is tested in a "thought experiment," by testing whether the agent would be better off with the new strategy compared with its current strategy, assuming all other agents keep their behavior. If the new strategy performs better, it replaces the current strategy in the next iteration. Otherwise, the new strategy is discarded and the agent again uses its old strategy in the next iteration. The testing of new strategies against their parents has been termed election operator in Arifovic (1994), and makes sure that some very bad and obviously implausible agent strategies never enter the artificial society.

Examples

One of the first forerunners of artificial societies was Schelling's segregation model, 1969. In this study, Schelling placed some artificial agents of two different colors on a simple grid. Each agent follows a simple rule: if less than a given percentage of agents in the neighborhood had the same color, the agent moves to a random free spot. Otherwise, it stays. As the simulation shows, in this model, segregation of agent colors could be observed even if every individual agent was satisfied to live in a neighborhood with only 50% of its neighbors being of the same color. Thus, with this simple model, Schelling demonstrated that segregation of races in suburbs can occur even if each individual would be happy to live in a diverse neighborhood. Note that the simulations were actually not implemented on a computer but carried out by moving coins on a grid by hand.

Other milestones in artificial societies are certainly the work by Epstein and Axtell on their "sugarscape" model (Epstein & Axtell, 1996), and the Santa Fe artificial stock market (Arthur, Holland, LeBaron, Palmer, & Taylor, 1997). In the former, agents populate a simple grid world, with sugar growing as the only resource. The agents need the sugar for survival, and can move around to collect it. Axtell and Epstein have shown that even with agents following some very simple rules, the emerging behavior of the overall system can be quite complex and similar in many aspects to observations in the real world, e.g., showing a similar wealth distribution or population trajectories.

The latter is a simple model of a stock market with only a single stock and a risk-free fixed-interest alternative. This model has subsequently been refined and studied by many researchers. One remarkable result of the first model was to demonstrate that technical trading can actually be a viable strategy, something widely accepted in practice, but which classical analytical economics struggled to explain.

One of the most sophisticated artificial societies is perhaps the model of the Anasazi tribe, who left their dwellings in the Long House Valley in northeastern Arizona for so far unknown reasons around 1300 BC (Axtell et al., 2002). By building an artificial society of this tribe and the natural surroundings (climate etc.), it was possible to replicate macro behavior which is known to have occurred and provide a possible explanation for the sudden move.

The NewTies project (Gilbert et al., 2006) has a different and quite ambitious focus: it constructs artificial societies with the hope of an emerging artificial language and culture, which then might be studied to help explain how language and culture formed in human societies.

Software Systems

Agent-based simulations can be facilitated by using specialized software libraries such as Ascape, Netlogo, Repast, StarLogo, Mason, and Swarm. A comparison of different libraries can be found in Railsback, Lytinen, and Jackson (2006).

Applications

Artificial societies have many practical applications, from rather simple simulation models to very complex economic decision problems, examples include

traffic simulation, market design, evaluation of vaccination programs, evacuation plans, or supermarket layout optimization. See, e.g., Bonabeau (2002) for a discussion of several such applications.

Future Directions, Challenges

The science on artificial societies is still at its infancy, but the field is burgeoning and has already produced some remarkable results. Major challenges lie in the model building, calibration, and validation of the artificial society simulation model. Despite several agent-based modeling toolkits available, there is a lot to be gained by making them more flexible, intuitive, and user-friendly, allowing to construct complex models simply by selecting and combining provided building blocks of agent behavior. ►Behavioral Cloning may be a suitable machine learning approach to generate representative agent models.

Cross References

►Artificial Life
►Behavioral Cloning
►Co-Evolutionary Learning
►Multi-Agent Learning

Recommended Reading

Agent-based computational economics, website maintained by Tesfatsion (2009)

Axelrod: The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration (Axelrod, 1997)

Bonabeau: Agent-based modeling (Bonabeau, 2002)

Brenner: Agent learning representation: Advice on modeling economic learning (Brenner, 2006)

Epstein: Generative social science (Epstein, 2006)

Journal of Artificial Societies and Social Simulation (2009)

Tesfatsion and Judd (eds.): Handbook of computational economics (Tesfatsion & Judd, 2006)

Arifovic, J. (1994). Genetic algorithm learning and the cobweb-model. *Journal of Economic Dynamics and Control, 18*, 3–28.

Arthur, B., Holland, J., LeBaron, B., Palmer, R., & Taylor, P. (1997). Asset pricing under endogenous expectations in an artificial stock market. In B. Arthur et al., (Eds.), *The economy as an evolvin complex system II* (pp. 15–44). Boston: Addison-Wesley.

Axelrod, R. (1997). *The complexity of cooperation: Agent-based models of competition and collaboration*. Princeton, NJ: Princeton University Press.

Axtell, R. L., Epstein, J. M., Dean, J. S., Gumerman, G. J., Swedlund, A. C., Harburger, J., et al. (2002). Population growth and collapse in a multiagent model of the kayenta anasazi in long house valley. *Proceedings of the National Academy of Sciences, 99*, 7275–7279.

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, *99*, 7280–7287.

Brenner, T. (2006). Agent learning representation: Advice on modelling economic learning. In L. Tesfatsion & K. L. Judd, (Eds.), *Handbook of computational economics* (Vol. 2, pp.895–947). Amsterdam: North-Holland.

Duffy, J. (2006). Agent-based models and human subject experiments. In L. Tesfatsion & K. L. Judd, (Eds.), *Handbook of computational economics* (Vol. 2, pp.949–1011). Amsterdam: North-Holland.

Epstein, J. M. (2006). *Generative social science: Studies in agent-based computational modeling*. Princeton, NJ: Princeton University Press.

Epstein, J. M., & Axtell, R. (1996). *Growing artificial societies*. Washington, DC: Brookings Institution Press.

Gilbert, N., den Besten, M., Bontovics, A., Craenen, B. G. W., Divina, F., Eiben, A. E., et al. (2006). Emerging artificial societies through learning. *Journal of Artificial Societies and Social Simulation, 9*(2). http://jasss.soc.surrey.ac.uk/9/2/9.html.

Railsback, S. F., Lytinen, S. L., & Jackson, S. K. (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation, 82*(9), 609–623.

Schelling, T. C. (1969). Dynamic models of segregation. *Journal of Mathematical Sociology, 2*, 143–186.

Tesfatsion, L. (2009). Website on agent-based computational economics. http://www.econ.iastate.edu/tesfatsi/ace.htm.

Tesfatsion, L., & Judd, K. L. (Eds.) (2006). *Handbook of computational economics – Vol 2: Agent-based computational economics*. Amsterdam: Elsevier.

The journal of artificial societies and social simulation. http://jasss.soc.surrey.ac.uk/JASSS.html.

Assertion

In ►Minimum Message Length, the code or language shared between sender and receiver that is used to describe the model.

Association Rule

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Definition

Association rules (Agrawal, Imieliński, & Swami, 1993) can be extracted from data sets where each example

consists of a set of items. An association rule has the form $X \rightarrow Y$, where X and Y are ►itemsets, and the interpretation is that if set X occurs in an example, then set Y is also likely to occur in the example.

Each association rule is usually associated with two statistics measured from the given data set. The *frequency* or *support* of a rule $X \rightarrow Y$, denoted $\mathrm{fr}(X \rightarrow Y)$, is the number (or alternatively the relative frequency) of examples in which $X \cup Y$ occurs. Its *confidence*, in turn, is the observed conditional probability $P(Y \mid X) = \mathrm{fr}(X \cup Y)/\mathrm{fr}(X)$.

The ►Apriori algorithm (Agrawal, Mannila, Srikant, Toivonen & Verkamo, 1996) finds all association rules, between any sets X and Y, which exceed user-specified support and confidence thresholds. In association rule mining, unlike in most other learning tasks, the result thus is a set of rules concerning different subsets of the feature space.

Association rules were originally motivated by supermarket ►basket analysis, but as a domain independent technique they have found applications in numerous fields. Association rule mining is part of the larger field of ►frequent itemset or ►frequent pattern mining.

Cross References

►Apriori Algorithm
►Basket Analysis
►Frequent Itemset
►Frequent Pattern

Recommended Reading

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC* (pp. 207–216). New York: ACM.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307–328). Menlo Park: AAAI Press.

Associative Bandit Problem

►Associative Reinforcement Learning

Associative Reinforcement Learning

Alexander L. Strehl
Rütgers University, USA

Synonyms

Associative bandit problem; Bandit problem with side information; Bandit problem with side observations; One-step reinforcement learning

Definition

The *associative reinforcement-learning* problem is a specific instance of the ►*reinforcement learning* problem whose solution requires *generalization* and *exploration* but not *temporal credit assignment*. In associative reinforcement learning, an action (also called an arm) must be chosen from a fixed set of actions during successive timesteps and from this choice a real-valued reward or payoff results. On each timestep, an input vector is provided that along with the action determines, often probabilistically, the reward. The goal is to maximize the expected long-term reward over a finite or infinite horizon. It is typically assumed that the action choices do not affect the sequence of input vectors. However, even if this assumption is not asserted, learning algorithms are not required to infer or model the relationship between input vectors from one timestep to the next. Requiring a learning algorithm to discover and reason about this underlying process results in the full reinforcement learning problem.

Motivation and Background

The problem of associative reinforcement learning may be viewed as connecting the problems of ►supervised learning or ►classification, which is more specific, and reinforcement learning, which is more general. Its study is motivated by real-world applications such as choosing which internet advertisements to display based on information about the user or choosing which stock to buy based on current information related to the market. Both problems are distinguished from supervised learning by the absence of labeled training examples to learn from. For instance, in the advertisement problem, the learner is never told which ads would have resulted in the greatest expected reward (in this problem, reward is

determined by whether an ad is clicked on or not). In the stock problem, the best choice is never revealed since the choice itself affects the future price of the stocks and therefore the payoff.

The Learning Setting

The learning problem consists of the following core objects:

- An input space $\mathcal{X}$, which is a set of objects (often a subset of the n-dimension Euclidean space $\mathbb{R}^n$).
- A set of actions or arms $\mathcal{A}$, which is often a finite set of size k.
- A distribution D over $\mathcal{X}$. In some cases, D is allowed to be time-dependent and may be denoted D_t on timestep t for $t = 1, 2, \ldots$.

A learning sequence proceeds as follows. During each timestep $t = 1, 2, \ldots$, an input vector $x_t \in \mathcal{X}$ is is drawn according to the distribution D and is provided to the algorithm. The algorithm selects an aarm at $a_t \in \mathcal{A}$. This choice may be stochastic and depend on all previous inputs and rewards observed by the algorithm as well as all previous action choices made by the algorithm for timesteps $t = 1, 2, \ldots$. Then, the learner receives a payoff r_t generated according to some unknown stochastic process that depends only on the x_t and a_t. The informal goal is to maximize the expected long-term payoff. Let $\pi : \mathcal{X} \to \mathcal{A}$ be any policy that maps input vectors to actions. Let

$$V^{\pi}(T) := E\left[\sum_{i=1}^{T} r_i \,\middle|\, a_i = \pi(x_i) \text{ for } i = 1, 2, \ldots, T\right] \quad (1)$$

denotes the expected total reward over T steps obtained by choosing arms according to policy π. The expectation is taken over any randomness in the generation of input vectors x_i and rewards r_i. The expected regret of a learning algorithm with respect to policy π is defined as $V^{\pi}(T) - E[\sum_{i=1}^{T} r_i]$ the expected difference between the return from following policy π and the actual obtained return.

Power of Side Information

Wang, Kulkarni, and Poor (2005) studied the associative reinforcement learning problem from a statistical viewpoint. They considered the setting with two action and analyzed the *expected inferior sampling time*, which is the number of times that the lesser action, in terms of expected reward, is selected. The function mapping input vectors to conditional reward distributions belongs to a known parameterized class of functions, with the true parameters being unknown. They show that, under some mild conditions, an algorithm can achieve finite expected inferior sampling time. This demonstrates the power provided by the input vectors (also called *side observations* or *side information*), because such a result is not possible in the standard *multi-armed bandit problem*, which corresponds to the associative reinforcement-learning problem without input vectors x_i. Intuitively, this type of result is possible when the side information can be used to infer the payoff function of the optimal action.

Linear Payoff Functions

In its most general setting, the associative reinforcement learning problem is intractable. One way to rectify this problem is to assume that the payoff function is described by a linear system. For instance, Abe and Long (1999) and Auer (2002) consider a model where during each timestep t, there is a vector $z_{t,i}$ associated with each arm i. The expected payoff of pulling arm i on this timestep is given by $\theta^T z_{t,i}$ where θ is an unknown parameter vector and θ^T denotes the transpose of f. This framework maps to the framework described above by taking $x_t = (z_{t,1}, z_{t,2}, \ldots, z_{t,k})$. They assume a time-dependent distribution D and focus on obtaining bounds on the regret against the optimal policy. Assuming that all rewards lie in the interval $[0, 1]$, the worst possible regret of any learning algorithm is linear. When considering only the number of timesteps T, Auer (2002) shows that a regret (with respect to the optimal policy) of $O(\sqrt{T}\ln(T))$ can be obtained.

PAC Associative Reinforcement Learning

The previously mentioned works analyze the growth rate of the regret of a learning algorithm with respect to the optimal policy. Another way to approach the problem is to allow the learner some number of timesteps of *exploration*. After the exploration trials, the algorithm is required to output a policy. More specifically, given inputs $0 < \epsilon < 1$ and $0 < \delta < 1$, the algorithm is

required to output an ϵ-optimal policy with probability at least $1 - \delta$. This type of analysis is based on the work by Valiant (1984), and learning algorithms satisfying the above condition are termed *probably approximately correct* (PAC).

Motivated by the work of Kaelbling (1994), Fiechter (1995) developed a PAC algorithm when the true payoff function can be described by a *decision list* over the action and input vector. Building on both works, Strehl, Mesterharm, Littman, and Hirsh (2006) showed that a class of associative reinforcement learning problems can be solved efficiently, in a PAC sense, when given a learning algorithm for efficiently solving classification problems.

Recommended Reading

Section 6.1 of the survey by Kaelbling, Littman, and Moore (1996) presents a nice overview of several techniques for the associative reinforcement-learning problem, such as CRBP (Ackley and Littman, 1990), ARC (Sutton, 1984), and REINFORCE (Williams, 1992).

Abe, N., & Long, P. M. (1999). Associative reinforcement learning using linear probabilistic concepts. In *Proceedings of the 16th international conference on machine learning* (pp. 3–11).

Ackley, D. H., & Littman, M. L. (1990). Generalization and scaling in reinforcement learning. In *Advances in neural information processing systems 2* (pp. 550–557). San Mateo, CA: Morgan Kaufmann.

Auer, P. (2002). Using confidence bounds for exploitation–exploration trade-offs. *Journal of Machine Learning Research, 3*, 397–422.

Fiechter, C.-N. (1995). PAC associative reinforcement learning. Unpublished manuscript.

Kaelbling, L. P. (1994). Associative reinforcement learning: Functions in *k*-DNF. *Machine Learning, 15*, 279–298.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Strehl, A. L., Mesterharm, C., Littman, M. L., & Hirsh, H. (2006). Experience-efficient learning in associative bandit problems. In *ICML-06: Proceedings of the 23rd international conference on machine learning*, Pittsburgh, Pennsylvania (pp. 889–896).

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning*. Doctoral dissertation, University of Massachusetts, Amherst, MA.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM, 27*, 1134–1142.

Wang, C.-C., Kulkarni, S. R., & Poor, H. V. (2005). Bandit problems with side observations. *IEEE Transactions on Automatic Control, 50*, 3988–3993.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*, 229–256.

Attribute

Chris Drummond
National Research Council of Canada, Ottawa, ON, Canada

Synonyms

Characteristic; Feature; Property; Trait

Definition

Attributes are properties of things, ways that we, as humans, might describe them. If we were talking about the appearance of our friends, we might describe one of them as "sex female," "hair brown," "height 5 ft 7 in." Linguistically, this is rather terse, but this very terseness has the advantage of limiting ambiguity. The attributes are sex, hair color, and height. For each friend, we could give the appropriate values to go along with each attribute, some examples are shown in Table 1. Attribute-value pairs are a standard way of describing things within the machine learning community. Traditionally, values have come in one of three types: binary, sex has two values; nominal, hair color has many values; real, height has an ordered set of values. Ideally, the attribute-value pairs are sufficient to describe some things accurately and to tell them apart from others. What might be described is very varied, so the attributes themselves will vary widely.

Motivation and Background

For machine learning to be successful, we need a language to describe everyday things that is sufficiently powerful to capture the similarities and differences between them and yet is computationally easy to manage. The idea that a sufficient number of attribute-value

Attribute. Table 1 Some friends

Sex	Hair color	Height
Male	Black	6 ft 2 in.
Female	Brown	5 ft 7 in.
Female	Blond	5 ft 9 in.
Male	Brown	5 ft 10 in.

pairs would meet this requirement is an intuitive one. It has also been studied extensively in philosophy and psychology, as a way that humans represent things mentally. In the early days of artificial intelligence research, the frame (Minsky, 1974) became a common way of representing knowledge. We have, in many ways, inherited this representation, attribute-value pairs sharing much in common with the labeled slots for values used in frames. In addition, the data for many practical problems comes in this form. Popular methods of storing and manipulating data such as relational databases, and less formal structures such as spread sheets, have columns as attributes and cells as values. So, attribute-value pairs are a ubiquitous way of representing data.

Future Directions

The notion of an attribute-value pair is so well entrenched in machine learning that it is difficult to perceive what might replace it. As, in many practical applications, the data comes in this form, this representation will undoubtedly be around for some time. One change that is occurring is the growing complexity of attribute-values. Traditionally, we have used the simple value types, binary, nominal, and real, discussed earlier. But to effectively describe many things, we need to extend this simple language and use more complex values. For example, in ►data mining applied to multimedia, more new complex representations abound. Sound and video streams, images, and various properties of them, are just a few examples (Cord et al., 2005; Simoff & Djeraba, 2000).

Perhaps, the most significant change is away from attributes, albeit with complex values, to structural forms where the relationship between things is included. As Quinlan (1996) states "Data may concern objects or observations with arbitrarily complex structure that cannot be captured by the values of a predetermined set of attributes." There is a large and growing community of researchers in ►relational learning. This is evidenced by the number, and growing frequency, of recent workshops at the International Conference for Machine Learning (Cord et al., 2005; De Raedt & Kramer, 2000; Dietterich, Getoor, & Murphy, 2004; Fern, Getoor, & Milch, 2006).

Limitations

In philosophy there is the idea of essence, the properties an object must have to be what it is. In machine learning, particularly in practical applications, we get what we are given and have little control in the choice of attributes and their range of values. If domain experts have chosen the attributes, we might hope that they are properties that can be readily ascertained and are relevant to the task at the hand. For example, when describing one of our friends, we would not say Fred is the one with the spleen. It is not only difficult to observe, it is also poor at discriminating between people. Data are collected for many reasons. In medical applications, all sorts of attribute-values would be collected on patients. Most are unlikely to be important to the current task. An important part of learning is ►feature extraction, determining which attributes are necessary for learning.

Whether or not attribute-value pairs are an essential representation for the type of learning required in the development, and functioning, of intelligent agents, remains to be seen. Attribute-values readily capture symbolic information, typically at the level of words that humans naturally use. But if our agents need to move around in their environment, recognizing what they encounter, we might need a different nonlinguistic representation. Certainly, other representations based on a much finer granularity of features, and more holistic in nature, have been central to areas such as ►neural networks for some time. In research into ►dynamic systems, attractors in a sensor space might be more realistic that attribute-values (See chapter on ►Classification).

Recommended Reading

Cord, M., Dahyot, R., Cunningham, P., & Sziranyi, T. (Eds.). (2005). Workshop on machine learning techniques for processing multimedia content. In *Proceedings of the twenty-second international conference on machine learning*.

De Raedt, L., & Kramer, S. (Eds.). (2000). In *Proceedings of the seventeenth international conference on machine learning*. Workshop on attribute-value and relational learning: Crossing the boundaries, Stanford University, Palo Alto, CA.

Dietterich, T., Getoor, L., & Murphy, K. (Eds.). (2004). In *Proceedings of the twenty-first international conference on machine learning*. Workshop on statistical relational learning and its connections to other fields.

Fern, A., Getoor, L., & Milch, B. (Eds.). (2006). In *Proceedings of the twenty-fourth international conference on machine learning*. Workshop on open problems in statistical relational learning.

Minsky, M. (1974). A framework for representing knowledge. Technical report, Massachusetts Institute of Technology, Cambridge, MA.

Quinlan, J. R. (1996). Learning first-order definitions of functions. *Journal of Artificial Intelligence Research, 5*, 139–161.

Simoff, S. J., & Djeraba, C. (Eds.). (2000). In *Proceedings of the sixth international conference on knowledge discovery and data mining*. Workshop on multimedia data mining.

Attribute Selection

►Feature Selection

Attribute-Value Learning

Attribute-value learning refers to any learning task in which the each ►Instance is described by the values of some finite set of attributes (see ►Attribute). Each of these instances is often represented as a vector of attribute values, each position in the vector corresponding to a unique attribute.

AUC

►Area Under Curve

Autonomous Helicopter Flight Using Reinforcement Learning

Adam Coates[1], Pieter Abbeel[2], Andrew Y. Ng[3]
[1]Stanford University, Stanford, CA, USA
[2]University of California, Berkeley, CA, USA
[3]Stanford University, Stanford, CA, USA

Definition

Helicopter flight is a highly challenging control problem. While it is possible to obtain controllers for simple maneuvers (like hovering) by traditional manual design procedures, this approach is tedious and typically requires many hours of adjustments and flight testing, even for an experienced control engineer. For complex maneuvers, such as aerobatic routines, this approach is likely infeasible. In contrast, ►reinforcement learning (RL) algorithms enable faster and more automated design of controllers. Model-based RL algorithms have been used successfully for autonomous helicopter flight for hovering, forward flight and, using apprenticeship learning methods for expert-level aerobatics. In model-based RL, first one builds a model of the helicopter dynamics and specifies the task using a reward function. Then, given the model and the reward function, the RL algorithm finds a controller that maximizes the expected sum of rewards accumulated over time.

Motivation and Background

Autonomous helicopter flight represents a challenging control problem and is widely regarded as being significantly harder than control of fixed-wing aircraft. (See, e.g., Leishman, (2000); Seddon, (1990)). At the same time, helicopters provide unique capabilities such as in-place hover, vertical takeoff and landing, and low-speed maneuvering. These capabilities make helicopter control an important research problem for many practical applications.

Building autonomous flight controllers for helicopters, however, is far from trivial. When done by hand, it can require many hours of tuning by experts with extensive prior knowledge about helicopter dynamics. Meanwhile, the automated development of helicopter controllers has been a major success story for RL methods. Controllers built using RL algorithms have established state-of-the-art performance for both basic flight maneuvers, such as hovering and forward flight (Bagnell & Schneider, 2001; Ng, Kim, Jordan, & Sastry, 2004), as well as being among the only successful methods for advanced aerobatic stunts. Autonomous helicopter aerobatics has been successfully tackled using the innovation of "apprenticeship learning," where the algorithm learns by watching a human demonstrator (Abbeel & Ng, 2004). These methods have enabled autonomous helicopters to fly aerobatics as well as an expert human pilot, and often even better (Coates, Abbeel, & Ng, 2008).

Developing autonomous flight controllers for helicopters is challenging for a number of reasons:

1. Helicopters have unstable, high-dimensional, asymmetric, noisy, nonlinear, non-minimum phase dynamics. As a consequence, all successful helicopter flight

controllers (to date) have many parameters. Controllers with 10–100 gains are not atypical. Hand engineering the right setting for each of the parameters is difficult and time consuming, especially since their effects on performance are often highly coupled through the helicopter's complicated dynamics. Moreover, the unstable dynamics, especially in the low-speed flight regime, complicates flight testing.

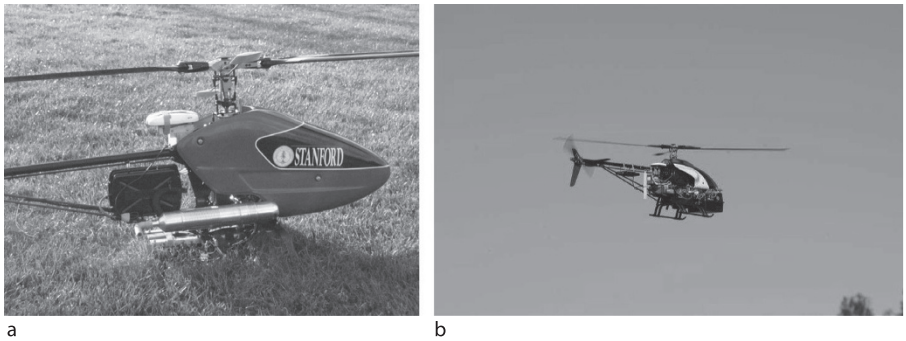
2. Helicopters are underactuated: their position and orientation is representable using six parameters, but they have only four control inputs. Thus helicopter control requires significant planning and making trade-offs between errors in orientation and errors in desired position.
3. Helicopters have highly complex dynamics: Even though we describe the helicopter as having a twelve dimensional state (position, velocity, orientation, and angular velocity), the true dynamics are significantly more complicated. To determine the precise effects of the inputs, one would have to consider the airflow in a large volume around the helicopter, as well as the parasitic coupling between the different inputs, the engine performance, and the non-rigidity of the rotor blades. Highly accurate simulators are thus difficult to create, and controllers developed in simulation must be sufficiently robust that they generalize to the real helicopter in spite of the simulator's imperfections.
4. Sensing capabilities are often poor: For small remotely controlled (RC) helicopters, sensing is limited because the on-board sensors must deal with a large amount of vibration caused by the helicopter blades rotating at about 30 Hz, as well as higher frequency noise from the engine. Although noise at these frequencies (which are well above the roughly 10 Hz at which the helicopter dynamics can be modeled reasonably) might be easily removed by low pass filtering, this introduces latency and damping effects that are detrimental to control performance. As a consequence, helicopter flight controllers have to be robust to noise and/or latency in the state estimates to work well in practice.

Typical Hardware Setup

A typical autonomous helicopter has several basic sensors on board. An Inertial Measurement Unit (IMU) measures angular rates and linear accelerations for each of the helicopter's three axes. A 3-axis magnetometer senses the direction of the Earth's magnetic field, similar to a magnetic compass (Fig. 1).

Attitude-only sensing, as provided by the inertial and magnetic sensors, is insufficient for precise, stable hovering, and slow-speed maneuvers. These maneuvers require that the helicopter maintain relatively tight control over its position error, and hence high-quality position sensing is needed. GPS is often used to determine helicopter position (with carrier-phase GPS units achieving sub-decimeter accuracy), but vision-based solutions have also been employed (Abbeel, Coates, Quigley, & Ng, 2007; Coates et al., 2008; Saripalli, Montgomery, & Sukhatme, 2003).

Vibration adds errors to the sensor measurements and may damage the sensors themselves, hence significant effort may be required to mount the sensors on the airframe (Dunbabin, Brosnan, Roberts, & Corke, 2004). Provided there is no aliasing, sensor errors added by



a b

Autonomous Helicopter Flight Using Reinforcement Learning. Figure 1. (a) Stanford University's instrumented XCell Tempest autonomous helicopter. (b) A Bergen Industrial Twin autonomous helicopter with sensors and on-board computer

vibration can be removed by using a digital filter on the measurements (though, again, one must be careful to avoid adding too much latency).

Sensor data from the aircraft sensors is used to estimate the state of the helicopter for use by the control algorithm. This is usually done with an extended Kalman filter (EKF). A unimodal distribution (as computed by the EKF) suffices to represent the uncertainty in the state estimates and it is common practice to use the mode of the distribution as the state estimate for feedback control. In general the accuracy obtained with this method is sufficiently high that one can treat the state as fully observed.

Most autonomous helicopters have an on-board computer that runs the EKF and the control algorithm (Gavrilets, Martinos, Mettler, & Feron, 2002a; La Civita, Papageorgiou, Messner, & Kanade, 2006; Ng et al., 2004). However, it is also possible to use ground-based computers by sending sensor data by wireless to the ground, and then transmitting control signals back to the helicopter through the pilot's RC transmitter (Abbeel et al., 2007; Coates et al., 2008).

Helicopter State and Controls

The helicopter state s is defined by its position (p_x, p_y, p_z), orientation (which could be expressed using a unit quaternion q), velocity (v_x, v_y, v_z) and angular velocity $(\omega_x, \omega_y, \omega_z)$.

The helicopter is controlled via a 4-dimensional action space:

1. u_1 and u_2: The lateral (left-right) and longitudinal (front-back) cyclic pitch controls (together referred to as the "cyclic" controls) cause the helicopter to roll left or right, and pitch forward or backward, respectively.
2. u_3: The tail rotor pitch control affects tail rotor thrust, and can be used to yaw (turn) the helicopter about its vertical axis. In analogy to airplane control, the tail rotor control is commonly referred to as "rudder."
3. u_4: The collective pitch control (often referred to simply as "collective"), increases and decreases the pitch of the main rotor blades, thus increasing or decreasing the vertical thrust produced as the blades sweep through the air.

By using the cyclic and rudder controls, the pilot can rotate the helicopter into any orientation. This allows the pilot to direct the thrust of the main rotor in any particular direction, and thus fly in any direction, by rotating the helicopter appropriately.

Helicopter Flight as an RL Problem

Formulation

A RL problem can be described by a tuple $(S, \mathcal{A}, T, H, s(0), R)$, which is referred to as a ▶Markov decision process (MDP). Here S is the set of states; $\mathcal{A}$ is the set of actions or inputs; T is the dynamics model, which is a set of probability distributions $\{P^t_{su}\}$ ($P^t_{su}(s'|s,u)$ is the probability of being in state s' at time $t+1$, given the state and action at time t are s and u); H is the horizon or number of time steps of interest; $s(0) \in S$ is the initial state; $R : S \times \mathcal{A} \to \mathbb{R}$ is the reward function.

A policy $\pi = (\mu_0, \mu_1, \ldots, \mu_H)$ is a tuple of mappings from states S to actions $\mathcal{A}$, one mapping for each time $t = 0, \ldots, H$. The expected sum of rewards when acting according to a policy π is given by: $U(\pi) = \mathrm{E}[\sum_{t=0}^{H} R(s(t), u(t))|\pi]$. The optimal policy π^* for an MDP $(S, \mathcal{A}, T, H, s(0), R)$ is the policy that maximizes the expected sum of rewards. In particular, the optimal policy is given by: $\pi^* = \arg\max_\pi U(\pi)$.

The common approach to finding a good policy for autonomous helicopter flight proceeds in two steps: First one collects data from manual helicopter flights to build a model (One could also build a helicopter model by directly measuring physical parameters such as mass, rotor span, etc. However, even when this approach is pursued, one often resorts to collecting flight data to complete the model.). Then one solves the MDP comprised of the model and some chosen reward function. Although the controller obtained, in principle, is only optimal for the learned simulator model, it has been shown in various settings that optimal controllers perform well even when the model has some inaccuracies (see, e.g., Anderson & Moore, (1989)).

Modeling

One way to create a helicopter model is to use direct knowledge of aerodynamics to derive an explicit mathematical model. This model will depends on a number of parameters that are particular to the helicopter

being flown. Many of the parameters may be measured directly (e.g., mass, rotational inertia), while others must be estimated from flight experiments. This approach has been used successfully on several systems (see, e.g., (Gavrilets, Martinos, Mettler, & Feron, 2002b; Gavrilets, Mettler, & Feron, 2001; La Civita, 2003)). However, substantial expert aerodynamics knowledge is required for this modeling approach. Moreover, these models tend to cover only a limited fraction of the flight envelope.

Alternatively, one can learn a model of the dynamics directly from flight data, with only limited *a priori* knowledge of the helicopter's dynamics. Data is usually collected from a series of manually controlled flights. These flights involve the human sweeping the control sticks back and forth at varying frequencies to cover as much of the flight envelope as possible, while recording the helicopter's state and the pilot inputs at each instant.

Given a corpus of flight data, various different learning algorithms can be used to learn the underlying model of the helicopter dynamics.

If one is only interested in a single flight regime, one could learn a linear model that maps from the current state and action to the next state. Such a model can be easily estimated using ►linear regression (While the methods presented here emphasize time-domain estimation, frequency domain estimation is also possible for the special case of estimating linear models (Tischler & Cauffman, 1992).). Linear models are restricted to small flight regimes (e.g., hover or inverted hover) and do not immediately generalize to full-envelope flight. To cover a broader flight regime, non parametric algorithms such as locally-weighted linear regression have been used (Bagnell & Schneider, 2001; Ng et al., 2004). Non parametric models that map from current state and action to next state can, in principle, cover the entire flight regime. Unfortunately, one must collect large amounts of data to obtain an accurate model and the models are often quite slow to evaluate.

An alternative way to increase the expressiveness of the model, without resorting to non parametric methods, is to consider a time-varying model where the dynamics are explicitly allowed to depend on time. One can then proceed to compute simpler (say, linear) parametric models for each choice of the time parameter. This method is effective when learning a model specific to a trajectory whose dynamics are repeatable but vary as the aircraft travels along the trajectory. Since this method can also require a great deal of data (similar to nonparametric methods) in practice, it is helpful to begin with a non-time-varying parametric model fit from a large amount of data, and then augment it with a time-varying component that has fewer parameters (Abbeel, Quigley, & Ng, 2006; Coates et al., 2008).

One can also take advantage of symmetry in the helicopter dynamics to reduce the amount of data needed to fit a parametric model. In Abbeel, Ganapathi, and Ng (2006) observe that – in a coordinate frame attached to the helicopter – the helicopter dynamics are essentially the same for any orientation (or position) once the effect of gravity is removed. They learn a model that predicts (angular and linear) accelerations – except for the effects of gravity – in the helicopter frame as a function of the inputs and the (angular and linear) velocity in the helicopter frame. This leads to a lower-dimensional learning problem, which requires significantly less data. To simulate the helicopter dynamics over time, the predicted accelerations augmented with the effects of gravity are integrated over time to obtain velocity, angular rates, position, and orientation.

Abbeel et al. (2007) used this approach to learn a helicopter model that was later used for autonomous aerobatic helicopter flight maneuvers covering a large part of the flight envelope. Significantly less data is required to learn a model using the gravity-free parameterization compared to a parameterization that directly predicts the next state as a function of current state and actions (as was used in Bagnell and Schneider (2001), Ng et al. (2004)). Abbeel et al. evaluate their model by checking its simulation accuracy over longer time scales than just a one-step acceleration prediction. Such an evaluation criterion maps more directly to the reinforcement learning objective of maximizing the expected sum of rewards accumulated over time (see also Abbeel & Ng, (2005b)).

The models considered above are deterministic. This normally would allow us to drop the expectation when evaluating a policy according to $E[\sum_{t=0}^{H} R(s(t), u(t))|\pi]$. However, it is common to add stochasticity to account for unmodeled effects. Abbeel et al. (2007) and Ng et al. (2004) include additive process noise in

their models. Bagnell and Schneider (2001) go further, learning a distribution over models. Their policy must then perform well, on expectation, for a (deterministic) model selected randomly from the distribution.

Control Problem Solution Methods

Given a model of the helicopter, we now seek a policy π that maximizes the expected sum of rewards $U(\pi) = \mathrm{E}[\sum_{t=0}^{H} R(s(t), u(t))|\pi]$ achieved when acting according to the policy π.

Policy Search General policy search algorithms can be employed to search for optimal policies for the MDP based on the learned model. Given a policy π, we can directly try to optimize the objective $U(\pi)$. Unfortunately, $U(\pi)$ is an expectation over a complicated distribution making it impractical to evaluate the expectation exactly in general.

One solution is to approximate the expectation $U(\pi)$ by Monte Carlo sampling: under certain boundedness assumptions the empirical average of the sum of rewards accumulated over time will give a good estimate $\hat{U}(\pi)$ of the expectation $U(\pi)$. Naively Applying Monte Carlo sampling to accurately compute, e.g., the local gradient from the difference in function value at nearby points, requires very large amounts of samples due to the stochasticity in the function evaluation.

To get around this hurdle, the PEGASUS algorithm (Ng & Jordan, 2000) can be used to convert the stochastic optimization problem into a deterministic one. When evaluating by averaging over n simulations, PEGASUS initially fixes n random seeds. For each policy evaluation, the same n random seeds are used so that the simulator is now deterministic. In particular, multiple evaluations of the same policy will result in the same computed reward. A search algorithm can then be applied to the deterministic problem to find an optimum.

The PEGASUS algorithm coupled with a simple local policy search was used by Ng et al. (2004) to develop a policy for their autonomous helicopter that successfully sustains inverted hover. Bagnell and Schneider (2001) proceed similarly, but use the "amoeba" search algorithm (Nelder & Mead, 1964) for policy search.

Because of the searching involved, the policy class must generally have low dimension. Nonetheless, it is often possible to find good policies within these policy classes. The policy class of Ng et al. (2004), for instance, is a decoupled, linear PD controller with a sparse dependence on the state variables (For instance, the linear controller for the pitch axis is parametrized as $u_2 = c_0(p_x - p_x^*) + c_1(v_x - v_x^*) + c_2\theta$, which has just three parameters while the entire state is nine dimensional. Here, $p_\cdot$, $v_\cdot$, and $p_\cdot^*$, $v_\cdot^*$, respectively, are the actual and desired position and velocity. θ denotes the pitch angle.). The sparsity reduces the policy class to just nine parameters. In Bagnell and Schneider (2001), two-layer neural network structures are used with a similar sparse dependence on the state variables. Two neural networks with five parameters each are learned for the cyclic controls.

Differential Dynamic Programming Abbeel et al. (2007) use differential dynamic programming (DDP) for the task of aerobatic trajectory following. DDP (Jacobson & Mayne, 1970) works by iteratively approximating the MDP as linear quadratic regulator (LQR) problems. The LQR control problem is a special class of MDPs, for which the optimal policy can be computed efficiently. In LQR the set of states is given by $S = \mathbb{R}^n$, the set of actions/inputs is given by $\mathcal{A} = \mathbb{R}^p$, and the dynamics model is given by:

$$s(t+1) = A(t)s(t) + B(t)u(t) + w(t),$$

where for all $t = 0, \ldots, H$ we have that $A(t) \in \mathbb{R}^{n \times n}$, $B(t) \in \mathbb{R}^{n \times p}$ and $w(t)$ is a mean zero random variable (with finite variance). The reward for being in state $s(t)$ and taking action $u(t)$ is given by:

$$-s(t)^\top Q(t)s(t) - u(t)^\top R(t)u(t).$$

Here $Q(t), R(t)$ are positive semi-definite matrices which parameterize the reward function. It is well-known that the optimal policy for the LQR control problem is a linear feedback controller which can be efficiently computed using dynamic programming (see, e.g., Anderson & Moore, (1989), for details on linear quadratic methods.)

DDP approximately solves general continuous state-space MDPs by iterating the following two steps until convergence:

1. Compute a linear approximation to the nonlinear dynamics and a quadratic approximation to

the reward function around the trajectory obtained when executing the current policy in simulation.
2. Compute the optimal policy for the LQR problem obtained in Step 1 and set the current policy equal to the optimal policy for the LQR problem.

During the first iteration, the linearizations are performed around the target trajectory for the maneuver, since an initial policy is not available.

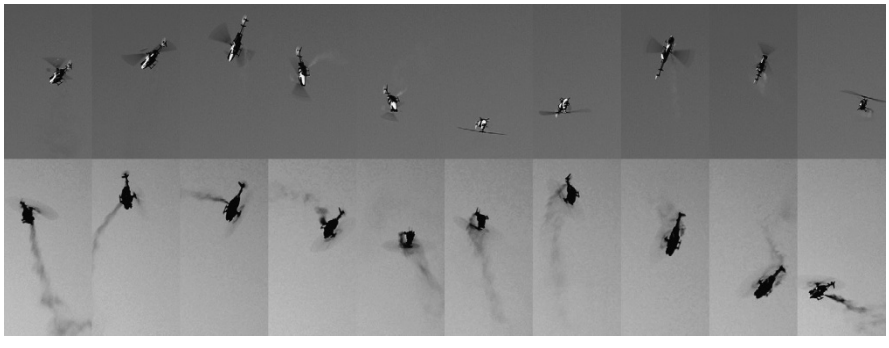
This method is used to perform autonomous flips, rolls, and "funnels" (high-speed sideways flight in a circle) in Abbeel et al. (2007) and autonomous autorotation (Autorotation is an emergency maneuver that allows a skilled pilot to glide a helicopter to a safe landing in the event of an engine failure or tail-rotor failure.) in Abbeel, Coates, Hunter, and Ng (2008), Fig. 2.

While DDP computes a solution to the non-linear optimization problem, it relies on the accuracy of the non-linear model to correctly predict the trajectory that will be flown by the helicopter. This prediction is used in Step 1 above to linearize the dynamics. In practice, the helicopter will often not follow the predicted trajectory closely (due to stochasticity and modeling errors), and thus the linearization will become a highly inaccurate approximation of the non-linear model. A common solution to this, applied by Coates et al. (2008), is to compute the DDP solution online, linearizing around a trajectory that begins at the current helicopter state. This ensures that the model is always linearized around a trajectory near the helicopter's actual flight path.

Apprenticeship Learning and Inverse RL In computing a policy for an MDP, simply finding a solution (using any method) that performs well in simulation may not be enough. One may need to adjust both the model and reward function based on the results of flight testing. Modeling error may result in controllers that fly perfectly in simulation but perform poorly or fail entirely in reality. Because helicopter dynamics are difficult to model exactly, this problem is fairly common. Meanwhile, a poor reward function can result in a controller that is not robust to modeling errors or unpredicted perturbations (e.g., it may use large control inputs that are unsafe in practice). If a human "expert" is available to demonstrate the maneuver, this demonstration flight can be leveraged to obtain a better model and reward function.

The reward function encodes both the trajectory that the helicopter should follow, as well as the trade-offs between different types of errors. If the desired trajectory is infeasible (either in the non-linear simulation or in reality), this results in a significantly more difficult control problem. Also, if the trade-offs are not specified correctly, the helicopter may be unable to compensate for significant deviations from the desired trajectory. For instance, a typical reward function for hovering implicitly specifies a trade-off between position error and orientation error (it is possible to reduce one error, but usually at the cost of increasing the other). If this trade-off is incorrectly chosen, the controller may be pushed off course by wind (if it tries too hard to keep the helicopter level) or, conversely, may tilt the helicopter to an unsafe attitude while trying to correct for a large position error.

We can use demonstrations from an expert pilot to recover both a good choice for the desired trajectory as well as good choices of reward weights for errors relative to this trajectory. In apprenticeship learning, we are given a set of N recorded state and control sequences,



Autonomous Helicopter Flight Using Reinforcement Learning. Figure 2. Snapshots of an autonomous helicopter performing in-place flips and rolls

$\{s_k(t), u_k(t)\}_{t=0}^{H}$ for $k = 1, \ldots, N$, from demonstration flights by an expert pilot. Coates et al. (2008) note that these demonstrations may be sub-optimal but are often sub-optimal in different ways. They suggest that a large number of expert demonstrations may implicitly encode the optimal trajectory and propose a generative model that explains the expert demonstrations as stochastic instantiations of an "ideal" trajectory. This is the desired trajectory that the expert has in mind but is unable to demonstrate exactly. Using an Expectation-Maximization (Dempster, Laird, & Rubin, 1977) algorithm, they infer the desired trajectory and use this as the target trajectory in their reward function.

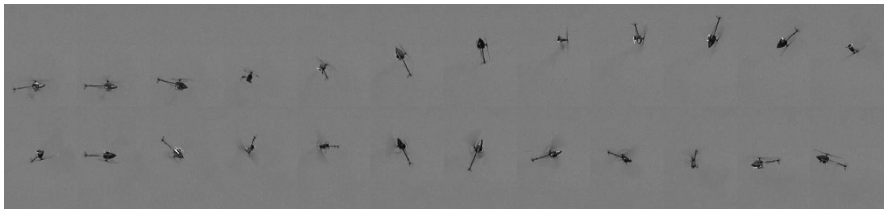
A good choice of reward weights (for errors relative to the desired trajectory) can be recovered using inverse reinforcement learning (Abbeel & Ng, 2004; Ng & Russell, 2000). Suppose the reward function is written as a linear combination of features as follows: $R(s,u) = c_0\phi_0(s,u) + c_1\phi_1(s,u) + \cdots$. For a single recorded demonstration, $\{s(t), u(t)\}_{t=0}^{H}$, the pilot's accumulated reward corresponding to each feature can be computed as $c_i\phi_i^* = c_i \sum_{t=0}^{H} \phi_i(s(t), u(t))$. If the pilot out-performs the autonomous flight controller with respect to a particular feature ϕ_i, this indicates that the pilot's own "reward function" places a higher value on that feature, and hence its weight c_i should be increased. Using this procedure, a good choice of reward function that makes trade-offs similar to that of a human pilot can be recovered. This method has been used to guide the choice of reward for many maneuvers during flight testing (Abbeel et al., 2007, 2008; Coates et al., 2008).

In addition to learning a better reward function from pilot demonstration, one can also use the pilot demonstration to improve the model directly and attempt to reduce modeling error. Coates et al. (2008), for instance, use errors observed in expert demonstrations to jointly infer an improved dynamics model along with the desired trajectory. Abbeel et al. (2007), however, have proposed the following alternating procedure that is broadly applicable (see also Abbeel and Ng (2005a) for details):

1. Collect data from a human pilot flying the desired maneuvers with the helicopter. Learn a model from the data.
2. Find a controller that works in simulation based on the current model.
3. Test the controller on the helicopter. If it works, we are done. Otherwise, use the data from the test flight to learn a new (improved) model and go back to Step 2.

This procedure has similarities with model-based RL and with the common approach in control to first perform system identification and then find a controller using the resulting model. However, the key insight from Abbeel and Ng (2005a) is that this procedure is guaranteed to converge to expert performance in a polynomial number of iterations. The authors report needing at most three iterations in practice. Importantly, unlike the E^3 family of algorithms (Kearns & Singh, 2002), this procedure does not require explicit exploration policies. One only needs to test controllers that try to fly as well as possible (according to the current choice of dynamics model) (Indeed, the E^3-family of algorithms (Kearns & Singh, 2002) and its extensions (Brafman & Tennenholtz, 2002; Kakade, Kearns, & Langford, 2003; Kearns & Koller, 1999) proceed by generating "exploration" policies, which try to visit inaccurately modeled parts of the state space. Unfortunately, such exploration policies do not even try to fly the helicopter well, and thus would almost invariably lead to crashes.).

The apprenticeship learning algorithms described above have been used to fly the most advanced autonomous maneuvers to date. The apprenticeship learning algorithm of Coates et al. (2008), for example, has been used to attain expert level performance on challenging aerobatic maneuvers as well as entire airshows composed of many maneuvers in rapid sequence. These maneuvers include in-place flips and rolls, tic-tocs ("Tic-toc" is a maneuver where the helicopter pitches forward and backward with its nose pointed toward the sky (resembling an inverted clock pendulum).), and chaos ("Chaos" is a maneuver where the helicopter flips in-place but does so while continuously pirouetting at a high rate. Visually, the helicopter body appears to tumble chaotically while nevertheless remaining in roughly the same position.) (see Fig. 3). These maneuvers are considered among the most challenging possible and can only be performed



Autonomous Helicopter Flight Using Reinforcement Learning. Figure 3. Snapshot sequence of an autonomous helicopter flying a "chaos" maneuver using apprenticeship learning methods. Beginning from top-left and proceeding left-to-right, top-to-bottom, the helicopter performs a flip while pirouetting counter-clockwise about its vertical axis. (This maneuver has been demonstrated continuously for as long as 15 cycles like the one shown here)

Autonomous Helicopter Flight Using Reinforcement Learning. Figure 4. Super-imposed sequence of images of autonomous autorotation landings (from Abbeel et al. (2008))

by advanced human pilots. In fact, Coates et al. (2008) show that their learned controller performance can even exceed the performance of the expert pilot providing the demonstrations, putting many of the maneuvers on par with professional pilots (Fig. 4).

A similar approach has been used in Abbeel et al. (2008) to perform the first successful autonomous autorotations. Their aircraft has performed more than 30 autonomous landings successfully without engine power.

Not only do apprenticeship methods achieve state-of-the-art performance, but they are among the fastest learning methods available, as they obviate the need for arduous hand tuning by engineers. Coates et al. (2008), for instance, report that entire airshows can be created from scratch with just 1 h of work. This is in stark contrast to previous approaches that may have required hours or even days of tuning for relatively simple maneuvers.

Conclusion

Helicopter control is a challenging control problem and has recently seen major successes with the application of learning algorithms. This Chapter has shown how each step of the control design process can be automated using machine learning algorithms for system identification and reinforcment learning algorithms for control. It has also shown how apprenticeship learning algorithms can be employed to achieve

expert-level performance on challenging aerobatic maneuvers when an expert pilot can provide demonstrations. Autonomous helicopters with control systems developed using these methods are now capable of flying advanced aerobatic maneuvers (including flips, rolls, tic-tocs, chaos, and auto-rotation) at the level of expert human pilots.

Cross References

►Apprenticeship Learning
►Reinforcement Learning
►Reward Shaping

Recommended Reading

Abbeel, P., Coates, A., Hunter, T., & Ng, A. Y. (2008). Autonomous autorotation of an rc helicopter. In *ISER 11*.

Abbeel, P., Coates, A., Quigley, M., & Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *NIPS 19* (pp. 1–8). Vancouver.

Abbeel, P., Ganapathi, V., & Ng, A. Y. (2006). Learning vehicular dynamics with application to modeling helicopters. In *NIPS 18*. Vancouver.

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the international conference on machine learning*. New York: ACM.

Abbeel, P., & Ng, A. Y. (2005a). Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the international conference on machine learning*. New York: ACM

Abbeel, P., & Ng, A. Y. (2005b). Learning first order Markov models for control. In *NIPS 18*.

Abbeel, P., Quigley, M., & Ng, A. Y. (2006). Using inaccurate models in reinforcement learning. In *ICML '06: Proceedings of the 23rd international conference on machine learning* (pp. 1–8). New York: ACM.

Anderson, B., & Moore, J. (1989). *Optimal control: linear quadratic methods*. Princeton, NJ: Prentice-Hall.

Bagnell, J., & Schneider, J. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *International conference on robotics and automation*. Canada: IEEE.

Brafman, R. I., & Tennenholtz, M. (2002). R-max, a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research, 3*, 213–231.

Coates, A., Abbeel, P., & Ng, A. Y. (2008). Learning for control from multiple demonstrations. In *ICML '08: Proceedings of the 25th international conference on machine learning*.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, 39*1, 1–38.

Dunbabin, M., Brosnan, S., Roberts, J., & Corke, P. (2004). Vibration isolation for autonomous helicopter flight. In *Proceedings of the IEEE international conference on robotics and automation* (Vol. 4, pp. 3609–3615).

Gavrilets, V., Martinos, I., Mettler, B., & Feron, E. (2002a). Control logic for automated aerobatic flight of miniature helicopter. In *AIAA guidance, navigation and control conference*. Cambridge, MA: Massachusetts Institute of Technology.

Gavrilets, V., Martinos, I., Mettler, B., & Feron, E. (2002b). Flight test and simulation results for an autonomous aerobatic helicopter. In *AIAA/IEEE digital avionics systems conference*.

Gavrilets, V., Mettler, B., & Feron, E. (2001). Nonlinear model for a small-size acrobatic helicopter. In *AIAA guidance, navigation and control conference* (pp. 1593–1600).

Jacobson, D. H., & Mayne, D. Q. (1970). *Differential dynamic programming*. New York: Elsevier.

Kakade, S., Kearns, M., & Langford, J. (2003). Exploration in metric state spaces. In *Proceedings of the international conference on machine learning*.

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. In *Proceedings of the 16th international joint conference on artificial intelligence*. San Francisco: Morgan Kaufmann.

Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning Journal, 49*(2–3), 209–232.

La Civita, M. (2003). *Integrated modeling and robust control for full-envelope flight of robotic helicopters*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA.

La Civita, M., Papageorgiou, G., Messner, W. C., & Kanade, T. (2006). Design and flight testing of a high-bandwidth $\mathcal{H}_\infty$ loop shaping controller for a robotic helicopter. *Journal of Guidance, Control, and Dynamics, 29*(2), 485–494.

Leishman, J. (2000). *Principles of helicopter aerodynamics*. Cambridge: Cambridge University Press.

Nelder, J. A., & Mead, R. (1964). A simplex method for function minimization. *The Computer Journal, 7*, 308–313.

Ng, A. Y., & Jordan, M. (2000). Pegasus: A policy search method for large MDPs and POMDPs. In *Proceedings of the uncertainty in artificial intelligence 16th conference*. San Francisco: Morgan Kaufmann.

Ng, A. Y., & Russell, S. (2000). Algorithms for inverse reinforcement learning. In *Procedings of the 17th international conference on machine learning* (pp. 663–670). San Francisco: Morgan Kaufmann.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., et al., (2004). Autonomous inverted helicopter flight via reinforcement learning. In *International symposium on experimental robotics*. Berlin: Springer.

Ng, A. Y., Kim, H. J., Jordan, M., & Sastry, S. (2004). Autonomous helicopter flight via reinforcement learning. In *NIPS 16*.

Saripalli, S., Montgomery, J. F., & Sukhatme, G. S. (2003). Visually-guided landing of an unmanned aerial vehicle. *IEEE Transactions on Robotics and Autonomous Systems, 19*(3), 371–380.

Seddon, J. (1990). *Basic helicopter aerodynamics*. In AIAA education series. El Segundo, CA: America Institute of Aeronautics and Astronautics.

Tischler, M. B., & Cauffman, M. G. (1992). Frequency response method for rotorcraft system identification: Flight application to BO-105 couple rotor/fuselage dynamics. *Journal of the American Helicopter Society, 37*.

Average-Cost Neuro-Dynamic Programming

►Average-Reward Reinforcement Learning

Average-Cost Optimization

►Average-Reward Reinforcement Learning

Averaged One-Dependence Estimators

Fei Zheng, Geoffrey I. Webb
Monash University

Synonyms

AODE

Definition

Averaged one-dependence estimators is a ►semi-naive Bayesian Learning method. It performs classification by aggregating the predictions of multiple one-dependence classifiers in which all attributes depend on the same single parent attribute as well as the class.

Classification with AODE

An effective approach to accommodating violations of naive Bayes' attribute independence assumption is to allow an attribute to depend on other non-class attributes. To maintain efficiency it can be desirable to utilize one-dependence classifiers, such as ►Tree Augmented Naive Bayes (TAN), in which each attribute depends upon the class and at most one other attribute. However, most approaches to learning with one-dependence classifiers perform model selection, a process that usually imposes substantial computational overheads and substantially increases variance relative to naive Bayes.

AODE avoids model selection by averaging the predictions of multiple one-dependence classifiers. In each one-dependence classifier, an attribute is selected as the parent of all the other attributes. This attribute is called the *SuperParent* and this type of one-dependence classifier is called a *SuperParent one-dependence estimator* (SPODE). Only those SPODEs with SuperParent x_i where the value of x_i occurs at least m times are used for predicting a class label y for the test instance $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$. For any attribute value x_i,

$$P(y, \mathbf{x}) = P(y, x_i) P(\mathbf{x} \mid y, x_i).$$

This equality holds for every x_i. Therefore,

$$P(y, \mathbf{x}) = \frac{\sum_{1 \le i \le n \wedge F(x_i) \ge m} P(y, x_i) P(\mathbf{x} \mid y, x_i)}{|\{1 \le i \le n \wedge F(x_i) \ge m\}|}, \quad (1)$$

where $F(x_i)$ is the frequency of attribute value x_i in the training sample. Utilizing (1) and the assumption that attributes are independent given the class and the SuperParent x_i, AODE predicts the class for $\mathbf{x}$ by selecting

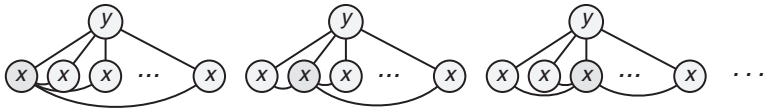
$$\operatorname*{argmax}_y \sum_{1 \le i \le n \wedge F(x_i) \ge m} \hat{P}(y, x_i) \prod_{1 \le j \le n, j \ne i} \hat{P}(x_j \mid y, x_i). \quad (2)$$

It averages over estimates of the terms in (1), rather than the true values, which has the effect of reducing the variance of these estimates.

Figure 1 shows a Markov network representation of an example AODE.

As AODE makes a weaker attribute conditional independence assumption than naive Bayes while still avoiding model selection, it has substantially lower ►bias with a very small increase in ►variance. A number of studies (Webb, Boughton, & Wang, 2005; Zheng & Webb, 2005) have demonstrated that it often has considerably lower zero-one loss than naive Bayes with moderate time complexity. For comparisons with other semi-naive techniques, see ►semi-naive Bayesian learning. One study (Webb, Boughton, & Wang, 2005) found AODE to provide classification accuracy competitive to a state-of-the-art discriminative algorithm, boosted decision trees.

When a new instance is available, like naive Bayes, AODE only needs to update the probability estimates. Therefore, it is also suited to incremental learning.



Averaged One-Dependence Estimators. Figure 1. A Markov network representation of the SPODEs that comprise an example AODE

Cross References

►Bayesian Network
►Naive Bayes
►Semi-Naive Bayesian Learning
►Tree-Augmented Naive Bayes

Recommended Reading

Webb, G. I., Boughton, J., & Wang, Z. (2005). Not so naive Bayes: aggregating one-dependence estimators. *Machine Learning, 58*(1), 5–24.

Zheng, F., & Webb, G. I. (2005). A comparative study of semi-naive Bayes methods in classification learning. In *Proceedings of the Fourth Australasian Data Mining Conference*. (pp. 141–156).

Average-Payoff Reinforcement Learning

►Average-Reward Reinforcement Learning

Average-Reward Reinforcement Learning

Prasad Tadepalli
Oregon State University, Corvallis, OR, USA

Synonyms

ARL; Average-cost neuro-dynamic programming; Average-cost optimization; Average-payoff reinforcement learning

Definition

Average-reward reinforcement learning (ARL) refers to learning policies that optimize the average reward per time step by continually taking actions and observing the outcomes including the next state and the immediate reward.

Motivation and Background

►Reinforcement learning (RL) is the study of programs that improve their performance at some task by receiving rewards and punishments from the environment (Sutton & Barto, 1998). RL has been quite successful in automatic learning of good procedures for complex tasks such as playing Backgammon and scheduling elevators (Crites & Barto, 1998; Tesauro, 1992). In episodic domains in which there is a natural termination condition such as the end of the game in Backgammon, the obvious performance measure to optimize is the expected total reward per episode. But some domains such as elevator scheduling are recurrent, i.e., do not have a natural termination condition. In such cases, total expected reward can be infinite, and we need a different optimization criterion.

In the discounted optimization framework, in each time step, the value of the reward is multiplied by a discount factor $\gamma < 1$, so that the total *discounted* reward is always finite. However, in many domains, there is no natural interpretation for the discount factor γ. A natural performance measure to optimize in such domains is the average reward received per time step. Although one could use a discount factor which is close to 1 to approximate average-reward optimization, an approach that directly optimizes the average reward avoids this additional parameter and often leads to faster convergence in practice.

There is significant theory behind average-reward optimization based on ►Markov decision processes (MDPs) (Puterman, 1994). An MDP is described by a 4-tuple $\langle S, A, P, r \rangle$, where S is a discrete set of states and A is a discrete set of actions. P is a conditional probability distribution over the next states, given the current state and action, and r gives the immediate reward for a given state and action. A *policy* π is a mapping from states to actions. Each policy π induces a Markov process over some set of states. In ergodic MDPs, every policy π forms a single closed set of states, and the average reward per time step of π in the limit of infinite

horizon is independent of the starting state. We call it the "gain" of the policy π, denoted by $\rho(\pi)$, and consider the problem of finding a "gain-optimal policy," π^*, that maximizes $\rho(\pi)$.

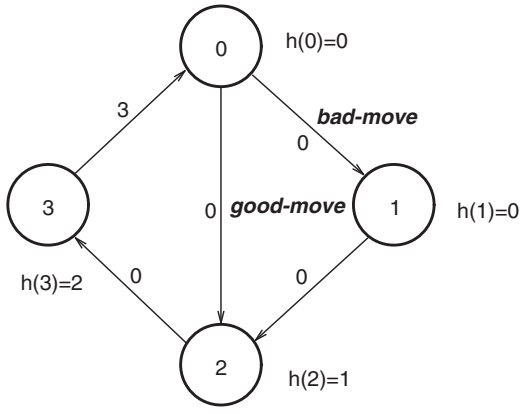
Even though the gain $\rho(\pi)$ of a policy π is independent of the starting state s, the total expected reward in time t is not. It can be denoted by $\rho(\pi)t + h(s)$, where $h(s)$ is a state-dependent bias term. It is the bias values of states that determine which states and actions are preferred, and need to be learned for optimal performance. The following theorem gives the Bellman equation for the bias values of states.

Theorem 3 *For ergodic MDPs, there exist a scalar ρ and a real-valued bias function h over S that satisfy the recurrence relation*

$$\forall s \in S, \quad h(s) = \max_{a \in A} \left\{ r(s,a) + \sum_{s' \in S} P(s'|s,a) h(s') \right\} - \rho. \tag{1}$$

Further, the gain-optimal policy μ^* attains the above maximum for each state s, and ρ is its gain.

Note that any one solution to (1) yields an infinite number of solutions by adding the same constant to all h-values. However, all these sets of h-values will result in the same set of optimal policies μ^*, since the optimal action in a state is determined only by the relative differences between the values of h.



Average-Reward Reinforcement Learning. Figure 1. A simple Markov decision process (MDP) that illustrates the Bellman equation

For example, in Fig. 1, the agent has to select between the actions good-move and bad-move in state 0. If it stays in state 1, it gets an average reward of 1. If it stays in state 2, it gets an average reward of −1. For this domain, $\rho = 1$ for the optimal policy of choosing good-move in state 0. If we arbitrarily set $h(0)$ to 0, then $h(1) = 0$, $h(2) = 1$, and $h(3) = 2$ satisfy the recurrence relations in (1). For example, the difference between $h(3)$ and $h(1)$ is 2, which equals the difference between the immediate reward for the optimal action in state 3 and the optimal average reward 1.

Given the probability model P and the immediate rewards r, the above equations can be solved by White's relative value iteration method by setting the h-value of an arbitrarily chosen reference state to 0 and using synchronous successive approximation (Bertsekas, 1995). There is also a policy iteration approach to determine the optimal policy starting with some arbitrary policy, solving for its values using the value iteration, and updating the policy using one step look-ahead search. The above iteration is repeated until the policy converges (Puterman, 1994).

Model-Based Learning

If the probabilities and the immediate rewards are not known, the system needs to learn them before applying the above methods. A model-based approach called H-learning interleaves model learning with Bellman backups of the value function (Tadepalli & Ok, 1998). This is an average-reward version of ▶adaptive real-time dynamic programming (Barto, Bradtke, & Singh, 1995). The models are learned by collecting samples of state-action-next-state triples $\langle s, a, s' \rangle$ and computing $P(s'|s,a)$ using the maximum likelihood estimation. It then employs the "certainty equivalence principle" by using the current estimates as the true value while updating the h-value of the current state s according to the following update equation derived from the Bellman equation.

$$h(s) \leftarrow \max_{a \in A} \left\{ r(s,a) + \sum_{s' \in S} P(s'|s,a) h(s') \right\} - \rho. \tag{2}$$

One complication in ARL is the estimation of the average reward ρ in the update equations during learning. One could use the current estimate of the long-term average reward, but it is distorted

by the exploratory actions that the agent needs to take to learn about the unexplored parts of the state space. Without the exploratory actions, ARL methods converge to a suboptimal policy. To take this into account, we have from (1), in any state s and a non-exploratory action a that maximizes the right-hand side, $\rho = r(s,a) - h(s) + \sum_{s' \in S} P(s'|S,a)h(s')$. Hence, ρ is estimated by cumulatively averaging $r - h(s) + h(s')$, whenever a greedy action a is executed in state s resulting in state s' and immediate reward r. ρ is updated using the following equation where α is the learning rate.

$$\rho \leftarrow \rho + \alpha(r - h(s) + h(s')). \quad (3)$$

One issue with model-based learning is that the models require too much space and time to learn as tables. In many cases, actions can be represented much more compactly. For example, Tadepalli and Ok (1998) uses dynamic Bayesian networks to represent and learn action models, resulting in significant savings in space and time for learning the models.

Model-Free Learning

One of the disadvantages of the model-based methods is the need to explicitly represent and learn action models. This is completely avoided in model-free methods such as ►Q-learning by learning value functions over state–action pairs. Schwartz's R-learning is an adaptation of Q-learning, which is a discounted reinforcement learning method, to optimize average reward (Schwartz, 1993).

The state–action value $R(s,a)$ can be defined as the expected long-term advantage of executing action a in state s and from then on following the optimal average-reward policy. It can be defined using the bias values h and the optimal average reward ρ as follows.

$$R(s,a) = r(s,a) + \sum_{s' \in S} P(s'|s,a)h(s') - \rho. \quad (4)$$

The main difference with Q-values is that instead of discounting the expected total reward from the next state, we subtract the average reward ρ in each time step, which is the constant penalty for using up a time step. The h value of any state s can now be defined using the following equation.

$$h(s') = \max_u R(s',u). \quad (5)$$

Initially all the R-values are set to 0. When action a is executed in state s, the value of $R(s,a)$ is updated using the update equation

$$R(s,a) \leftarrow (1-\beta)R(s,a) + \beta(r + h(s') - \rho), \quad (6)$$

where β is the learning rate, r is the immediate reward received, s' is the next state, and ρ is the estimate of the average reward of the current greedy policy. In any state s, the greedy action a maximizes the value $R(s,a)$; so R-learning does not need to explicitly learn the immediate reward function $r(s,a)$ or the action models $P(s'|s,a)$, since it does not use them either for the action selection or for updating the R-values.

Both model-free and model-based ARL methods have been evaluated in several experimental domains (Mahadevan, 1996; Tadepalli & Ok, 1998). When there is a compact representation for models and can be learned quickly, the model-based method seems to perform better. It also has the advantage of fewer number of tunable parameters. However, model-free methods are more convenient to implement especially if the models are hard to learn or represent.

Scaling Average-Reward Reinforcement Learning

Just as for discounted reinforcement learning, scaling issues are paramount for ARL. Since the number of states is exponential to the number of relevant state variables, a table-based approach does not scale well. The problem is compounded in multi-agent domains where the number of joint actions is exponential in the number of agents. Several function approximation approaches, such as linear functions, multi-layer perceptrons (Marbach, Mihatsch, & Tsitsiklis, 2000), local ►linear regression (Tadepalli & Ok, 1998), and tile coding (Proper & Tadepalli, 2006) were tried with varying degrees of success.

►Hierarchical reinforcement learning based on the MAXQ framework was also explored in the average-reward setting and was shown to lead to significantly faster convergence. In MAXQ framework, we have a directed acyclic graph, where each node represents a task and stores the value function for that task. Usually, the value function for subtasks depends on fewer state variables than the overall value function and hence can

be more compactly represented. The relevant variables for each subtask are fixed by the designer of the hierarchy, which makes it much easier to learn the value functions. One potential problem with the hierarchical approach is the loss due to the hierarchical constraint on the policy. Despite this limitation, both model-based (Seri & Tadepalli, 2002) and model-free approaches (Ghavamzadeh & Mahadevan, 2006) were shown to yield optimal policies in some domains that satisfy the assumptions of these methods.

Applications

A temporal difference method for average reward based on TD(0) was used to solve a call admission control and routing problem (Marbach et al., 2000). On a modestly sized network of 16 nodes, it was shown that the average-reward TD(0) outperforms the discounted version because it required more careful tuning of its parameters. Similar results were obtained in other domains such as automatic guided vehicle routing (Ghavamzadeh & Mahadevan, 2006) and transfer line optimization (Wang & Mahadevan, 1999).

Convergence Analysis

Unlike their discounted counterparts, both R-Learning and H-Learning lack convergence guarantees. This is because due to the lack of discounting, the updates can no longer be thought of as contraction mappings, and hence the standard theory of stochastic approximation does not apply. Simultaneous update of the average reward ρ and the value functions makes the analysis of these algorithms much more complicated. However, some ARL algorithms have been proved convergent in the limit using analysis based on ordinary differential equations (ODE) (Abounadi, Bertsekas, & Borkar, 2002). The main idea is to turn to ordinary differential equations that are closely tracked by the update equations and use two time-scale analysis to show convergence. In addition to the standard assumptions of stochastic approximation theory, the two time-scale analysis requires that ρ is updated at a much slower time scale than the value function.

The previous convergence results are based on the limit of infinite exploration. One of the many challenges in reinforcement learning is that of efficient exploration of the MDP to learn the dynamics and the rewards. There are model-based algorithms that guarantee learning an approximately optimal average-reward policy in time polynomial in the numbers of states and actions of the MDP and its mixing time. These algorithms work by alternating between learning the action models of the MDP by taking actions in the environment, and solving the learned MDP using offline value iteration.

In the "Explicit Explore and Exploit" or E^3 algorithm, the agent explicitly decides between exploiting the known part of the MDP and optimally trying to reach the unknown part of the MDP (exploration) (Kearns & Singh, 2002). During exploration, it uses the idea of "balanced wandering," where the least executed action in the current state is preferred until all actions are executed a certain number of times. In contrast, the R-Max algorithm implicitly chooses between exploration and exploitation by using the principle of "*optimism under uncertainty*" (Brafman & Tennenholtz, 2002). The idea here is to initialize the model parameters optimistically so that all unexplored actions in all states are assumed to reach a fictitious state that yields maximum possible reward from then on regardless of which action is taken. The optimistic initialization of the model parameters automatically encourages the agent to execute unexplored actions, until the true models and values of more states and actions are gradually revealed to the agent. It has been shown that with a probability at least $1 - \delta$, both E^3 and R-MAX learn approximately correct models whose optimal policies have an average reward ϵ-close to the true optimal in time polynomial in the numbers of states and actions, the mixing time of the MDP, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$.

Unfortunately the convergence results do not apply when there is function approximation involved. In the presence of linear function approximation, the average-reward version of temporal difference learning, which learns a state-based value function for a fixed policy, is shown to converge in the limit (Tsitsiklis & Van Roy, 1999). The transient behavior of this algorithm is similar to that of the corresponding discounted TD-learning with an appropriately scaled constant basis function (Van Roy & Tsitsiklis, 2002). As in the discounted case, development of provably convergent optimal policy learning algorithms with function approximation is a challenging open problem.

Cross References

►Efficient Exploration in Reinforcement Learning
►Hierarchical Reinforcement Learning
►Model-Based Reinforcement Learning

Recommended Reading

Abounadi, J., Bertsekas, D. P., & Borkar, V. (2002). Stochastic approximation for non-expansive maps: Application to Q-learning algorithms. *SIAM Journal of Control and Optimization, 41*(1), 1–22.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 72*(1), 81–138.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control*. Belmont, MA: Athena Scientific.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research, 2*, 213–231.

Crites, R. H., & Barto, A. G. (1998). Elevator group control using multiple reinforcement agents. *Machine Learning, 33*(2/3), 235–262.

Ghavamzadeh, M., & Mahadevan, S. (2006). Hierarchical average reward reinforcement learning. *Journal of Machine Learning Research, 13*(2), 197–229.

Kearns, M., & Singh S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning, 49*(2/3), 209–232.

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning, 22*(1/2/3), 159–195.

Marbach, P., Mihatsch, O., & Tsitsiklis, J. N. (2000). Call admission control and routing in integrated service networks using neuro-dynamic programming. *IEEE Journal on Selected Areas in Communications, 18*(2), 197–208.

Proper, S., & Tadepalli, P. (2006). Scaling model-based average-reward reinforcement learning for product delivery. In *European conference on machine learning* (pp. 725–742). Springer.

Puterman, M. L. (1994). *Markov decision processes: Discrete dynamic stochastic programming*. New York: Wiley.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the tenth international conference on machine learning* (pp. 298–305). San Mateo, CA: Morgan Kaufmann.

Seri, S., & Tadepalli, P. (2002). Model-based hierarchical average-reward reinforcement learning. In *Proceedings of international machine learning conference* (pp. 562–569). Sydney, Australia: Morgan Kaufmann.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Tadepalli, P., & Ok, D. (1998). Model-based average-reward reinforcement learning. *Artificial Intelligence, 100*, 177–224.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8*(3–4), 257–277.

Tsitsiklis, J., & Van Roy, B. (1999). Average cost temporal-difference learning. *Automatica, 35*(11), 1799–1808.

Van Roy, B., & Tsitsiklis, J. (2002). On average versus discounted temporal-difference learning. *Machine Learning, 49*(2/3), 179–191.

Wang, G., & Mahadevan, S. (1999). Hierarchical optimization of policy-coupled semi-Markov decision processes. In *Proceedings of the 16th international conference on machine learning* (pp. 464–473). Bled, Slovenia.

B

Backprop

►Backpropagation

Backpropagation

Paul Munro
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

Backprop; BP; Generalized delta rule

Definition

Backpropagation of error (henceforth *BP*) is a method for training feed-forward neural networks see ►Artificial Neural Networks. A specific implementation of BP is an iterative procedure that adjusts network weight parameters according to the gradient of an error measure. The procedure is implemented by computing an error value for each output unit, and by *backpropagating* the error values through the network.

Characteristics

Feed-Forward Networks

A feed-forward neural network is a mathematical function that is composed of constituent "semi-linear" functions constrained by a feed-forward network architecture, wherein the constituent functions correspond to nodes (often called *units* or *artificial neurons*) in a graph, as in Fig. 1. A feedfoward network architecture has a connectivity structure that is an *acyclic graph*; that is, there are no closed loops.

In most cases, the unit functions have a finite range such as $[0,1]$. Thus, the network maps $\mathbb{R}^N$ to $[0,1]^M$, where N is the number of input values and M is the number of output units. Let $FanIn(k)$ refer to the set of units that provide input to unit k, and let $FanOut(k)$ denote the set of units that receive input from unit k.

In an acyclic graph, at least one unit has a FanIn that is the null set. These are the *input units*; the activity of an input unit is not computed; rather it is set to a value external to the network (i.e., from the training data). Similarly, at least one unit has a null FanOut set. Such units typically represent the output of the network; i.e., this set of values is the result of the network computation. Intermediate units (often called *hidden units*) receive input from other units and project outputs to other computational units.

For the BP procedure, the activity of each unit is computed in two steps:

Linear step: the activities of the FanIn are each multiplied by an independent "weight" parameter, to which a "bias" parameter is added; each computational unit has a single bias parameter, independent of the other units. Let this sum be denoted x_k for unit k.

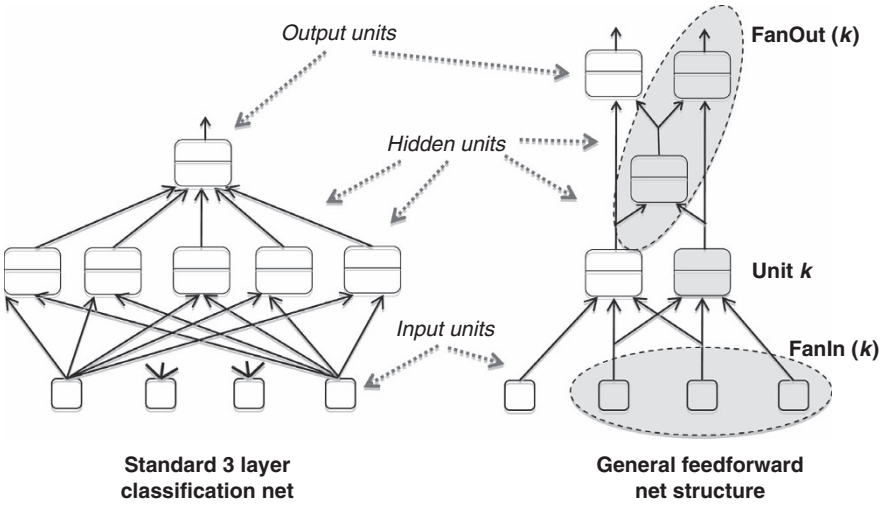
Nonlinear step: The activity a_k of unit k is a differentiable nonlinear function of x_k. A favorite function is the logistic $a = 1/(1 + \exp(-x))$, because it maps the range $[-\infty, +\infty]$ to $[0,1]$ and its derivative has properties conducive to the implementation of BP.

$$a_k = f_k(x_k); \quad \text{where } x_k = b_k + \sum_{j \in \text{FanIn}(k)} w_{kj} s_j$$

Gradient Descent

Derivation of BP is a direct application of the *gradient descent* approach to optimization and is dependent on a definition of network error, a function of the actual network response to a stimulus, $\mathbf{r}(\mathbf{s})$ and the target $\mathbf{T}(\mathbf{s})$. The two most common error functions are the summed squared error (SSE) and the cross entropy error (CE) (CE error as defined here is based on the presumption that the output values are in the range [0, 1]. Likewise

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,



Backpropagation. Figure 1. Two networks are shown. Input units are shown as simple squares at the bottom of each figure. Other units are computational (designated by a horizontal line). Left: A standard 3-layer network. Four input units project to five hidden units, which in turn project to a single output unit. Not all connections are shown. Such a network is commonly used for classification tasks. Right: An example of a feed-forward network with four inputs, three hidden units, and two outputs

for the target values; this is often used for classification tasks, wherein target values are set to the endpoints of the range, 0 and 1).

$$E^{\mathrm{SSE}} \equiv \sum_{\substack{i \in \mathrm{Outut} \\ \mathbf{s} \in \mathrm{Train}}} \left(T_i(\mathbf{s}) - r_i(\mathbf{s})\right)^2$$

$$E^{\mathrm{CE}} \equiv \sum_{\substack{i \in \mathrm{Outut} \\ \mathbf{s} \in \mathrm{Train}}} \left[T_i(\mathbf{s}) \ln\left(r_i(\mathbf{s})\right) - \left(1 - T_i(\mathbf{s})\right) \ln\left(1 - r_i(\mathbf{s})\right)\right]$$

Each weight parameter, w_{ij} (the weight of the connection from j to i), is updated by an amount proportional to the negative gradient of the error measure with respect to that parameter:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}},$$

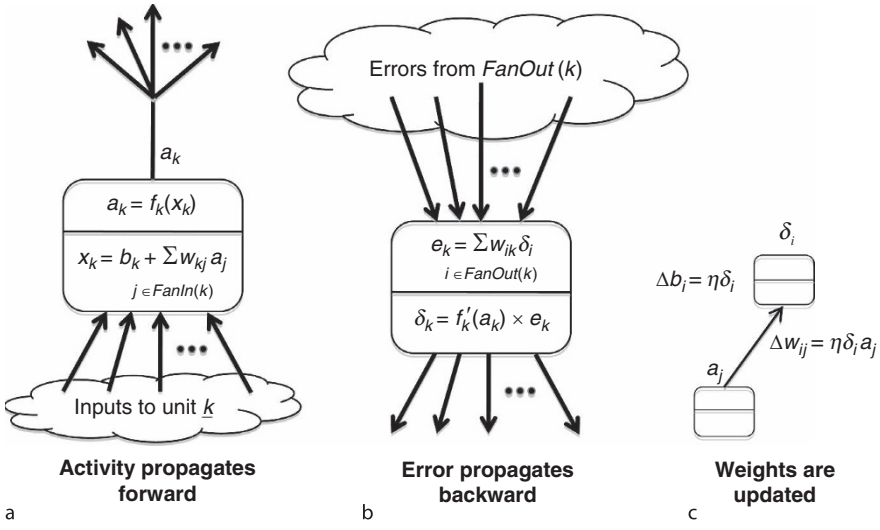
where the *step size*, η, modulates the intrinsic tradeoff between smooth convergence of the weights and the speed of convergence; in the regime where η is small, the system is well-behaved and converges smoothly, but slowly, and for larger η, the system may learn some subsets of the training set faster at the expense of smooth convergence on all patterns in the set. Thus, η is also called the *learning rate*.

Implementation

Several aspects of the feed-forward network must be defined prior to running a BP program, such as the configuration of the hidden units, the initial values of the weights, the functions they will compute, and the numerical representation of the input and target data. There are also parameters of the learning algorithm that must be chosen, such as the value of η and the form of the error function.

The weight and bias parameters are set to their initial values (these are usually random within specified limits). BP is implemented as an iterative process as follows:

1. A stimulus-target pair is drawn from the training set.
2. The activity values for the units in the network are computed for all the units in the network in a forward fashion from input to output (Fig. 2a).
3. The network output values are compared to the target and a *delta* (δ) value is computed for each output unit based on the difference between the target and the actual output response value.



Backpropagation. Figure 2. With each iteration of the backprop algorithm, (a) An activity value is computed for every unit in the network from the input to the output. (b) The network output is compared with the target. The error e_k for output unit k is defined as $(T_k - r_k)$. A value δ_k is computed for each output unit by multiplying e_k by the derivative of the activity function. For hidden units, the error is propagated backward using the weights. (c) The weight parameters w_{ij} are updated in proportion to the product of δ_i and a_j

4. The deltas are propagated backward through the network using the same weights that were used to compute the activity values (Fig. 2b).
5. Each weight is updated by an amount proportional to the product of the downstream delta value and the upstream activity (Fig. 2c).

The procedure can be run either in an *online* mode or *batch* mode. In the online mode, the network parameters are updated for each stimulus-target pair. In the batch mode, the weight changes are computed and accumulated over several iterations without updating the weights until a large number (B) of stimulus-target pairs have been processed (often, the entire training set), at which the weights are updated by the accumulated amounts.

$$\text{online}: \quad \Delta w_{ij}(t) = \eta \delta_i(t) a_j(t) \quad \Delta b_i(t) = \eta \delta_i(t)$$

$$\text{batch}: \quad \Delta w_{ij}(t+B) = \sum_{s=t-1}^{t+B} \eta \delta_i(s) a_j(s)$$

$$\Delta b_i(t+T) = \sum_{s=t+1}^{t-B} \eta \delta_i(s)$$

Classification Tasks with BP

The simplest and most common classification function returns a binary value, indicating membership in a particular class. The most common network architecture for a task of this kind is the three-layer network of Fig. 1 (left), with training values of 0 and 1. For classification tasks, the cross entropy error function generally gives significantly faster convergence. After training, the network is in test mode or production mode, and the responses are in the continuous range [0, 1]; the response must thus be interpreted. The value of the response could be interpreted as a probability or fuzzy Boolean value. Often, however, a single threshold is applied to give a binary answer. A double threshold is sometimes used, with the midrange defined as "uncertain."

Curve Fitting with BP

A feed-forward network can be trained to approximate any function, given the sufficient hidden units. The range of the output unit(s) must be capable of generating activity values in the required range. In order to accommodate an arbitrary range uniformly, a linear

function is advisable for the output units, and the SSE function is the basis for gradient descent.

The Autoencoder Architecture

The autoencoder is a network design in which the target pattern is identical to the input pattern. The hidden units are configured such that there is a "bottleneck layer" of units that is smaller than the input layer, through which information flows; i.e., there are no connections bypassing the bottleneck. Thus, any information necessary to reconstruct the input pattern at the output layer must be represented at the bottleneck. This approach has been successfully applied as an approach to *nonlinear dimensionality reduction* (e.g., Demers & Cottrell, 1993). It bears notable similarities and differences to linear techniques, such as ►*principal components analysis* (*PCA*).

Prediction with BP

The plain "vanilla" BP propagates input to output with no explicit representation of time. Several approaches to processing of temporal patterns have been put forward. Most prominent among these are:

Time delay neural network. In this approach, the input stimulus is simply a sample of a time varying signal. The input patterns are typically generated by a sliding window of samples over time or over a sequence.

►Simple recurrent network (Elman, 1990). A sequence of stimulus patterns is presented as input for the network, which has a single hidden layer design. With each iteration, the input is augmented by a secondary set of input units whose activity is a copy of the hidden layer activity from the previous iteration. Thus, the network is able to maintain a representation of the recent history of network stimuli.

Backpropagation through time (Rumelhart, Hinton, & Williams, 1986). A recurrent network (i.e., a cyclic network) is "unfolded in time" by forming a large multilayer network, in which each layer is a copy of the entire network shifted in time. Thus, the number of layers limits the temporal window available to the network.

Recurrent backpropagation (Pineda, 1989). An acyclic network is run with activity propagation and error propagation, until variables converge. Then the weights are updated.

Cognitive Modeling with BP

Interest in BP as a training technique for classifiers has waned somewhat since the introduction of ►*Support vector machines* (SVMs) in the mid 1990s. However, the influence of BP as an approach to modeling cognitive processes, including perception, concept learning, spatial cognition, and language learning, remains strong. Analysis of hidden unit representations (e.g., using clustering techniques) has given insight into plausible intermediate processes that may underlie cognitive phenomena. Also, many cognitive models trained with BP have exhibited time courses consistent with stages of human learning.

Biological Inspiration and Plausibility

The "connectionist" approach to modeling cognition is based on "neural network" models, which have been touted as "biologically inspired" since their inception. The similarities and differences between connectionist architectures and living brains have been exhaustively debated. Like the brain, the models consist of elements that are extremely limited, computationally. Computational power is derived by several units in network architecture. However, there are compelling differences as well. For example, the temporal dynamics in biological neurons is far more complex than the simple functions used in connectionist networks. It remains unclear what level of neurobiological detail is relevant to understand the cognitive functions.

Shortcomings of BP

The BP method is notorious for convergence problems. An inherent problem of gradient descent approaches to optimization is the issue of locally optimal values. Seeking a minimum value be heading downhill is like water running downhill. Not all water reaches the lowest point (sea level). Water that flows into a mountain lake has landed in a local minimum, a region that is bounded by higher ground.

Even when BP converges to a global minimum (or a local minimum that is "good enough"), it is sometimes very slow. The convergence properties of BP depend on the learning rate and random factors, such as the initial weight and bias values.

Another difficulty with BP is the selection of a network structure. The number of hidden units and the

interconnectivity among them has a strong influence on both the generalization performance and the convergence time. Since the nature of this influence is poorly understood, the design of the network is left to guesswork. The standard approach is to use a single hidden layer (as in Fig. 1, left), which has the advantage of relatively fast convergence.

History

The idea of training a multilayered network using error propagation was originated by Frank Rosenblatt (1958, 1962). However, he was unable to apply gradient descent because he was using linear threshold functions that were not differentiable; therefore, the technique of gradient descent was unavailable. He developed a technique known as the perceptron learning rule that is only applicable to two layer networks (no hidden units). Without hidden units, the computational power of the network is severely reduced. Work in the field virtually stopped with the publication of *Perceptrons* (Minsky & Papert, 1969). The backpropagation procedure was first published by Werbos (1974), but did not receive significant recognition until it was put forward by Rumelhart et al. (1986).

Cross References

►Artificial Neural Networks

Recommended Reading

Demers, D., & Cottrell, G. (1993). Non-linear dimensionality reduction. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Advances in neural information processing systems* (Vol. 5). San Mateo, CA: Morgan Kaufmann.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Minsky, M. L., & Papert, S. A. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

Pineda, F. J. (1989). Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Computation*, *1*, 161–172.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*, 386–408.

Rosenblatt, F. (1962). *Principles of statistical neurodynamics*. Washington, DC: Spartan.

Werbos, P. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. Ph.D. thesis, Harvard University, Cambridge.

Bagging

Bagging is an ►ensemble learning technique. The name "Bagging" is an acronym derived from *B*ootstrap *AGG*regat*ING*. Each member of the ensemble is constructed from a different training dataset. Each dataset is a ►bootstrap sample from the original. The models are combined by a uniform average or vote. Bagging works best with ►unstable learners, that is those that produce differing generalization patterns with small changes to the training data. Bagging therefore tends not to work well with linear models. See ►ensemble learning for more details.

Bake-Off

Definition

Bake-off is a disparaging term for experimental evaluation of multiple learning algorithms by a process of applying each algorithm to a limited set of benchmark problems.

Cross References

►Algorithm Evaluation

Bandit Problem with Side Information

►Associative Reinforcement Learning

Bandit Problem with Side Observations

►Associative Reinforcement Learning

Basic Lemma

►Symmetrization Lemma

Basket Analysis

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Synonyms
Market basket analysis

Definition
The goal of basket analysis is to utilize large volumes of electronic receipts, stored at the checkout terminals of supermarkets, for better understanding of customer behavior.

While many forms of learning and mining can be applied to market baskets, the term usually refers to some variant of ►association rule mining. In the basic setting, each market basket constitutes an example essentially defined by the set of purchased products. Association rules then identify sets of items that tend to be bought together. A classical, anecdotal discovery from supermarket data is that "if a basket contains diapers then it often also contains beer." This example illustrates several potential benefits of market basket analysis by association rules: simplicity and understandability of the results, actionability of the results, and a form of nonsupervised approach where the consequent of the rule has not been fixed by the user.

Association rules are often found with the ►Apriori algorithm, and are based on ►frequent itemsets.

Cross References
►Apriori Algorithm
►Association Rule
►Frequent Itemset
►Frequent Pattern

Batch Learning

Synonyms
Offline Learning

Definition
A *batch learning algorithm* accepts a single input that is a set or sequence of observations. The algorithm produces its ►model, and does no further learning. Batch learning stands in contrast to ►online learning.

Baum–Welch Algorithm

The Baum–Welch algorithm is used for computing maximum likelihood estimates and posterior mode estimates for the parameters (transition and emission probabilities) of a HMM, when given only output sequences (emissions) as training data.

The Baum–Welch algorithm is a particular instantiation of the expectation-maximization algorithm, suited for HMMs.

Bayes Adaptive Markov Decision Processes

►Bayesian Reinforcement Learning

Bayes Net

►Bayesian Network

Bayes Rule

Geoffrey I. Webb
Monash University

Definition
Bayes rule provides a decomposition of a conditional probability that is frequently used in a family of learning techniques collectively called *Bayesian Learning*. Bayes rule is the equality

$$P(z \mid w) = \frac{P(z)P(w \mid z)}{P(w)} \quad (1)$$

$P(w)$ is called the *prior probability*, $P(w \mid z)$ is called the *posterior probability*, and $P(z \mid w)$ is called the *likelihood*.

Discussion
Bayes rule is used for two purposes. The first is *Bayesian update*. In this context, z represents some new information that has become available since an estimate $P(w)$

was formed of some hypothesis w. The application of Bayes' rule enables a new estimate of the probability of w (the posterior probability) to be calculated from estimates of the prior probability, the likelihood and $\mathrm{P}(z)$.

The second common application of Bayes' rule is for estimating posterior probabilities in probabilistic learning, where it is the core of ►Bayesian networks, ►naïve Bayes, and ►semi-naïve Bayesian techniques.

While Bayes' rule may initially appear mysterious, it is readily derived from the basic principle of conditional probability that

$$\mathrm{P}(w\,|\,z) = \mathrm{P}(w,z)\mathrm{P}(z) \tag{2}$$

As

$$\mathrm{P}(w,z) = \frac{\mathrm{P}(w)\mathrm{P}(w,z)}{\mathrm{P}(w)} \tag{3}$$

and

$$\frac{\mathrm{P}(w,z)}{\mathrm{P}(w)} = \mathrm{P}(z\,|\,w), \tag{4}$$

Bayes' rule (Eq. 1) follows by simple substitution of Eq. (4) into Eq. (3) and then of the result into Eq. (2).

Cross References

►Bayesian Methods
►Bayesian Network
►Naïve Bayes
►Semi-Naïve Bayesian Learning

Bayesian Methods

Wray Buntine
NICTA, Canberra, Australia

Definition

The two most important concepts used in Bayesian modeling are *probability* and *utility*. Probabilities are used to model our belief about the state of the world and utilities are used to model the *value* to us of different outcomes, thus to model costs and benefits. Probabilities are represented in the form of $p(x|C)$, where C is the current known context and x is some event(s) of interest from a space χ. The left and right arguments of the probability function are in general propositions (in the logical sense). Probabilities are updated based on new evidence or outcomes y using *Bayes rule*, which takes the form

$$p(x|C,y) = \frac{p(x|C)p(y|x,C)}{p(y|C)},$$

where χ is the discrete domain of x. More generally, any measurable set can be used for the domain χ. An integral or mixed sum and integral can replace the sum. For a utility function $u(x)$ of some event x, for instance the benefit of a particular outcome, the *expected value* of $u()$ is

$$\mathcal{E}_{x|C}[u(x)] = \sum_{x\in\mathcal{X}} p(x|C)u(x).$$

One then estimates the expected utility $\mathcal{E}_{x|C,y}[u(x)]$ based on different evidence, actions or outcomes y. An action is taken to maximize this expected utility, appealing to the principle of *maximum expected utility (MEU)*. A common application of this principle is recursive: one should take the action now that will maximize utility in the future, assuming all future actions are also taken to maximize utility.

Motivation and Background

In modeling a problem, primarily, one considers an interrelated space of *events* or *states, actions*, and *outcomes*. Events describe the state of the world, outcomes are also sometimes considered events but they are special in that one directly obtains from them costs or benefits. Actions allow one to influence the world. Some actions may instigate tests and thus also help measure the state of the world to reduce uncertainty. Some problems may be dynamic in that a sequence of actions and outcomes are considered and the resulting changes in states modeled.

The Bayesian approach is a modeling methodology that provides a principled approach of how to reason and act in the context of uncertainty and a dynamic environment. In the approach, probabilities are used to model all forms of belief or proportions about events and states, and then utilities are used to model the costs and benefits of any actions taken. An explicit assumption is that these probabilities and utilities can be adequately elicited and precisely modeled for the problem. An implicit assumption is that the computation required – recursive evaluation of

possibly nested integrals and sums (over domain variables) – can be done quickly enough so that the computation itself does not become a significant factor in the costs considered.

The Bayesian approach is named after Rev. Thomas Bayes, whose work was contributed to the Royal Society in 1763 after his death, although it was independently more generally presented as a theory by Laplace in 1812. The field was subsequently developed into a field of statistics, inference and decision theory by a stream of authors in the 1900s including Jeffreys (Bernardo and Smith, 1994). The field of statistics was dominated by the frequentist school during the 1990s, and for a time Bayesian methods were considered controversial. Like the different schools of theory in machine learning, these statistical approaches now coexist.

The Bayesian approach can be justified by axiomatic prescriptions of how a rational agent should reason and act, and by appeal to principles of consistency. In the context of learning, probabilities are used to infer models of the problem of interest, and then utilities are used to recommend predictions or analysis based on the models.

Theory

Basic Theory

First, consider definitions, the different kinds of probability, the process of reasoning (about probabilities), and making decisions.

Basic definitions: Probabilities are represented in the form of $p(x|C)$, where C is the current known context and x is some event(s) of interest. It is sufficient to place in C only terms relevant to x and ignore terms assumed by default. Moreover, both x and C must have well-defined events. For instance, x = "John is tall" is not considered a well-defined event since the word "tall" is not precise. One would instead replace it with something like x = "John is greater than 6 foot tall" or x = "Julie said John is tall."

An important functional used with probabilities is the *expected value*. For a function $f(x)$ of some event x from a space χ, the expected value of $f()$ is $\mathcal{E}_{x\in\chi}[f(x)]$.

Utility is used to measure value or relative satisfaction, and is usually represented as a function on outcomes. Costs are negative utility and benefits are positive. Utilities should be additive in worth, and are often practically interpreted in monetary units. Strictly speaking, the value of money is nonlinear (for most people, 2 billion dollars is not significantly better than 1 billion dollars), so it is not a correct utility measure. However, it is adequate when the range of financial transactions expected is reasonable.

Expected utility, which is the expected value of the utility function, is the fundamental quantity assessed with Bayesian methods. Some scenarios are the following:

Prediction: For prediction problems, the outcome is the "true" value, and the utility is sometimes the mean square error or the absolute error. In data mining, the choices are much richer, see ▶Model Evaluation.

Diagnosis: The outcome is the "true" diagnosis, and utility is made up of the differing costs of treatment, mistreatment, and delay or nontreatment, as well as any benefit from correct diagnosis.

Game playing: The utility comes from the eventual outcome of the game, each player has their own utility and the state of the game constantly changes as plays are made.

In Bayesian machine learning, we usually take utilities as a given, and the majority of the work revolves around evaluating and estimating probabilities and maximizing of expected utility. In some ranking tasks and generalized agent learning, the utilities themselves may be poorly understood.

Belief and proportions: Some probabilities correspond to *proportions* that exist in the real world, such as the proportion of school children in the general population of a given state. These real proportions can be measured by counting or sampling, and they are governed by Kolmogorov's Axioms for probability, including the probability of certainty is 1 and the probability of a disjunction of mutually exclusive events is the sum of the probabilities of the individual events. This kind of probability is used in the *Frequentist School* that only considers long term average proportions obtained from a series of independent and identical experiments. These proportions can be model parameters one wishes to reason about.

Probabilities can also represent beliefs. For instance, in 2000, one could have had a belief about the event that

George Bush would win the 2001 Presidential Election in the USA. This event is unique and has only one outcome, so the frequentist notion cannot be justified, i.e., there is no long-term sequence of different 2001 presidential elections with George Bush. Beliefs are usually considered to be *subjective*, in that they are specific to each agent, reflecting their sum of unique experiences, and the unique context in which the event in question occurs.

To better understand the role beliefs play in Bayesian methods, also see ▶Prior Probabilities.

Reasoning: A stylized version of probabilistic reasoning considers an event of interest one is reasoning about, x, and evidence, y, one may obtain. Typical scenarios are

Learning: $x = (\boldsymbol{\Theta}, M)$ are parameters $\boldsymbol{\Theta}$ of a model from family M, and $y = \boldsymbol{D}$ is a set of data $\boldsymbol{D} = \{d_1, \ldots, d_N\}$. So one considers $p(\boldsymbol{\Theta}, M|\boldsymbol{D}, C)$ versus $p(\boldsymbol{\Theta}, M|C)$.

Diagnosis: x a disease or condition, and y is a set of observable symptoms or diagnostic tests. One might choose a test y that maximizes the expected utility.

Hypothesis testing: x is a hypothesis H and y is some sequence of evidence $E_1, E_2, \ldots, E_n$, so we consider $p(H|E_1, E_2, \ldots, E_n)$ and hope it is sufficiently high.

Different probabilities are then considered:

$p(x|C)$: *The prior probability* for event x, called the base-rate in some contexts.

$p(y|C)$: The *prior probability* for evidence y. Once the evidence has been seen, this is also used as a proxy for the quality of the model.

$p(x|y, C)$: The *posterior probability* for event x given evidence y.

$p(y|x, C)$: The *likelihood* for the event x based on evidence y.

In the case of diagnostic reasoning, the prior $p(x|C)$ is usually the base rate for the disease or condition, and can be got from the population base rate.

In the case of learning, however, the prior $p(\boldsymbol{\Theta}, M|C)$ represents a prior distribution on parameters about which we may well be largely ignorant, or at least may not be able to readily elicit from experts. For instance, the proportion θ_D might be the probability of a new drug slowing the onset of AIDS related diseases. At the moment of initial testing, θ_D is unknown so one places a probability distribution over θ_D, which represents one's belief about the proportion.

These priors are second-order probabilities, beliefs about proportions, and they are the most challenging quantity modeled with the Bayesian approach. They can be a function on thousands of parameters, and can be critical in the success of applications. They are also challenging from the philosophical perspective.

Decision theory: The term *Bayesian inference* is usually reserved for the process of manipulating priors and posteriors, computing probabilities, and computing expected values. *Bayesian decision theory* describes the process of formulating utilities and then evaluating the (sometimes) recursive maximum expected utility formula, such as in game playing, or interactive advertising.

In Bayesian theory one takes the action that maximizes expected utility (MEU) in the current context, sometimes referred to as the *expected utility hypothesis*. Decision theory places this in a dynamic context and says each action should be taken to maximize expected future utility. This is defined recursively, so taken to the limit this implies the optimal future actions need to be determined before the optimal current action can be got via MEU.

Justifications

This section covers basic mathematical justifications of the theory. The best general reference for this is Bernardo and Smith (1994). Additional discussion of prior probabilities appears in ▶Prior Probabilities.

Note that Bayesian theory, with its acceptance as a branch of mainstream statistics, is widely accepted for the following reasons:

Application: It has extensive support through practical success, often times by clever combination of prior knowledge and statistical and computational finesse.

Explanation: It provides a convenient common language in which a variety of other theoretical approaches can be represented. For instance PAC, MDL methods, penalized likelihood methods, and the maximum margin approach all find good interpretations within the Bayesian framework.

Composition: It allows different reasoning tasks to be composed in a coherent way. With a probabilistic framework, the components can interoperate in a coherent manner, so that information may flow bidirectionally between components via probabilities.

Composition of processing steps in intelligent systems is a key application for Bayesian methods. For instance, natural language and vision recognition tasks can sometimes be broken down into a processing chain (for instance, doing a named entity recognition step before a dependency parsing step), but these components rarely work conclusively and unambiguously. By attaching probabilities to the output of components, and allowing probabilistic inputs, the uncertainty inherent in individual steps can be propagated and managed.

Theoretical justifications also exist to support each of the different components, probabilities, and utilities. These justifications are based on the concept of *normative* axioms, axioms that do not describe reasoning but rather prescribe basic principles it should follow. The axioms try to capture principles such as coherence and consistency in a quantitative manner. These various justifications have their reported shortcomings and a rich literature exists arguing about the details and postulating new variants. These axiomatic justifications are supportive of the Bayesian approach, but they are not irrefutable.

Justifying probabilities: In the Bayesian approach, beliefs and proportions are given the same mathematical treatment.

One set of arguably controversial justifications for this revolve around betting (Bernardo and Smith, 1994, Sect. 2.8.3). Someone's subjective beliefs about specific events, such as significant economic and political events (or horse races), are claimed to be measurable by offering them a series of options or bets. Moreover, if their beliefs do not behave like proportions, then a clever bookmaker can use a so-called Dutch book to consistently profit from them.

An alternative scheme for justifying probability by Cox is based on normative axioms that beliefs should follow. For instance, one controversial axiom by Cox is that belief about a single event should be represented by a single real number. These axioms are presented by Jaynes as rules for a robot (Jaynes, 2003), and as rules for intelligent systems by Horvitz et al. (1986).

Justifying decision theory: Another scheme again using normative axioms, by von Neumann and Morgenstern, is used to justify the use of utilities. This scheme assumes probabilities are the basis of inference about uncertainty. A different set of normative axiomatic schemes have been developed that justify the use of probabilities and utilities together under MEU, the best known is by Savage but others exist (Bernardo and Smith, 1994).

Bayesian Computation

The first part of this article has been devoted to a brief overview of the Bayesian approach. Computation for Bayesian inference is an extensive field itself. Here we review the basic aspects as a pointer to the literature. This is an active area of research in machine learning, statistics, and a many applied artificial intelligence communities such as natural language processing, image analysis, and others.

In general, in Bayesian reasoning one wants to estimate posterior average parameter values, or their average variance, or some other averaged quantity, then general formulas are given by (in the case of continuous parameters)

$$\overline{\boldsymbol{\Theta}} = \mathcal{E}_{\boldsymbol{\Theta}|\boldsymbol{D},M,C}[\boldsymbol{\Theta}] = \int_{\boldsymbol{\Theta}} \boldsymbol{\Theta} p(\boldsymbol{\Theta}|\boldsymbol{D},M,C)\mathrm{d}\boldsymbol{\Theta}$$
$$\mathrm{var}(\boldsymbol{\Theta}) = \mathcal{E}_{\boldsymbol{\Theta}|\boldsymbol{D},M,C}\left[(\boldsymbol{\Theta}-\overline{\boldsymbol{\Theta}})^2\right]$$

Marginal likelihood: A useful quantity to assist in evaluating results, and a worthy score in its own right is the marginal likelihood, in the continuous parameter case found from the likelihood $p(\boldsymbol{D}|\boldsymbol{\Theta},M,C)$ by taking an average

$$p(\boldsymbol{D}|M,C) = \int_{\boldsymbol{\Theta}} p(\boldsymbol{\Theta}|M,C)p(\boldsymbol{D}|\boldsymbol{\Theta},M,C)\mathrm{d}\boldsymbol{\Theta}.$$

This is also called the *normalizing constant* due to its occurrence in the posterior formula

$$p(\boldsymbol{\Theta}|\boldsymbol{D},M,C) = \frac{p(\boldsymbol{\Theta}|M,C)p(\boldsymbol{D}|\boldsymbol{\Theta},M,C)}{p(\boldsymbol{D}|M,C)}!.$$

It is generally difficult to estimate because of the multidimensional integrals and sums.

Exponential family distributions: Standard probability distributions covered in mathematical statistics, such as the ▶Gaussian Distribution, the Poisson, Dirichlet, Gamma, and Wishart, have very convenient mathematical properties that make Bayesian estimation easier. With these distributions, one computes statistics, called *sufficient statistics*, such as a mean and sum of squares (for the Gaussian), and then parameter estimation follows with a function inverse on a concave function. This is the basis of ▶linear regression, ▶principal components analysis, and some ▶decision tree learning methods, for instance. All good texts on mathematical statistics cover these in detail. Note the marginal likelihood is often computable in closed form for exponential family distributions.

Graphical models: ▶Graphical Models are a general family of of probabilistic models formed by composing graphs over variables. They work particularly well with exponential family distributions, and allow a rich variety of popular machine learning and data mining methods to be represented and manipulated. Graphical models allow complex models to be composed from simpler components and provide a family of algorithm schemes for developing inference and learning methods that operate on them. They have become the *de facto* standard for presenting (suitable decomposed) models and algorithms in the machine learning community.

Maximum a posterior estimation: known as MAP, is usually the simplest form of parameter estimation that could be called Bayesian. It also corresponds to a penalized or regularized maximum likelihood method. Given the posterior for a stylized learning problem of the previous section, one finds the parameters $\boldsymbol{\Theta}$ that maximizes the posterior $p(\boldsymbol{\Theta}, M|\boldsymbol{D}, C)$, which can be conveniently done without computing the marginal likelihood above, so

$$\widehat{\boldsymbol{\Theta}_{MP}} = \underset{\boldsymbol{\Theta}}{\operatorname{argmax}} \log p(\boldsymbol{\Theta}, \boldsymbol{D}|M, C),$$

where the log probability can be broken down as a prior and a likelihood term

$$\log p(\boldsymbol{\Theta}, \boldsymbol{D}|M, C) = \log p(\boldsymbol{\Theta}|M, C) + \log p(\boldsymbol{D}|\boldsymbol{\Theta}, M, C).$$

The Laplace approximation: When the posterior is well behaved, and there is a large amount of data, the posterior is focused around a vanishing small region in parameter space of diameter $O(1/\sqrt{(}N))$. If this occurs away from the boundary of the parameter space, then one can make a second-order Taylor expansion of the log. posterior at the MAP point and the result is a Gaussian approximation to the posterior.

$$\log p(\boldsymbol{D}, \boldsymbol{\Theta}|M, C) \approx \log p(\boldsymbol{D}, \widehat{\boldsymbol{\Theta}_{MP}}|M, C) + \frac{1}{2}\left(\widehat{\boldsymbol{\Theta}_{MP}} - \boldsymbol{\Theta}\right)^T \left.\frac{\mathrm{d}^2 \log p(\boldsymbol{D}, \boldsymbol{\Theta}|M, C)}{\mathrm{d}\boldsymbol{\Theta}\mathrm{d}\boldsymbol{\Theta}^T}\right|_{\boldsymbol{\Theta}=\widehat{\boldsymbol{\Theta}_{MP}}} \left(\widehat{\boldsymbol{\Theta}_{M,P}} - \boldsymbol{\Theta}\right).$$

From this, one can approximate integrals such as the marginal likelihood $p(\boldsymbol{D}|M, C)$. This is known as the *Laplace approximation*, the name of the corresponding general method used for the asymptotic expansion of integrals. In general, this is a poor approximation, but it serves to aid our understanding of parameter estimation (MacKay, 2003 Chaps. 27 and 28), and is the approximate basis for some model selection criteria.

Latent variable models: Latent variables are data that are hidden and thus never observed in the evidence. However, their existence is postulated as a significant component of the model. For instance, in ▶Clustering (an unsupervised method) and finite mixture models generally, one assumes each data point has a hidden class label, thus the Bayesian model of clustering is a simple kind of latent variable model.

▶*Markov chain Monte Carlo* methods: The most general form of reasoning and estimation available are the *Markov chain Monte Carlo* (MCMC) methods. The MCMC methods couple two processes: first, they use *Monte Carlo* or simulation methods to estimate the integral, and second they use a *Markov Chain* to sample, so sampling is sequentially (Markovian) based, and samples are not independent.

Simulation methods generally use the functional form of $p(\boldsymbol{\Theta}, \boldsymbol{D}|M, C)$ so we do not need to compute the marginal likelihood. Hence, given a set of I samples $\{\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_I\}$ the expected value is approximated with a weighted average

$$\overline{\boldsymbol{\Theta}} \approx \frac{1}{I}\sum_{i=1}^{I} w_i \boldsymbol{\Theta}_i.$$

The simplest case is where the samples are made independently according to the posterior itself and then the

weights $w_i = 1$, This is called the ordinary Monte Carlo (OMC) method, but it is not often usable in practice because efficient multidimensional posterior samplers rarely exist. Alternatively, one can sample according to a Markov Chain, $\boldsymbol{\Theta}_{i+1} \sim q(\boldsymbol{\Theta}_{i+1}|\boldsymbol{\Theta}_i)$, so each $\boldsymbol{\Theta}_{i+1}$ is conditionally dependent on $\boldsymbol{\Theta}_i$. So while samples are not independent, as long as the long run distribution of the Markov chain is the same as the posterior, the same approximation formula holds. There are a rich variety of MCMC methods, and this forms one of the key areas of current research.

Gibbs sampling: The simplest kind of MCMC method samples each dimension (or sub-vector) in turn. Suppose the parameter vector has K real components, $\boldsymbol{\Theta} = (\theta_1, \ldots, \theta_K)$. Sampling a complete $\boldsymbol{\Theta}$ in one go is not generally possible given just a functional form of the posterior $p(\boldsymbol{\Theta}|\boldsymbol{D}, M, C)$ but given no computable form for the normalizing constant. Gibbs sampling works in the one-dimensional case where normalizing bounds can be obtained and sampling tricks used. The conditional posterior of θ_k is given by

$$p(\theta_k|(\theta_1, \ldots, \theta_{k-1}, \theta_{k+1}, \ldots, \theta_K), \boldsymbol{D}, M, C),$$

and this is usually easier to sample from.

The Gibbs (and MCMC) sample $\boldsymbol{\Theta}_{i+1}$ can be drawn given the previous sample $\boldsymbol{\Theta}_i$ by progressively resampling each dimension in turn and so slowly updating the full vector:

1. Sample $\theta_{i+1,1}$ according to $p(\theta_1|\theta_{i,2}, \ldots, \theta_{i,K}, \mathbf{D}, M, C)$.
...
k. Sample $\theta_{i+1,k}$ according to $p(\theta_2|\theta_{i+1,1}, \ldots, \theta_{i+1,k-1}, \theta_{i,k+1}, \ldots, \theta_{i,K}, \mathbf{D}, M, C)$.
...
K. Sample $\theta_{i+1,k}$ according to $p(\theta_K|\theta_{i+1,1}, \ldots, \theta_{i+1,K-1}, \mathbf{D}, M, C)$.

In samping terms, this method is no more successful than coordinate-wise ascent is as a primitive greedy search method: it is supported by theoretical results but can be very slow to converge.

Variational approximations: When the function you seek to optimize or average over presents difficulty, perhaps it is highly multimodal, then one option is to change the function itself, and replace it with a more readily approximated function. Variational methods provide a general principle for doing this safely. The general principle uses variational calculus, which is the calculus over functions, not just variables. Variational methods are a very general approach that can be used to develop a broad range of algorithms (Wainwright and Jordan, 2008).

Nonparametric models: The above discussion implicitly assumed the model has a fixed finite parameter vector $\boldsymbol{\Theta}$. If one is attempting to model a regression function, or a language grammar, or image model of unknown *a priori* structural complexity, then one cannot know the dimension ahead of time. Moreover, as in the case of functions, the dimension cannot always be finite. The ▶Bayesian Nonparametric Models address this situation, and are perhaps the most important family of techniques for general machine learning.

Cross References

▶Bayes Rule
▶Bayesian Nonparametric Models
▶Markov Chain Monte Carlo
▶Prior Probability

Recommended Reading

A good introduction to the problems of uncertainty and philosophical issues behind the Bayesian treatment of probability is in Lindley (2006). From the statistical machine learning perspective, a good introductory text is by MacKay (2003) who carefully covers information theory, probability, and inference but not so much statistical machine learning. Another alternative introduction to probabilities is the posthumously completed and published work of Jaynes (2003).

Discussions from the frequentist versus Bayesian battlefront can be found in works such as (Rosenkrantz and Jaynes, 1983), and in the approximate artificial intelligence versus probabilistic battlefront in discussion articles such as Cheeseman's (1988) and the many responses and rebuttals. It should be noted that it is the continued success in applications that have really led these methods into the mainstream, not the entertaining polemics.

Good mathematical statistics text books, such as Casella and Berger (2001) cover the breadth of statistical methods and therefore handle basic Bayesian theory. A more comprehensive treatment is given in Bayesian texts such as Gelman et al. (2003).

Most advanced statistical machine learning text books cover Bayesian methods, but to fully understand the subtleties of prior beliefs and Bayesian methodology one needs to view more advanced Bayesian literature. A detailed theoretical reference for Bayesian methods is Bernardo and Smith (1994).

Bernardo, J., & Smith, A. (1994). *Bayesian theory*. Chichester: Wiley.
Casella, G., & Berger, R. (2001). *Statistical inference* (2nd ed.). Pacific Grove: Duxbury.

Cheeseman, P. (1988). An inquiry into computer understanding. *Computational Intelligence, 4*(1), 58–66.
Gelman, A., Carlin, J., Stern, H., & Rubin, D. (2003). *Bayesian data analysis* (2nd ed.). Boca Raton: Chapman & Hall/CRC Press.
Horvitz, E., Heckerman, D., & Langlotz, C. (1986). A framework for comparing alternative formalisms for plausible reasoning. *Fifth National Conference on Artificial Intelligence*, Philadelphia, pp. 210–214.
Jaynes, E. (2003). *Probability theory: the logic of science*. New York: Cambridge University Press.
Lindley, D. (2006). *Understanding uncertainty*. Hoboken: Wiley.
MacKay, D. (2003). *Information theory, inference, and learning algorithms*. Cambridge: Cambridge University Press.
Rosenkrantz, R. (Ed.). (1983). *E.T. Jaynes: papers on probability, statistics and statistical physics*. Dordrecht: D. Reidel.
Wainwright, M. J., & Jordan, M. I. (2008). *Graphical models, exponential families, and variational inference*. Hanover: Now Publishers.

Bayesian Model Averaging

►Learning Graphical Models

Bayesian Network

Synonyms

Bayes net

Definition

A Bayesian network is a form of directed ►graphical model for representing multivariate probability distributions.

The nodes of the network represent a set of random variables, and the directed arcs represent causal relationships between variables. The *Markov property* is usually required: every direct dependency between a possible cause and a possible effect has to be shown with an arc. Bayesian networks with the Markov property are called *I-maps* (independence maps). If all arcs in the network correspond to a direct dependence on the system being modeled, then the network is said to be a *D-map* (dependence-map). Each node is associated with a conditional probability distribution, that quantifies the effects the parents of the node, if any, have on it. Bayesian support various forms of reasoning: *diagnosis*, to derive causes from symptoms, *prediction*, to derive effects from causes, and *intercausal reasoning*, to discover the mutual causes of a common effect.

Cross References

►Graphical Models

Bayesian Nonparametric Models

Peter Orbanz[1], Yee Whye Teh[2]
[1]Cambridge University, Cambridge, UK
[2]University College London, London, UK

Synonyms

Bayesian methods; Dirichlet process; Gaussian processes; Prior probabilities

Definition

A Bayesian nonparametric model is a Bayesian model on an infinite-dimensional parameter space. The parameter space is typically chosen as the set of all possible solutions for a given learning problem. For example, in a regression problem, the parameter space can be the set of continuous functions, and in a density estimation problem, the space can consist of all densities. A Bayesian nonparametric model uses only a finite subset of the available parameter dimensions to explain a finite sample of observations, with the set of dimensions chosen depending on the sample such that the effective complexity of the model (as measured by the number of dimensions used) adapts to the data. Classical adaptive problems, such as nonparametric estimation and model selection, can thus be formulated as Bayesian inference problems. Popular examples of Bayesian nonparametric models include Gaussian process regression, in which the correlation structure is refined with growing sample size, and Dirichlet process mixture models for clustering, which adapt the number of clusters to the complexity of the data. Bayesian nonparametric models have recently been applied to a variety of machine learning problems, including regression, classification, clustering, latent variable modeling, sequential modeling, image segmentation, source separation, and grammar induction.

Motivation and Background

Most of machine learning is concerned with learning an appropriate set of parameters within a model class from ►training data. The meta-level problems of determining appropriate model classes are referred to as model selection or model adaptation. These constitute important concerns for machine learning practitioners, not only for avoidance of over-fitting and under-fitting, but also for discovery of the causes and structures underlying data. Examples of model selection and adaptation include selecting the number of clusters in a clustering problem, the number of hidden states in a hidden Markov model, the number of latent variables in a latent variable model, or the complexity of features used in nonlinear regression.

Nonparametric models constitute an approach to model selection and adaptation where the sizes of models are allowed to grow with data size. This is as opposed to *parametric models*, which use a fixed number of parameters. For example, a parametric approach to density estimation would be to fit a Gaussian or a mixture of a fixed number of Gaussians by maximum likelihood. A nonparametric approach would be a Parzen window estimator, which centers a Gaussian at each observation (and hence uses one mean parameter per observation). Another example is the support vector machine with a Gaussian kernel. The representer theorem shows that the decision function is a linear combination of Gaussian radial basis functions centered at every input vector, and thus has a complexity that grows with more observations. Nonparametric methods have long been popular in classical (non-Bayesian) statistics (Wasserman, 2006). They often perform impressively in applications and, though theoretical results for such models are typically harder to prove than for parametric models, appealing theoretical properties have been established for a wide range of models.

Bayesian nonparametric methods provide a Bayesian framework for model selection and adaptation using nonparametric models. A Bayesian formulation of nonparametric problems is nontrivial, since a Bayesian model defines prior and posterior distributions on a single fixed parameter space, but the dimension of the parameter space in a nonparametric approach should change with sample size. The Bayesian nonparametric solution to this problem is to use an infinite-dimensional parameter space, and to invoke only a finite subset of the available parameters on any given finite data set. This subset generally grows with the data set. In the context of Bayesian nonparametric models, "infinite-dimensional" can therefore be interpreted as "of finite but unbounded dimension." More precisely, a Bayesian nonparametric model is a model that (1) constitutes a Bayesian model on an infinite-dimensional parameter space and (2) can be evaluated on a finite sample in a manner that uses only a finite subset of the available parameters to explain the sample.

We make the above description more concrete in the next section when we describe a number of standard machine learning problems and the corresponding Bayesian nonparametric solutions. As we will see, the parameter space in (1) typically consists of functions or of measures, while (2) is usually achieved by marginalizing out surplus dimensions over the prior. Random functions and measures and, more generally, probability distributions on infinite-dimensional random objects are called *stochastic processes*; examples that we will encounter include Gaussian processes, Dirichlet processes, and beta processes. Bayesian nonparametric models are often named after the stochastic processes they contain. The examples are then followed by theoretical considerations, including formal constructions and representations of the stochastic processes used in Bayesian nonparametric models, exchangeability, and issues of consistency and convergence rate. We conclude this chapter with future directions and a list of literature available for reading.

Examples

Clustering with mixture models. Bayesian nonparametric generalizations of finite mixture models provide an approach for estimating both the number of components in a mixture model and the parameters of the individual mixture components simultaneously from data. Finite mixture models define a density function over data items x of the form $p(x) = \sum_{k=1}^{K} \pi_k p(x|\theta_k)$, where π_k is the mixing proportion and θ_k are parameters associated with component k. The density can be written in a non-standard manner as an integral: $p(x) = \int p(x|\theta)G(\theta)d\theta$, where $G = \sum_{k=1}^{K} \pi_k \delta_{\theta_k}$ is a discrete mixing distribution encapsulating all the parameters of the mixture model and δ_θ is a dirac distribution (atom) centered at θ. Bayesian nonparametric mixtures use

mixing distributions consisting of a *countably infinite* number of atoms instead:

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}. \quad (1)$$

This gives rise to mixture models with an infinite number of components. When applied to a finite training set, only a finite (but varying) number of components will be used to model the data, since each data item is associated with exactly one component but each component can be associated with multiple data items. Inference in the model then automatically recovers both the number of components to use and the parameters of the components. Being Bayesian, we need a prior over the mixing distribution G, and the most common prior to use is a *Dirichlet process* (DP). The resulting mixture model is called a DP mixture.

Formally, a Dirichlet process $DP(\alpha, H)$ parametrized by a concentration paramter $\alpha > 0$ and a base distribution H is a prior over distributions (probability measures) G such that, for any finite partition $A_1, \ldots, A_m$ of the parameter space, the induced random vector $(G(A_1), \ldots, G(A_m))$ is Dirichlet distributed with parameters $(\alpha H(A_1), \ldots, \alpha H(A_m))$ (see entitled Section "Theory" for a discussion of subtleties involved in this definition). It can be shown that draws from a DP will be discrete distributions as given in (1). The DP also induces a distribution over partitions of integers called the *Chinese restaurant process* (CRP), which directly describes the prior over how data items are clustered under the DP mixture. For more details on the DP and the CRP, see ►Dirichlet Process.

Nonlinear regression. The aim of regression is to infer a continuous function from a training set consisting of input–output pairs $\{(t_i, x_i)\}_{i=1}^n$. Parametric approaches parametrize the function using a finite number of parameters and attempt to infer these parameters from data. The prototypical Bayesian nonparametric approach to this problem is to define a prior distribution over continuous functions directly by means of a *Gaussian process* (GP). As explained in the Chapter ►Gaussian Process, a GP is a distribution on an infinite collection of random variables X_t, such that the joint distribution of each finite subset $X_{t_1}, \ldots, X_{t_m}$ is a multivariate Gaussian. A value x_t taken by the variable X_t can be regarded as the value of a continuous function f at t, that is, $f(t) = x_t$. Given the training set, the Gaussian process posterior is again a distribution on functions, conditional on these functions taking values $f(t_1) = x_1, \ldots, f(t_n) = x_n$.

Latent feature models. These models represent a set of objects in terms of a set of latent features, each of which represents an independent degree of variation exhibited by the data. Such a representation of data is sometimes referred to as a distributed representation. In analogy to nonparametric mixture models with an unknown number of clusters, a Bayesian nonparametric approach to latent feature modeling allows for an unknown number of latent features. The stochastic processes involved here are known as the *Indian buffet process* (IBP) and the *beta process* (BP). Draws from BPs are random discrete measures, where each of an infinite number of atoms has a mass in $(0, 1)$ but the masses of atoms need not sum to 1. Each atom corresponds to a feature, with the mass corresponding to the probability that the feature is present for an object. We can visualize the occurrences of features among objects using a binary matrix, where the (i, k) entry is 1 if object i has feature k and 0 otherwise. The distribution over binary matrices induced by the BP is called the IBP.

►*Hidden Markov models* (*HMMs*). HMMs are popular models for sequential or temporal data, where each time step is associated with a state, with state transitions dependent on the previous state. An *infinite HMM* is a Bayesian nonparametric approach to HMMs, where the number of states is unbounded and allowed to grow with the sequence length. It is defined using one DP prior for the transition probabilities going out from each state. To ensure that the set of states reachable from each outgoing state is the same, the base distributions of the DPs are shared and given a DP prior recursively. The construction is called a *hierarchical Dirichlet process* (HDP); see below.

►*Density estimation*. A nonparametric Bayesian approach to density estimation requires a prior on densities or distributions. However, the DP is not useful in this context, since it generates discrete distributions. A useful density estimator should smooth the empirical density (such as a Parzen window estimator), which requires a prior that can generate smooth distributions. Priors applicable in density estimation problems include DP mixture models and Pólya trees.

If $p(x|\theta)$ is a smooth density function, the density $\sum_{k=1}^{\infty} \pi_k p(x|\theta_k)$ induced by a DP mixture model is a

smooth random density, such that DP mixtures can be used as prior in density estimation problems.

Pólya trees are priors on probability distributions that can generate both discrete and piecewise continuous distributions, depending on the choice of parameters. Pólya trees are defined by a recursive infinitely deep binary subdivision of the domain of the generated random measure. Each subdivision is associated with a beta random variable which describes the relative amount of mass on each side of the subdivision. The DP is a special case of a Pólya tree corresponding to a particular parametrization. For other parametrizations the resulting random distribution can be smooth, so it is suitable for density estimation.

Power-law Phenomena. Many naturally occurring phenomena exhibit power-law behavior. Examples include natural languages, images, and social and genetic networks. An interesting generalization of the DP, called the *Pitman-Yor process*, $\mathrm{PYP}(\alpha, d, H)$, has recently been successfully used to model power-law data. The Pitman-Yor process augments the DP by a third parameter $d \in [0,1)$. When $d = 0$ the PYP is a $DP(\alpha, H)$, while when $\alpha = 0$ it is a so called *normalized stable process*.

Sequential modeling. HMMs model sequential data using latent variables representing the underlying state of the system, and assuming that each state only depends on the previous state (the so called Markov property). In some applications, for example language modeling and text compression, we are interested in directly modeling sequences without using latent variables, and without making any Markov assumptions, i.e., modeling each observation conditional on all previous observations in the sequence. Since the set of potential sequences of previous observations is unbounded, this calls for nonparametric models. A *hierarchical Pitman-Yor process* can be used to construct a Bayesian nonparametric solution whereby the conditional probabilities are coupled hierarchically.

Dependent and hierarchical models. Most of the Bayesian nonparametric models described so far are applied in settings where observations are homogeneous or exchangeable. In many real world settings observations are not homogeneous, and in fact are often structured in interesting ways. For example, the data generating process might change over time thus observations at different times are not exchangeable, or observations might come in distinct groups with those in the same group being more similar than across groups.

Significant recent efforts in Bayesian nonparametrics research have been placed in developing extensions that can handle these non-homogeneous settings. Dependent Dirichlet processes are stochastic processes, typically over a spatial or temporal domain, which define a Dirichlet process (or a related random measure) at each point with neighboring DPs being more dependent. These are used for spatial modeling, nonparametric regression, as well as for modeling temporal changes. Alternatively, hierarchical Bayesian nonparametric models like the hierarchical DP aim to couple multiple Bayesian nonparametric models within a hierarchical Bayesian framework. The idea is to allow sharing of statistical strength across multiple groups of observations. Among other applications, these have been used in the infinite HMM, topic modeling, language modeling, word segmentation, image segmentation, and grammar induction. For an overview of various dependent Bayesian nonparametric models and their applications in biostatistics please refer to Dunson (2010). Teh and Jordan (2010) is an overview of hierarchical Bayesian nonparametric models as well as a variety of applications in machine learning.

Theory

As we saw in the preceding examples, Bayesian nonparametric models often make use of priors over functions and measures. Because these spaces typically have uncountable number of dimensions, extra care has to be taken to define the priors properly and to study the asymptotic properties of estimation in the resulting models. In this section we give an overview of the basic concepts involved in the theory of Bayesian nonparametric models. We start with a discussion of the importance of exchangeability in Bayesian parametric and nonparametric statistics. This is followed by representations of the priors and issues of convergence.

Exchangeability

The underlying assumption of all Bayesian methods is that the parameter specifying the observation model is a random variable. This assumption is subject to

much criticism, and at the heart of the Bayesian versus non-Bayesian debate that has long divided the statistics community. However, there is a very general type of observation for which the existence of such a random variable can be derived mathematically: For so-called *exchangeable* observations, the Bayesian assumption that a randomly distributed parameter exists is not a modeling assumption, but a mathematical consequence of the data's properties.

Formally, a sequence of variables $X_1, X_2, \ldots, X_n$ over the same probability space $(\mathcal{X}, \Omega)$ is *exchangeable* if their joint distribution is invariant to permuting the variables. That is, if P is the joint distribution and σ any permutation of $\{1, \ldots, n\}$, then

$$P(X_1 = x_1, X_2 = x_2 \ldots X_n = x_n) = P(X_1 = x_{\sigma(1)}, X_2 = x_{\sigma(2)} \ldots X_n = x_{\sigma(n)}). \quad (2)$$

An infinite sequence $X_1, X_2, \ldots$ is *infinitely exchangeable* if $X_1, \ldots, X_n$ is exchangeable for *every* $n \geq 1$. In this chapter, we mean infinite exchangeability whenever we write exchangeability. Exchangeability reflects the assumption that the variables do not depend on their indices although they may be dependent among themselves. This is typically a reasonable assumption in machine learning and statistical applications, even if the variables are not themselves independently and identically distributed (iid).

Exchangeability is a much weaker assumption than iid since iid variables are automatically exchangeable.

If θ parametrizes the underlying distribution, and one assumes a prior distribution over θ, then the resulting marginal distribution over $X_1, X_2, \ldots$ with θ marginalized out will still be exchangeable. A fundamental result credited to de Finetti (1931) states that the converse is also true. That is, if $X_1, X_2, \ldots$ is (infinitely) exchangeable, then there is a random θ such that:

$$P(X_1, \ldots, X_n) = \int P(\theta) \prod_{i=1}^{n} P(X_i|\theta) d\theta \quad (3)$$

for every $n \geq 1$. In other words, the seemingly innocuous assumption of exchangeability automatically implies the existence of a hierarchical Bayesian model with θ being the random latent parameter. This the crux of the fundamental importance of exchangeability to Bayesian statistics.

In de Finetti's Theorem it is important to stress that θ can be infinite dimensional (it is typically a random measure), thus the hierarchical Bayesian model (3) is typically a nonparametric one. For an example, the Blackwell–MacQueen urn scheme (related to the CRP) is exchangeable and thus implicitly defines a random measure, namely the DP (see ▶Dirichlet Process for more details). In this sense, we will see below that de Finetti's theorem is an alternative route to Kolmogorov's extension theorem, which implicitly defines the stochastic processes underlying Bayesian nonparametric models.

Model Representations

In finite dimensions, a probability model is usually defined by a density function or probability mass function. In infinite dimensional spaces, this approach is not generally feasible, for reasons explained below. To define or work with a Bayesian nonparametric model, we have to choose alternative mathematical representations.

Weak distributions. A weak distribution is a representation for the distribution of a stochastic process, that is, for a probability distribution on an infinite-dimensional sample space. If we assume that the dimensions of the space are indexed by $t \in T$, the stochastic process can be regarded as the joint distribution P of an infinite set of random variables $\{X_t\}_{t \in T}$. For any finite subset $S \subset T$ of dimensions, the joint distribution P_S of the corresponding subset $\{X_t\}_{t \in S}$ of random variables is a finite-dimensional marginal of P. The *weak distribution* of a stochastic process is the set of all its finite-dimensional marginals, that is, the set $\{P_S : S \subset T, |S| < \infty\}$. For example, the customary definition of the Gaussian process as an infinite collection of random variables, each finite subset of which has a joint Gaussian distribution, is an example of a weak distribution representation. In contrast to the explicit representations to be described below, this representation is generally not generative, because it represents the distribution rather than a random draw, but is more widely applicable.

Apparently, just defining a weak distribution in this manner need not imply that it is a valid representation of a stochastic process. A given collection of finite-dimensional distributions represents a stochastic

process only (1) if a process with these distributions as its marginals actually exists, and (2) if it is uniquely defined by the marginals. The mathematical result which guarantees that weak distribution representations are valid is the *Kolmogorov extension theorem* (also known as the Daniell–Kolmogorov theorem or the Kolmogorov consistency theorem). Suppose that a collection $\{P_S : S \subset T, |S| < \infty\}$ of distributions is given. If all distributions in the collection are marginals of each other, that is, if P_{S_1} is a marginal of P_{S_2} whenever $S_1 \subset S_2$, the set of distributions is called a *projective family*. The Kolmogorov extension theorem states that, if the set T is countable, and if the distributions P_S form a projective family, then there exists a uniquely defined stochastic process with the collection $\{P_S\}$ as its marginal distributions. In other words, any projective family for a countable set T of dimensions is the weak distribution of a stochastic process. Conversely, any stochastic process can be represented in this manner, by computing its set of finite-dimensional marginals.

The weak distribution representation assumes that all individual random variable X_t of the stochastic process take values in the same sample space Ω. The stochastic process P defined by the weak distribution is then a probability distribution on the sample space Ω^T, which can be interpreted as the set of all functions $f : T \to \Omega$. For example, to construct a GP we might choose $T = \mathbb{Q}$ and $\Omega = \mathbb{R}$ to obtain real-valued functions on the countable space of rational numbers. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, the function f can then be extended to all of $\mathbb{R}$ by continuity. To define the DP as a distribution over probability measures on $\mathbb{R}$, we note that a probability measure is a set function that maps "random events," i.e., elements of the Borel σ-algebra $\mathcal{B}(\mathbb{R})$ of $\mathbb{R}$, into probabilities in $[0,1]$. We could therefore choose a weak distribution consisting of Dirichlet distributions, and set $T = \mathcal{B}(\mathbb{R})$ and $\Omega = [0,1]$. However, this approach raises a new problem because the set $\mathcal{B}(\mathbb{R})$ is not countable. As in the GP, we can first define the DP on a countable "base" for $\mathcal{B}(\mathbb{R})$ then extend to all random events by continuity of measures. More precise descriptions are unfortunately beyond the scope of this chapter.

Explicit representations. Explicit representations directly describe a random draw from a stochastic process, rather than its distribution. A prominent example of an explicit representation is the so-called *stick-breaking representation* of the Dirichlet process. The discrete random measure G in (1) is completely determined by the two infinite sequences $\{\pi_k\}_{k\in\mathbb{N}}$ and $\{\theta_k\}_{k\in\mathbb{N}}$. The stick-breaking representation of the DP generates these two sequences by drawing $\theta_k \sim H$ iid and $v_k \sim Beta(1,\alpha)$ for $k = 1, 2, \ldots$. The coefficients π_k are then computed as $\pi_k = v_k \prod_{j=1}^{k-1}(1 - v_k)$. The measure G so obtained can be shown to be distributed according to a $DP(\alpha, G_0)$. Similar representations can be derived for the Pitman–Yor process and the beta process as well. Explicit representations, if they exist for a given model, are typically of great practical importance for the derivation of algorithms.

Implicit Representations. A third representation of infinite dimensional models is based on de Finetti's Theorem. Any exchangeable sequence $X_1, \ldots, X_n$ uniquely defines a stochastic process θ, called the de Finetti measure, making the X_i's iid. If the X_i's are sufficient to define the rest of the model and their conditional distributions are easily specified, then it is sufficient to work directly with the X_i's and have the underlying stochastic process implicitly defined. Examples include the Chinese restaurant process (an exchangeable distribution over partitions) with the DP as the de Finetti measure, and the Indian buffet process (an exchangeable distribution over binary matrices) with the BP being the corresponding de Finetti measure. These implicit representations are useful in practice as they can lead to simple and efficient inference algorithms.

Finite representations. A fourth representation of Bayesian nonparametric models is as the infinite limit of finite (parametric) Bayesian models. For example, DP mixtures can be derived as the infinite limit of finite mixture models with particular Dirichlet priors on mixing proportions, GPs can be derived as the infinite limit of particular Bayesian regression models with Gaussian priors, while BPs can be derived as from the limit of an infinite number of independent beta variables. These representations are sometimes more intuitive for practitioners familiar with parametric models. However, not all Bayesian nonparametric models can be expressed in this fashion, and they do not necessarily make clear the mathematical subtleties involved.

Consistency and Convergence Rates

A recent series of works in mathematical statistics examines the convergence properties of Bayesian

nonparametric models, and in particular the questions of *consistency* and *convergence rates*. In this context, a Bayesian model is called consistent if, given that an infinite amount of data is available, the model posterior will concentrate in a neighborhood of the true solution (e.g., true function or density). A rate of convergence specifies, for a finite sample, how rapidly the posterior concentrates depending on the sample size. In their pioneering article Diaconis and Freedman (1986) showed, to the great surprise of much of the Bayesian community, that models such as the Dirichlet process can be inconsistent, and may converge to arbitrary solutions even for an infinite amount of data.

More recent results, notably by van der Vaart and Ghosal, apply modern methods of mathematical statistics to study the convergence properties of Bayesian nonparametric models (see e.g., Gho, (2010) and references therein). Consistency has been shown for a number of models, including Gaussian processes and Dirichlet process mixtures. However, a particularly interesting aspect of this line of work are results on convergence rates, which specify the rate of concentration of the posterior depending on sample size, on the complexity of the model, and on how much probability mass the prior places around the true solution. To make such results quantitative requires a measure for the complexity of a Bayesian nonparametric model. This is done by means of complexity measures developed in empirical process theory and statistical learning theory, such as metric entropies, covering numbers and bracketing, some of which are well-known in theoretical machine learning.

Inference

There are two aspects to inference from Bayesian nonparametric models: the analytic tractability of posteriors for the stochastic processes embedded in Bayesian nonparametric models, and practical inference algorithms for the overall models. Bayesian nonparametric models typically include stochastic processes such as the Gaussian process and the Dirichlet process. These processes have an infinite number of dimensions, hence naïve algorithmic approaches to computing posteriors are generally infeasible. Fortunately, these processes typically have analytically tractable posteriors, so all but finitely many of the dimensions can be analytically integrated out efficiently. The remaining dimensions, along with the parametric parts of the models, can then be handled by the usual inference techniques employed in parametric Bayesian modeling, including Markov chain Monte Carlo, sequential Monte Carlo, variational inference, and message-passing algorithms like expectation propagation. The precise choice of approximations to use will depend on the specific models under consideration, with speed/accuracy trade-offs between different techniques generally following those for parametric models. In the following, we will give two examples to illustrate the above points, and discuss a few theoretical issues associated with the analytic tractability of stochastic processes.

Examples

In Gaussian process regression, we model the relationship between an input x and an output y using a function f, so that $y \sim f(x) + \epsilon$, where ϵ is iid Gaussian noise. Given a GP prior over f and a finite training data set $\{(x_i, y_i)\}_{i=1}^n$ we wish to compute the posterior over f. Here we can use the weak representation of f and note that $\{f(x_i)\}_{i=1}^n$ is simply a finite-dimensional Gaussian with mean and covariance given by the mean and covariance functions of the GP. Inference for $\{f(x_i)\}_{i=1}^n$ is then straightforward. The approach can be thought of equivalently as marginalizing out the whole function except its values on the training inputs. Note that although we only have the posterior over $\{f(x_i)\}_{i=1}^n$, this is sufficient to reconstruct the function evaluated at any other point x_0 (say the test input), since $f(x_0)$ is Gaussian and independent of the training data $\{(x_i, y_i)\}_{i=1}^n$ given $\{f(x_i)\}_{i=1}^n$. In GP regression the posterior over $\{f(x_i)\}_{i=1}^n$ can be computed exactly. In GP classification or other regression settings with nonlinear likelihood functions, the typical approach is to use sparse methods based on variational approximations or expectation propagation; see Chapter ▶Gaussian Process for details.

Our second example involves Dirichlet process mixture models. Recall that the DP induces a clustering structure on the data items. If our training set consists of n data items, since each item can only belong to one cluster, there are at most n clusters represented in the training set. Even though the DP mixture itself has an infinite number of potential clusters, all but finitely

many of these are not associated with data, thus the associated variables need not be explicitly represented at all. This can be understood either as marginalizing out these variables, or as an implicit representation which can be made explicit whenever required by sampling from the prior. This idea is applicable for DP mixtures using both the Chinese restaurant process and the stick-breaking representations. In the CRP representation, each data item x_i is associated with a cluster index z_i, and each cluster k with a parameter θ_k^* (these parameters can be marginalized out if H is conjugate to F), and these are the only latent variables that need be represented in memory. In the stick-breaking representation, clusters are ordered by decreasing prior expected size, with cluster k associated with a parameter θ_k^* and a size π_k. Each data item is again associated with a cluster index z_i, and only the clusters up to $K = \max(z_1, \ldots, z_n)$ need to be represented. All clusters with index $> K$ need not be represented since their posterior conditioning on $\{(x_i, z_i)\}_{i=1}^n$ is just the prior.

On Bayes Equations and Conjugacy

It is worth noting that the posterior of a Bayesian model is, in abstract terms, defined as the conditional distribution of the parameter given the data and the hyperparameters, and this definition does not require the existence of a Bayes equation. If a Bayes equation exists for the model, the posterior can equivalently be defined as the left-hand side of the Bayes equation. However, for some stochastic processes, notably the DP on an uncountable space such as $\mathbb{R}$, it is not possible to define a Bayes equation even though the posterior is still a well-defined mathematical object. Technically speaking, existence of a Bayes equation requires the family of all possible posteriors to be dominated by the prior, but this is not the case for the DP. That posteriors of these stochastic processes can be evaluated at all is solely due to the fact that they admit an analytic representation.

The particular form of tractability exhibited by many stochastic processes in the literature is that of a *conjugate* posterior, that is, the posterior belongs to the same model family as the prior, and the posterior parameters can be computed as a function of the prior hyperparameters and the observed data. For example, the posterior of a $DP(\alpha, G_0)$ under observations $\theta_1, \ldots, \theta_n$ is again a Dirichlet process, $DP(\alpha + n, \frac{1}{\alpha+n}(\alpha G_0 + \sum \delta_{\theta_i}))$. Similarly the posterior of a GP under observations of $f(x_1), \ldots, f(x_n)$ is still a GP. It is this conjugacy that allows practical inference in the examples above. A Bayesian nonparametric model is conjugate if and only if the elements of its weak distribution, i.e., its finite-dimensional marginals, have a conjugate structure as well (Orbanz, 2010). In particular, this characterizes a class of conjugate Bayesian nonparametric models whose weak distributions consist of exponential family models. Note however, that lack of conjugacy does not imply intractable posteriors. An example is given by the Pitman–Yor process in which the posterior is given by a sum of a finite number of atoms and a Pitman-Yor process independent from the atoms.

Future Directions

Since MCMC (see ▶Markov Chain Monte Carlo) sampling algorithms for Dirichlet process mixtures became available in the 1990s and made latent variable models with nonparametric Bayesian components applicable to practical problems, the development of Bayesian nonparametrics has experienced explosive growth (Escobar & West, 1995; Neal, 2000). Arguably, though, the results available so far have only scratched the surface. The repertoire of available models is still mostly limited to using the Gaussian process, the Dirichlet process, the beta process, and generalizations derived from those. In principle, Bayesian nonparametric models may be defined on any infinite-dimensional mathematical object of possible interest to machine learning and statistics. Possible examples are kernels, infinite graphs, special classes of functions (e.g., piece-wise continuous or Sobolev functions), and permutations.

Aside from the obvious modeling questions, two major future directions are to make Bayesian nonparametric methods available to a larger audience of researchers and practitioners through the development of software packages, and to understand and quantify the theoretical properties of available methods.

General-Purpose Software Package

There is currently significant growth in the application of Bayesian nonparametric models across a

variety of application domains both in machine learning and in statistics. However significant hurdles still exist, especially the expense and expertise needed to develop computer programs for inference in these complex models. One future direction is thus the development of software packages that can compile efficient inference algorithms automatically given model specifications, thus allowing a much wider range of modeler to make use of these models. Current developments include the R DPpackage (http://cran.r-project.org/web/packages/DPpackage), the hierarchical Bayesian compiler (http://www.cs.utah.edu/hal/HBC), adaptor grammars (http://www.cog.brown.edu/mj/Software.htm), the MIT-Church project (http://projects.csail.mit.edu/church/wiki/Church), as well as efforts to add Bayesian nonparametric models to the repertoire of current Bayesian modeling environments like OpenBugs (http://mathstat.helsinki.fi/openbugs) and infer.NET (http://research.microsoft.com/en-us/um/cambridge/projects/infernet).

Statistical Properties of Models

Recent work in mathematical statistics provides some insight into the quantitative behavior of Bayesian nonparametric models (cf theory section). The elegant, methodical approach underlying these results, which quantifies model complexity by means of empirical process theory and then derives convergence rates as a function of the complexity, should be applicable to a wide range of models. So far, however, only results for Gaussian processes and Dirichlet process mixtures have been proven, and it will be of great interest to establish properties for other priors. Some models developed in machine learning, such as the infinite HMM, may pose new challenges to theoretical methodology, since their study will probably have to draw on both the theory of algorithms and mathematical statistics. Once a wider range of results is available, they may in turn serve to guide the development of new models, if it is possible to establish how different methods of model construction affect the statistical properties of the constructed model.

In addition to the references embedded in the text above, we recommend the books Hjort, Holmes, Müller, and Walker (2010), Ghosh and Ramamoorthi (2002), and the review articles Walker, Damien, Laud, and Smith (1999), Müller and Quintana (2004) on Bayesian nonparametrics. Further references can be found in the chapter by they Teh and Jordan (2010) of the book Hjort et al. (2010).

Cross References

►Bayesian Methods
►Dirichlet Processes
►Gaussian Processes
►Mixture Modelling
►Prior Probabilities

Recommended Reading

Diaconis, P., & Freedman, D. (1986) On the consistency of Bayes estimates (with discussion). *Annals of Statistics, 14*(1), 1–67.

Dunson, D. B. (2010). Nonparametric Bayes applications to biostatistics. In N. Hjort, C. Holmes, P. Müller, & S. Walker (Eds.), *Bayesian nonparametrics*. Cambridge: Cambridge University Press.

Escobar, M. D., & West, M. (1995). Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association, 90*, 577–588.

de Finetti, B. (1931). Funzione caratteristica di un fenomeno aleatorio. *Atti della R. Academia Nazionale dei Lincei, Serie 6. Memorie, Classe di Scienze Fisiche, Mathematice e Naturale, 4*, 251–299.

Ghosh, J. K., & Ramamoorthi, R. V. (2002). *Bayesian nonparametrics*. New York: Springer.

Hjort, N., Holmes, C., Müller, P., & Walker, S. (Eds.) (2010). Bayesian nonparametrics. In *Cambridge series in statistical and probabilistic mathematics* (No. 28). Cambridge: Cambridge University Press.

Müller, P., & Quintana, F. A. (2004). Nonparametric Bayesian data analysis. *Statistical Science, 19*(1), 95–110.

Neal, R. M. (2000). Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics, 9*, 249–265.

Orbanz, P. (2010). Construction of nonparametric Bayesian models from parametric Bayes equations. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, & A. Culotta (Eds.), *Advances in neural information processing systems, 22*, 1392–1400.

Teh, Y. W., & Jordan, M. I. (2010). Hierarchical Bayesian nonparametric models with applications. In N. Hjort, C. Holmes, P. Müller, & S. Walker (Eds.), *Bayesian nonparametrics*. Cambridge: Cambridge University Press.

Walker, S. G., Damien, P., Laud, P. W., & Smith, A. F. M. (1999). Bayesian nonparametric inference for random distributions and related functions. *Journal of the Royal Statistical Society, 61*(3), 485–527.

Wasserman, L. (2006). *All of nonparametric statistics*. New York: Springer.

Bayesian Reinforcement Learning

Pascal Poupart
University of Waterloo, Waterloo, Ontario, Canada

Synonyms

Adaptive control processes; Bayes adaptive Markov decision processes; Dual control; Optimal learning

Definition

Bayesian reinforcement learning refers to ►reinforcement learning modeled as a Bayesian learning problem (see ►Bayesian Methods). More specifically, following Bayesian learning theory, reinforcement learning is performed by computing a posterior distribution on the unknowns (e.g., any combination of the transition probabilities, reward probabilities, value function, value gradient, or policy) based on the evidence received (e.g., history of past state–action pairs).

Motivation and Background

Bayesian reinforcement learning can be traced back to the 1950s and 1960s in the work of Bellman (1961), Fel'Dbaum (1965), and several of Howard's students (Martin, 1967). Shortly after ►Markov decision processes were formalized, the above researchers (and several others) in Operations Research considered the problem of controlling a Markov process with uncertain transition and reward probabilities, which is equivalent to reinforcement learning. They considered Bayesian techniques since Bayesian learning is performed by probabilistic inference, which naturally combines with decision theory. In general, Bayesian reinforcement learning distinguishes itself from other reinforcement learning approaches by the use of probability distributions (instead of point estimates) to fully capture the uncertainty. This enables the learner to make more informed decisions, with the potential of learning faster with less data. In particular, the exploration/exploitation tradeoff can be naturally optimized. The use of a prior distribution also facilitates the encoding of domain knowledge, which is exploited in a natural and principled way by the learning process.

Structure of Learning Approach

A Markov decision process (MDP) (Puterman, 1994) can be formalized by a tuple $\langle S, A, T \rangle$ where S is the set of states s, A is the set of actions a, $T(s,a,s') = \Pr(s'|s,a)$ is the transition distribution indicating the probability of reaching s' when executing a in s. Let s_r denote the reward feature of a state and $\Pr(s'_r|s,a)$ be the probability of earning r when executing a in s. A policy $\pi : S \rightarrow A$ consists of a mapping from states to actions. For a given discount factor $0 \leq \gamma \leq 1$ and horizon h, the value V^π of a policy π is the expected discounted total reward earned while executing this policy: $V^\pi(s) = \sum_{t=o}^{h} \gamma^t E_{s|\pi}[s_r^t]$. The value function V^π can be written in a recursive form as the expected sum of the immediate reward s'_r with the discounted future rewards: $V^\pi(s) = \sum_{s'} \Pr(s'|s,\pi(s))[s'_r + \gamma V^\pi(s')]$. The goal is to find an optimal policy π^*, that is, a policy with the highest value V^* in all states (i.e., $V^*(s) \geq V^\pi(s)\ \forall \pi, s$). Many algorithms exploit the fact that the optimal value function V^* satisfies Bellman's equation:

$$V^*(s) = \max_a \sum_{s'} \Pr(s'|s,a)[s'_r + \gamma V^*(s)] \qquad (1)$$

Reinforcement learning (Sutton & Barto, 1998) is concerned with the problem of finding an optimal policy when the transition (and reward) probabilities T are unknown (or uncertain). Bayesian learning is a learning approach in which unknowns are modeled as random variables X over which distributions encode the uncertainty. The process of learning consists of updating the prior distribution $\Pr(X)$ based on some evidence e to obtain a posterior distribution $\Pr(X|e)$ according to Bayes theorem: $\Pr(X|e) = k \Pr(X) \Pr(e|X)$. (Here $k = 1/\Pr(e)$ is a normalization constant.) Hence, *Bayesian reinforcement learning* consists of using Bayesian learning for reinforcement learning. The unknowns are the transition (and reward) probabilities T, the optimal value function V^*, and the optimal policy π^*. Techniques that maintain a distribution on T are known as *model-based* Bayesian reinforcement learning since they explicitly learn the underlying model T. In contrast, techniques that maintain a distribution on V^* or π^* are known as *model-free* Bayesian reinforcement learning since they directly learn the optimal value function or policy without learning a model.

Model-Based Bayesian Learning

In model-based Bayesian reinforcement learning, the learner starts with a prior distribution over the parameters of T, which we denote by θ. For instance, let $\theta_{sas'} = \Pr(s'|s,a,\theta)$ be the unknown probability of reaching s' when executing a in s. In general, we denote by θ the set of all $\theta_{sas'}$. Then, the prior $b(\theta)$ represents the initial *belief* of the learner regarding the underlying model. The learner updates its belief after every s,a,s' triple observed by computing a posterior $b_{sas'}(\theta) = b(\theta|s,a,s')$ according to Bayes theorem:

$$b_{sas'}(\theta) = kb(\theta)\Pr(s'|s,a,\theta) = kb(\theta)\theta_{sas'}. \quad (2)$$

In order to facilitate belief updates, it is convenient to pick the prior from a family of distributions that is closed under Bayes updates. This ensures that beliefs are always parameterized in the same way. Such families are called *conjugate priors*. In the case of a discrete model (i.e., $\Pr(s'|s,a,\theta)$ is a discrete distribution), Dirichlets are conjugate priors and form a family of distributions corresponding to monomials over the simplex of discrete distributions (DeGroot, 1970). They are parameterized as follows: $Dir(\theta;n) = k\prod_i \theta_i^{n_i-1}$. Here θ is an unknown discrete distribution such that $\sum_i \theta_i = 1$ and n is a vector of strictly positive real numbers n_i (known as the hyperparameters) such that $n_i - 1$ can be interpreted as the number of times that the θ_i-probability event has been observed. Since the unknown transition model θ is made up of one unknown distribution θ_a^s per s,a pair, let the prior be $b(\theta) = \prod_{s,a} Dir(\theta_a^s; n_a^s)$ such that n_a^s is a vector of hyperparameters $n_a^{s,s'}$. The posterior obtained after transition $\hat{s},\hat{a},\hat{s}'$ is

$$\begin{aligned} b_a^{s,s'}(\theta) &= k\theta_a^{s,s'} \prod_{s,a} Dir(\theta_a^s; n_a^s) \\ &= \prod_{s,a} Dir(\theta_a^s; n_a^s + \delta_{\hat{s},\hat{a},\hat{s}'}(s,a,s')) \end{aligned} \quad (3)$$

where $\delta_{\hat{s},\hat{a},\hat{s}'}(s,a,s')$ is a Kronecker delta that returns 1 when $s = \hat{s}$, $a = \hat{a}$, $s' = \hat{s}'$ and 0 otherwise. In practice, belief monitoring is as simple as incrementing the hyperparameter corresponding to the observed transition.

Belief MDP Equivalence

At any point in time, the belief b provides an explicit representation of the uncertainty of the learner about the underlying model. This information is very useful to decide whether future actions should focus on exploring or exploiting. Hence, in Bayesian reinforcement learning, policies π are mappings from state-belief pairs $\langle s,b\rangle$ to actions. Equivalently, the problem of Bayesian reinforcement learning can be thought as one of planning with a belief MDP (or a partially observable MDP). More precisely, every Bayesian reinforcement learning problem has an equivalent belief MDP formulation $\langle S_{bel}, A_{bel}, T_{bel}\rangle$ where $S_{bel} = S \times B$ (B is the space of beliefs b), $A_{bel} = A$, and $T_{bel}(s_{bel}, a_{bel}, b'_{bel}) = \Pr(b'_{bel}|b_{bel}, a_{bel}) = \Pr(s',b'|s,b,a) = \Pr(b'|s,b,a,s')\Pr(s'|s,b,a)$. The decomposition of the transition dynamics is particularly interesting since $\Pr(b'|s,b,a,s')$ equals 1 when $b' = b_a^{s,s'}$ (as defined in Eq. 3) and 0 otherwise. Furthermore, $\Pr(s'|s,b,a) = \int_\theta b(\theta)\Pr(s'|s,\theta,a)d\theta$, which can be computed in closed form when b is a Dirichlet. As a result, the transition dynamics of the belief MDP are fully known. This is a remarkable fact since it means that Bayesian reinforcement learning problems, which by definition have unknown/uncertain transition dynamics, can be recast as belief MDPs with known transition dynamics. While this doesn't make the problem any easier since the belief MDP has a hybrid state space (discrete s with continuous b), it allows us to treat policy optimization as a problem of planning and in particular to adapt algorithms originally designed for belief MDPs (also known as partially observable MDPs).

Optimal Value Function Parameterization

Many planning techniques compute the optimal value function V^*, from which an optimal policy π^* can easily be extracted. Despite the hybrid nature of the state space, the optimal value function (for a finite horizon) has a simple parameterization corresponding to the upper envelope of a set of polynomials (Poupart, Vlassis, Hoey, & Regan, 2006). Recall that the optimal value function satisfies Bellman's equation, which can be adapted as follows for a belief MDP:

$$V^*(s,b) = \max_a \sum_{s'} \Pr(s',b'|s,b,a)\left[s'_r + \gamma V^*(s',b')\right]. \quad (4)$$

Using the fact that b' must be $b_a^{s,s'}$ (otherwise $\Pr(s',b'|s,b,a) = 0$) allows us to rewrite Bellman's equation as follows:

$$V^*(s,b) = \max_a \sum_{s'} \Pr(s'|s,b,a)\left[s'_r + \gamma V^*\left(s', b_a^{s,s'}\right)\right]. \quad (5)$$

Let Γ^n be a set of polynomials in θ such that the optimal value function V^n with n steps to go is $V^n(s,b) = \int_\theta b(\theta) poly_{s,b}(\theta) d\theta$ where $poly_{s,b} = \text{argmax}_{poly \in \Gamma_s^n} \int_\theta b(\theta) poly(\theta) d\theta$. It suffices to replace $\Pr(s'|s,b,a)$, $b_a^{s,s'}$ and V^n by their definitions in Bellman's equation

$$V^{n+1}(s,b) = \max_a \sum_{s'} \int_\theta b(\theta) \Pr(s'|s,\theta,a) \left[r'_s + \gamma\, poly_{s', b_a^{s,s'}}(\theta)\right] d\theta \quad (6)$$

$$= \max_a \int_\theta b(\theta) \sum_{s'} \theta_a^{s,s'} \left[r'_s + \gamma\, poly_{s', b_a^{s,s'}}(\theta)\right] d\theta \quad (7)$$

to obtain a similar set of polynomials $\Gamma_s^{n+1} = \left\{\sum_{s'} \theta_a^{s,s'} \left[r'_s + \gamma\, poly'_s(\theta)\right] | a \in A,\ poly_{s'} \in \Gamma_{s'}^n\right\}$ that represents V^{n+1}.

The fact that the optimal value function has a closed form with a simple parameterization is quite useful for planning algorithms based on value iteration. Instead of using an arbitrary function approximator to fit the value function, one can take advantage of the fact that the value function can be represented by a set of polynomials to choose a good representation. For instance, the Beetle algorithm (Poupart et al., 2006) performs point-based value iteration and approximates the value function with a bounded set of polynomials that each consists of a linear combination of monomial basis functions.

Exploration/Exploitation Tradeoff

Since the underlying model is unknown in reinforcement learning, it is not clear whether actions should be chosen to explore (gain more information about the model) or exploit (maximize immediate rewards based on information gathered so far). Bayesian reinforcement learning provides a principled solution to the exploration/exploitation tradeoff. Despite the appearance of multiple objectives induced by exploration and exploitation, there is a single objective in reinforcement learning: maximize total discounted rewards. Hence, an optimal policy (which maximizes total discounted rewards) must naturally optimize the exploration/exploitation tradeoff. In order for a policy to be optimal, it must use all the information available. The information available to the learner consists of the history of past states and actions. One can show that state–belief pairs $\langle s, b\rangle$ are sufficient statistics of the history. Hence, by searching for the mapping from state–belief pairs to actions that maximizes total discounted rewards, Bayesian reinforcement learning implicitly seeks an optimal tradeoff between exploration and exploitation. In contrast, traditional reinforcement learning approaches search in the space of mappings from states to actions. As a result, such techniques typically focus on asymptotic convergence (i.e., convergence to a policy that is optimal in the limit), but do not effectively balance exploration and exploitation since they do not use histories or beliefs to quantify the uncertainty about the underlying model.

Related Work

Michael Duff's PhD thesis (Duff, 2002) provides an excellent survey of Bayesian reinforcement learning up until 2002. The above text pertains mostly to model-based Bayesian reinforcement learning applied to discrete, fully observable, single agent domains. Bayesian learning has also been explored in model-free reinforcement learning (Dearden, Friedman, & Russell, 1998; Engel, Mannor, & Meir, 2005; Ghavamzadeh & Engel, 2006) continuous-valued state spaces (Ross, Chaib-Draa, & Pineau, 2008), partially observable domains (Poupart & Vlassis, 2008; Ross, Chaib-Draa, & Pineau, 2007), and multi-agent systems (Chalkiadakis & Boutilier, 2003, 2004; Gmytrasiewicz & Doshi, 2005).

Cross References

►Active Learning
►Markov Decision Processes
►Reinforcement Learning

Recommended Reading

Bellman, R. (1961). *Adaptive control processes: A guided tour*. Princeton, NJ: Princeton University Press.

Chalkiadakis, G., & Boutilier, C. (2003). Coordination in multiagent reinforcement learning: A Bayesian approach. In *International joint conference on autonomous agents and multiagent systems (AAMAS)*, Melbourne, Australia (pp. 709–716).
Chalkiadakis, G., & Boutilier, C. (2004). Bayesian reinforcement learning for coalition formation under uncertainty. In *International joint conference on autonomous agents and multiagent systems (AAMAS)*, New York (pp. 1090–1097).
Dearden, R., Friedman, N., & Russell, S. J. (1998). Bayesian Q-learning. In *National conference on artificial intelligence (AAAI)*, Madison, Wisconsin (pp. 761–768).
DeGroot, M. H. (1970). *Optimal statistical decisions*. New York: McGraw-Hill.
Duff, M. (2002). *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts, Amherst.
Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. In *International conference on machine learning (ICML)*, Bonn, Germany.
Fel'Dbaum, A. (1965). *Optimal control systems*. New York: Academic.
Ghavamzadeh, M., & Engel, Y. (2006). Bayesian policy gradient algorithms. In *Advances in neural information processing systems (NIPS)*, (pp. 457–464).
Gmytrasiewicz, P., & Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research (JAIR), 24*, 49–79.
Martin (1967). *Bayesian decision problems and Markov chains*. New York: Wiley.
Poupart, P., & Vlassis, N. (2008). Model-based Bayesian reinforcement learning in partially observable domains. In *International symposium on artificial intelligence and mathematics (ISAIM)*.
Poupart, P., Vlassis, N., Hoey, J., & Regan, K. (2006). An analytic solution to discrete Bayesian reinforcement learning. In *International conference on machine learning (ICML)*, Pittsburgh, Pennsylvania (pp. 697–704).
Puterman, M. L. (1994). *Markov decision processes*. New York: Wiley.
Ross, S., Chaib-Draa, B., & Pineau, J. (2007). Bayes-adaptive POMDPs. In *Advances in neural information processing systems (NIPS)*.
Ross, S., Chaib-Draa, B., & Pineau, J. (2008). Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In *IEEE International conference on robotics and automation (ICRA)*, (pp. 2845–2851).
Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning*. Cambridge, MA: MIT Press.

Beam Search

Claude Sammut
University of New South Wales, Sydney, Australia

A beam search is a heuristic search technique that combines elements of breadth-first and best-first searches. Like a breadth-first search, the beam search maintains a list of nodes that represent a frontier in the search space. Whereas the breadth-first adds all neighbors to the list, the beam search orders the neighboring nodes according to some heuristic and only keeps the n best, where n is the *beam size*. This can significantly reduce the processing and storage requirements for the search.

In machine learning, the beam search has been used in algorithms, such as AQ11 (Dietterich & Michalski, 1977).

Cross References

►Learning as Search

Recommended Reading

Dietterich, T. G., & Michalski, R. S. (1977). Learning and generalization of characteristic descriptions: Evaluation criteria and comparative review of selected methods. In *Fifth international joint conference on artificial intelligence* (pp. 223–231). Cambridge, MA: William Kaufmann.

Behavioral Cloning

Caude Sammut
The University of New South Wales, Sydney, Australia

Synonyms

Apprenticeship learning; Behavioral cloning; Learning by demonstration; Learning by imitation; Learning control rules

Definition

Behavioral cloning is a method by which human subcognitive skills can be captured and reproduced in a computer program. As the human subject performs the skill, his or her actions are recorded along with the situation that gave rise to the action. A log of these records is used as input to a learning program. The learning program outputs a set of rules that reproduce the skilled behavior. This method can be used to construct automatic control systems for complex tasks for which classical control theory is inadequate. It can also be used for training.

Motivation and Background

Behavioral cloning (Michie, Bain, & Hayes-Michie, 1990) is a form of *learning by imitation* whose main motivation is to build a model of the behavior of a human when performing a complex skill. Preferably, the model should be in a readable form. It is related to other forms of learning by imitation, such as ▶inverse reinforcement learning (Abbeel & Ng, 2004; Amit & Matarić, 2002; Hayes & Demiris, 1994; Kuniyoshi, Inaba, & Inoue, 1994; Pomerleau, 1989) and methods that use data from human performances to model the system being controlled (Atkeson & Schaal, 1997; Bagnell & Schneider, 2001).

Experts might be defined as people who know what they are doing not what they are talking about. That is, once a person becomes highly skilled in some task, the skill becomes sub-cognitive and is no longer available to introspection. So when the person is asked to explain why certain decisions were made, the explanation is a *post hoc* justification rather than a true explanation.

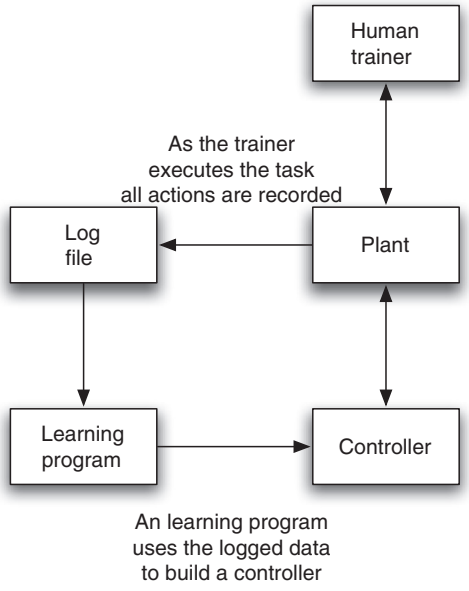
Michie et al. (1990) used an induction program to learn rules for balancing a pole (in simulation) and earlier work by Donaldson (1960), Widrow and Smith (1964), and Chambers and Michie (1969) demonstrated the feasibility of learning by imitation, also for pole-balancing.

Structure of the Learning System

Behavioral cloning assumes that there is a plant of some kind that is under the control of a human operator. The plant may be a physical system or a simulation. In either case, the plant must be instrumented so that it is possible to capture the state of the system, including all the control settings. Thus, whenever the operator performs an action, that is, changes a control setting, we can associate that action with a particular state.

Let us use a simple example of a system that has only one control action. A *pole balancer* has four state variables: the angle of the pole, θ, and its angular velocity, $\dot{\theta}$ and the position, x, and velocity $\dot{x}$, of the cart on the track. The only action available to the controller is to apply a fixed positive of negative force, F, to accelerate the cart left or right.

We can create an experimental setup where a human can control a pole and cart system (either real or in simulation) by applying a left push or a right push at



Behavioral Cloning. Figure 1. Structure of learning system

the appropriate time. Whenever a control action is performed, we record the action as well as values of the four state variables at the time of the action. Each of these records can be viewed as an example of a mapping from state to action.

Michie et al. (1990) demonstrated that it is possible to construct a controller by learning from these examples. The learning task is to predict the appropriate action, given the state. They used a ▶decision tree learning program to produce a classifier that, given the values of the four state variables, would output an action. A decision tree is easily convertible into an executable code as a nested **if** statement. The quality of the controller can be tested by inserting the decision tree into the simulator, replacing the human operator.

If the goal of learning is simply to produce an operational controller then any program capable of building a classifier could be used. The reason that Michie et al. (1990) chose a symbolic learner was their desire to produce a controller whose decision making was transparent as well as operational. That is, it should be possible to extract an explanation of the behavior that is meaningful to an expert in the task.

Learning Direct (Situation–Action) Controllers

A controller such as the one described above is referred to as a *direct controller* because it maps situations to actions. Other examples of learning a direct controller

are building an autopilot from behavioral traces of human pilots flying aircraft in a flight simulator (Sammut, Hurst, Kedzier, & Michie, 1992) and building a control system for a container crane (Urbančič & Bratko, 1994). These systems extended the earlier work by operating in domains in which there is more than one control variable and the task is sufficiently complex that it must be decomposed into several subtasks.

An operator of a container crane can control the speed of the cart and the length of the rope. A pilot of a fixed-wing aircraft can control the ailerons, elevators, rudder, throttle, and flaps. To build an autopilot, the learner must build a system that can set each of the control variables. Sammut et al. (1992), viewed this as a multitask learning problem.

Each training example is a feature vector that includes the position, orientation, and velocities of the aircraft as well as the values of each of the control settings: ailerons, elevator, throttle, and flaps. The rudder is ignored. A separate decision tree is built for each control variable. For example, the aileron setting is treated as the dependent variable and all the other variables, including the other controls, are treated as the attributes of the training example. A decision tree is built for ailerons, then the process is repeated for the elevators, etc. The result is a decision tree for each control variable.

The autopilot code executes each decision tree in each cycle of the control loop. This method treats the setting of each control as a separate task. It may be surprising that this method works since it is often necessary to adjust more than one control simultaneously to achieve the desired result. For example, to turn, it is normal to use the ailerons to roll the aircraft while adjusting the elevators to pull it around. This kind of multivariable control does result from multiple decision trees. When, say, the aileron decision tree initiates a roll, the elevator's decision tree detects the roll and causes the aircraft to pitch up and execute a turn.

Limitations Direct controllers work quite well for systems that have a relatively small state space. However, for complex systems, behavioral cloning of direct situation–action rules tends to produce very brittle controllers. That is, they cannot tolerate large disturbances. For example, when air turbulence is introduced into the flight simulator, the performance of the clone degrades very rapidly. This is because the examples provided by logging the performance of a human only cover a very small part of the state space of a complex system such as an aircraft in flight. Thus, the"expertise" of the controller is very limited. If the system strays outside the controller's region of expertise, it has no method for recovering and failure is usually catastrophic.

More robust control is possible but only with a significant change in approach. The more successful methods decompose the learning task into two stages: learning goals and learning the actions to achieve those goals.

Learning Indirect (Goal-Directed) Controllers

The problem of learning in a large search space can partially be addressed by decomposing the learning into subtasks. A controller built in this way is said to be an *indirect controller*. A control is "indirect" if it does not compute the next action directly from the system's current state but uses, in addition, some intermediate information. An example of such intermediate information is a subgoal to be attained before achieving the final goal.

Subgoals often feature in an operator's control strategies and can be automatically detected from a trace of the operator's behavior (Šuc & Bratko, 1997). The problem of subgoal identification can be treated as the inverse of the usual problem of controller design, that is, given the actions in an operator's trace, find the goal that these actions achieve. The limitation of this approach is that it only works well for cases in which there are just a few subgoals, not when the operator's trajectory contains many subgoals. In these cases, a better approach is to generalize the operator's trajectory. The generalized trajectory can be viewed as defining a continuously changing subgoal (Bratko & Šuc, 2002; Šuc & Bratko, 1999a) (see also the use of flow tubes in dynamic plan execution (Hofmann & Williams, 2006)).

Subgoals and generalized trajectories are not sufficient to define a controller. A model of the systems dynamics is also required. Therefore, in addition to inducing subgoals or a generalized trajectory, this approach also requires learning approximate system dynamics, that is a model of the controlled system. Bratko and Šuc (2003) and Šuc and Bratko (1999b) use a combination of the Goldhorn (Križman & Džeroski,

1995) discovery program and locally weighted regression to build the model of the system's dynamics. The next action is then computed "indirectly" by (1) computing the desired next state (e.g., next subgoal) and (2) determining an action that brings the system to the desired next state. Bratko and Šuc also investigated building qualitative control strategies from operator traces (Bratko & Šuc, 2002).

An analog to this approach is ►inverse reinforcement learning (Abbeel & Ng, 2004; Atkeson & Schaal, 1997; Ng & Russell, 2000) where the reward function is learned. Here, the learning the reward function corresponds to learning the human operator's goals.

Isaac and Sammut (2003) uses an approach that is similar in spirit to Šuc and Bratko but incorporates classical control theory. Learned skills are represented by a two-level hierarchical decomposition with an anticipatory goal level and a reactive control level. The goal level models how the operator chooses goal settings for the control strategy and the control level models the operator's reaction to any error between the goal setting and actual state of the system. For example, in flying, the pilot can achieve goal values for the desired heading, altitude, and airspeed by choosing appropriate values of turn rate, climb rate, and acceleration. The controls can be set to correct errors between the current state and the desired state of these goal-directing quantities. Goal models map system states to a goal setting. Control actions are based on the error between the output of each of the goal models and the current system state.

The control level is modeled as a set of proportional integral derivative (PID) controllers, one for each control variable. A PID controller determines a control value as a linear function proportional to the error on a goal variable, the integral of the error, and the derivative of the error.

Goal setting and control models are learned separately. The process begins be deciding which variables are to be used for the goal settings. For example, trainee pilots will learn to execute a "constant-rate turn," that is, their goal is to maintain a given turn rate. A separate goal rule is constructed for each goal variable using a ►model tree learner (Potts & Sammut, 2005).

A goal rule gives the setting for a goal variable and therefore, we can find the difference (error) between the current state value and the goal setting. The integral and derivative of the error can also be calculated. For example, if the set turn rate is 180° min, then the error on the turn rate is calculated as the actual turn rate minus 180. The integral is then the running sum of the error multiplied by the time interval between time samples, starting from the first time sample of the behavioral trace, and the derivative is calculated as the difference between the error and previous error all divided by the time interval.

For each control available to the operator, a model tree learner is used to predict the appropriate control setting. ►Linear regression is used in the leaf nodes of the model tree to produce linear equations whose coefficients are the *P*, *I*, and *D* of values of the PID controller. Thus the learner produces a collection of PID controllers that are selected according to the conditions in the internal nodes of the tree. In control theory, this is known as *piecewise linear control*.

Another indirect method is to learn a model of the dynamics of the system and use this to learn, in simulation, a controller for the system (Bagnell & Schneider, 2001; Ng, Jin Kim, Jordan, & Sastry, 2003). This approach does not seek to directly model the behavior of a human operator. A behavioral trace may be used to generate data for modeling the system but then a reinforcement learning algorithm is used to generate a policy for controlling the simulated system. The learned policy can then be transferred to the physical system. ►Locally weighted regression is typically used for system modeling, although ►model trees can also be used.

Cross References

►Apprenticeship Learning
►Inverse Reinforcement Learning
►Learning by Imitation
►Locally Weighted Regression
►Model Trees
►Reinforcement Learning
►System Identification

Recommended Reading

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *International conference on machine learning*, Banff, Alberta, Canada. New York: ACM.

Amit, R., & Matarić, M. (2002). Learning movement sequences from demonstration. In *Proceedings of the second international conference on development and learning*, Cambridge, MA, USA (pp. 203–208). Washington, D.C.: IEEE.

Atkeson, C. G., & Schaal, S. (1997). Robot learning from demonstration. In D. H. Fisher (Ed.), *Proceedings of the fourteenth international conference on machine learning*, Nashville, TN, USA (pp. 12–20). San Francisco: Morgan Kaufmann.

Bagnell, J. A., & Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *International conference on robotics and automation*, South Korea. IEEE Press, New York.

Bratko, I., & Šuc, D. (2002). Using machine learning to understand operator's skill. In *Proceedings of the 15th international conference on industrial and engineering applications of artificial intelligence and expert systems* (pp. 812–823). London: Springer. AAAI Press, Menlo Park, CA.

Bratko, I., & Šuc, D. (2003). Learning qualitative models. *AI Magazine, 24*(4), 107–119.

Chambers, R. A., & Michie, D. (1969). Man-machine co-operation on a learning task. In R. Parslow, R. Prowse, & R. Elliott-Green (Eds.), *Computer graphics: techniques and applications*. London: Plenum.

Donaldson, P. E. K. (1960). Error decorrelation: A technique for matching a class of functions. In *Proceedings of the third international conference on medical electronics* (pp. 173–178).

Hayes, G., & Demiris, J. (1994). A robot controller using learning by imitation. In *Proceedings of the international symposium on intelligent robotic systems*, Grenoble, France (pp. 198–204). Grenoble: LIFTA-IMAG.

Hofmann, A. G., & Williams, B. C. (2006). Exploiting spatial and temporal flexiblity for plan execution of hybrid, under-actuated systems. In *Proceedings of the 21st national conference on artficial intelligence*, July 2006, Boston, MA (pp. 948–955).

Isaac, A., & Sammut, C. (2003). Goal-directed learning to fly. In T. Fawcett & N. Mishra (Eds.), *Proceedings of the twentieth international conference on machine learning*, Washington, D.C. (pp. 258–265). Menlo Park: AAAI.

Križman, V., & Džeroski, S. (1995). Discovering dynamics from measured data. *Electrotechnical Review, 62*(3–4), 191–198.

Kuniyoshi, Y., Inaba, M., & Inoue, H. (1994). Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation, 10*, 799–822.

Michie, D., Bain, M., & Hayes-Michie, J. E. (1990). Cognitive models from subcognitive skills. In M. Grimble, S. McGhee, & P. Mowforth (Eds.), *Knowledge-based systems in industrial control*. Stevenage: Peter Peregrinus.

Ng, A. Y., Jin Kim, H., Jordan, M. I., & Sastry, S. (2003). Autonomous helicopter flight via reinforcement learning. In S. Thrun, L. Saul, & B. Schölkopf (Eds.), *Advances in neural information processing systems 16*. Cambridge: MIT Press.

Ng, A. Y., & Russell, S. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of 17th international conference on machine learning*, Stanford, CA, USA (pp. 663–670). San Francisco: Morgan Kaufmann.

Pomerleau, D. A. (1989). ALVINN: An autonomous land vehicle in a neural network. In D. S. Touretzky (Ed.), *Advances in neural information processing systems*. San Mateo: Morgan Kaufmann.

Potts, D., & Sammut, C. (November 2005). Incremental learning of linear model trees. *Machine Learning, 6*(1–3), 5–48.

Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. In D. Sleeman & P. Edwards (Eds.), *Proceedings of the ninth international conference on machine learning*, Aberdeen (pp. 385–393). San Francisco: Morgan Kaufmann.

Šuc, D., & Bratko, I. (1997). Skill reconstruction as induction of LQ controllers with subgoals. In *IJCAI-97: Proceedings of the fifteenth international joint conference on artificial intelligence*, Nagoya, Japan (Vol. 2, pp. 914–920). San Francisco: Morgan Kaufmann.

Šuc, D., & Bratko, I. (1999a). Modelling of control skill by qualitative constraints. In *Thirteenth international workshop on qualitative reasoning*, 7–9 June 1999, Lock Awe, Scotland (pp. 212–220). Aberystwyth: University of Aberystwyth.

Šuc, D., & Bratko, I. (1999b). Symbolic and qualitative reconstruction of control skill. *Electronic Transactions on Artificial Intelligence, 3*(B), 1–22.

Urbančič, T., & Bratko, I. (1994). Reconstructing human skill with machine learning. In A. Cohn (Ed.), *Proceedings of the 11th European conference on artificial intelligence*. Wiley. Amsterdam: New York.

Widrow, B., & Smith, F. W. (1964). Pattern recognising control systems. In J. T. Tou & R. H. Wilcox (Eds.), *Computer and information sciences*. London: Clever Hume.

Belief State Markov Decision Processes

►Partially Observable Markov Decision Processes

Bellman Equation

The *Bellman Equation* is a recursive formula that forms the basis for ►dynamic programming. It computes the expected total reward of taking an action from a state in a ►Markov decision process by breaking it into the immediate reward and the total future expected reward. (See ►dynamic programming.)

Bias

Bias has two meanings, ►inductive bias, and *statistical bias* (see ►bias variance decomposition).

Bias Specification Language

Hendrik Blockeel
Katholieke Universiteit Leuven, Belgium
The Netherlands

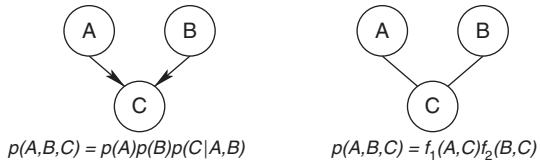
Definition

A *bias specification language* is a language in which a user can specify a ►Language Bias. The language bias of a learner is the set of hypotheses (or hypothesis descriptions) that this learner may return.

In contrast to the ►hypothesis language, the bias specification language allows us to describe not single hypotheses but sets (languages) of hypotheses.

Examples

In learning approaches based on ►graphical models or ►artificial neural networks, whenever the user provides the graph structure of the model, he or she is specifying a bias. The "language" used to specify this bias, in this case, consists of graphs. Figure 1 shows examples of such graphs. Not every kind of bias can necessarily be expressed by some bias specification language; for instance, the bias defined by the ►Bayesian network structure in Fig. 1 cannot be expressed using a ►Markov network. Bayesian networks and Markov networks have a different expressiveness, when viewed as bias specification languages.



Bias Specification Language. Figure 1. Graphs defining a bias for learning joint distributions. The Bayesian network structure to the left constrains the form of the joint distribution in a particular way (shown as the equation below the graph). Notably, it guarantees that only distributions can be learned in which the variables *A* and *B* are (unconditionally) independent. The Markov network structure to the right constrains the form of the joint distribution in a different way: it states that it must be possible to write the distribution as a product of a function of *A* and *C* and a function of *B* and *C*. These two biases are different. In fact, no Markov network structure over the variables *A*, *B*, and *C* exists that expresses the bias specified by the Bayesian network structure

Also certain parameters of decision tree learners or rule set learners effectively restrict the hypothesis language (for instance, an upper bound on the rule length or the size of the decision tree).

A combination of parameter values can hardly be called a language, and even the "language" of graphs is a relatively simple kind of language. More elaborate types of bias specification languages are typically found in the field of ►inductive logic programming (ILP).

Bias Specification Languages in Inductive Logic Programming

In ILP, the hypotheses returned by the learning algorithm are typically written as first-order logic clauses. As the set of all possible clauses is too large to handle, a subset of these clauses is typically defined; this subset is called the language bias. Several formalisms ("bias specification languages") have been proposed for specifying such subsets. We here focus on a few representative ones.

DLAB

In the DLAB bias specification language (Dehaspe & De Raedt, 1996), the language bias is defined in a declarative way, by defining a syntax that clauses must fulfill. In its simplest form, a DLAB specification simply gives a set of possible head and body literals out of which the system can build a clause.

Example 1 *The actual syntax of the DLAB specification language is relatively complicated (see Dehaspe & De Raedt, 1996), but in essence, one can write down a specification such as:*

```
{ parent({X,Y,Z},{X,Y,Z}),
  grandparent({X,Y,Z},
   {X,Y,Z}) }
:-
{ parent({X,Y,Z},{X,Y,Z}),
  parent({X,Y,Z},{X,Y,Z}),
  grandparent({X,Y,Z},{X,Y,Z}),
  grandparent({X,Y,Z}, {X,Y,Z}) }
```

which states that the hypothesis language consists of all clauses that have at most one `parent` *and at most one*

`grandparent` *literal in the head, and at most two of these literals in the body; the arguments of these literals may be variables* `X`, `Y`, `Z`. *Thus, the following clauses are in the hypothesis language:*

```
grandparent(X, Y) :- parent(X, Z),
  parent(Z,Y)
:- parent(X,Y), parent(Y,X)
:- parent(X,X)
```

These express the usual definition of grandparent as well as the fact that there can be no cycles in the parent relation.

Note that for each argument of each literal, all the variables and constants that may occur have to be enumerated explicitly. This can make a DLAB specification quite complex. While DLAB contains advanced constructs to alleviate this problem, it remains the case that often very elaborate bias specifications are needed in practical situations.

Type- and Mode-Based Biases

A more flexible bias specification language is used by Progol (Muggleton, 1995) and many other ILP systems. It is based on the notions of types and modes. In Progol, arguments of a predicate can be typed, and a variable can never occur in two locations with different types. Similarly, arguments of a predicate have an input (+) or output (−) mode; each variable that occurs as an input argument of some literal must occur elsewhere as an output argument, or must occur as input argument in the head literal of a clause.

Example 2 *In Progol, the specifications*

```
type(parent(human,human)).
type(grandparent(human,human)).
modeh(grandparent(+,+)).
  % modeh: specifies a head literal
modeb(grandparent(+,-)).
  % modeb: specifies a body literal
modeb(parent(+,-)).
```

allow the system to construct a clause such as

```
grandparent(X,Y) :- parent(X,Z),
  parent(Z,Y)
```

but not the following clause:

```
grandparent(X,Y) :- parent(Z,Y)
```

because `Z` *occurs as an input parameter for* `parent` *without occurring elsewhere as an output parameter (i.e., it is being used without having been given a value first).*

FLIPPER's Bias Specification Language

The FLIPPER system (Cohen, 1996) employs a powerful, but somewhat more procedural, bias specification formalism. The user does not specify a set of valid hypotheses directly, but rather, specifies a ▶Refinement Operator. The language bias is the set of all clauses that can be obtained from one or more starting clauses through repeated application of this refinement operator. The operator itself is defined by specifying under which conditions certain literals can be added to a clause.

Rules defining the operator have one of the following forms:

$A \leftarrow B$ `where` *Pre* `asserting` *Post*
L `where` *Pre* `asserting` *Post*

The first form defines a set of "starting clauses," and the second form defines when a literal L can be added to a clause. Each rule can only be applied when its preconditions *Pre* are fulfilled, and upon application will assert a set of literals *Post*. As an example (Cohen, 1996), the rules

$$\begin{aligned} &\mathit{illegal}(A,B,C,D,E,F) \leftarrow \\ &\quad \texttt{where}\ \mathit{true} \\ &\quad \texttt{asserting}\ \{\mathit{linked}(A), \mathit{linked}(B), \ldots, \\ &\qquad \mathit{linked}(F)\} \end{aligned}$$

$$R(X,Y)\ \texttt{where}\ \mathit{rel}(R), \mathit{linked}(X), \mathit{linked}(Y)\ \texttt{asserting}\ \varnothing$$

state that any clause of the form

$$\mathit{illegal}(A,B,C,D,E,F) \leftarrow$$

can be used as a starting point for the refinement operator, and the variables in this clause are all *linked*. Further, any literal of the form $R(X,Y)$ with R a relation

symbol (as defined by the *Rel* predicate) and *X* and *Y* linked variables can be added.

Other Approaches

Grammars or term rewriting systems have been proposed several times as a means of defining the hypothesis language. A relatively recent approach along these lines was given by Lloyd, who uses a rewriting system to define the tests that can occur in the nodes of a decision tree built by the Alkemy system (Lloyd, 2003).

Boström & Idestam-Almquist (1999) present an approach where the language bias is implicitly defined through the ▶Background Knowledge given to the learner.

Knobbe et al. (2000) propose the use of UML as a "common" bias specification language, specifications in which could be translated automatically to languages specific to a particular learner.

Further Reading

An overview of bias specification formalisms in ILP is given by Nédellec et al. (1996). The bias specification languages discussed above are discussed in more detail in Dehaspe and De Raedt (1996), Muggleton (1995), and Cohen (1996). De Raedt (1992) discusses language bias and the concept of bias shift (learners weakening their bias, i.e., extending the set of hypotheses that can be represented, when a given language bias turns out to be too restrictive). A more recent approach to learning declarative bias is presented by Bridewell and Todorovski (2008).

Cross References

▶Hypothesis Language
▶Inductive Logic Programminllg

Recommended Reading

Boström, H., & Idestam-Almquist, P. (1999). Induction of logic programs by example-guided unfolding. *Journal of Logic Programming, 40*(2–3), 159–183.

Bridewell, W., & Todorovski, L. (2008). Learning declarative bias. In *Proceedings of the 17th international conference on inductive logic programming. Lecture notes in computer science* (Vol. 4894, pp. 63–77). Berlin: Springer.

Cohen, W. (1996). Learning to classify English text with ILP methods. In L. De Raedt (Ed.), *Advances in inductive logic programming* (pp. 124–143). Amsterdam: IOS Press.

De Raedt, L. (1992). *Interactive theory revision: An inductive logic programming approach*. New York: Academic Press.

Dehaspe, L., & De Raedt, L. (1996). DLAB: A declarative language bias formalism. In *Proceedings of the international symposium on methodologies for intelligent systems. Lecture notes in artificial intelligence* (Vol. 1079, pp. 613–622). Berlin: Springer.

Knobbe, A. J., Siebes, A., Blockeel, H., & van der Wallen, D. (2000). Multi-relational data mining, using UML for ILP. In *Proceedings of PKDD-2000 – The fourth European conference on principles and practice of knowledge discovery in databases. Lecture notes in artificial intelligence* (Vol. 1910, pp. 1–12), Lyon, France. Berlin: Springer.

Lloyd, J. W. (2003). *Logic for learning*. Berlin: Springer.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, Special Issue on Inductive Logic Programming, 13*(3–4), 245–286.

Nédellec, C., Adé, H., Bergadano, F., & Tausend, B. (1996). Declarative bias in ILP. In L. De Raedt (Ed.), *Advances in inductive logic programming. Frontiers in artificial intelligence and applications* (Vol. 32, pp. 82–103). Amsterdam: IOS Press.

Bias Variance Decomposition

Definition

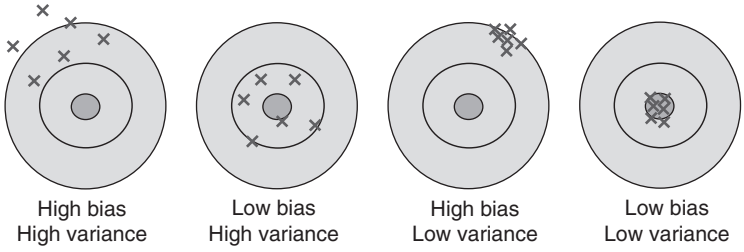
The bias-variance decomposition is a useful theoretical tool to understand the performance characteristics of a learning algorithm. The following discussion is restricted to the use of *squared loss* as the performance measure, although similar analyses have been undertaken for other loss functions. The case receiving most attention is the zero-one loss (i.e., classification problems), in which case the decomposition is nonunique and a topic of active research. See Domingos (1992) for details.

The decomposition allows us to see that the mean squared error of a model (generated by a particular learning algorithm) is in fact made up of two components. The *bias* component tells us how accurate the model is, on average across different possible training sets. The *variance* component tells us how sensitive the learning algorithm is to small changes in the training set (Fig. 1).

Mathematically, this can be quantified as a decomposition of the mean squared error function. For a testing example $\{\mathbf{x}, d\}$, the decomposition is:

$$\mathcal{E}_{\mathcal{D}}\{(f(\mathbf{x}) - d)^2\} = (\mathcal{E}_{\mathcal{D}}\{f(\mathbf{x})\} - d)^2 + \mathcal{E}_{\mathcal{D}}\{(f(\mathbf{x}) - \mathcal{E}_{\mathcal{D}}\{f(\mathbf{x})\})^2\},$$

$$\text{MSE} = \text{bias}^2 + \text{variance},$$



Bias Variance Decomposition. Figure 1. The bias-variance decomposition is like trying to hit the bullseye on a dartboard. Each dart is thrown after training our "dart-throwing" model in a slightly different manner. If the darts vary wildly, the learner is *high variance*. If they are far from the bullseye, the learner is *high bias*. The ideal is clearly to have both low bias and low variance; however this is often difficult, giving an alternative terminology as the bias-variance "dilemma" (*Dartboard analogy*, Moore & McCabe (2002))

where the expectations are with respect to all possible training sets. In practice, this can be estimated by cross-validation over a single finite training set, enabling a deeper understanding of the algorithm characteristics. For example, efforts to reduce variance often cause increases in bias, and vice versa. A large bias and low variance is an indicator that a learning algorithm is prone to ►overfitting the model.

Cross References

►Bias-Variance Trade-offs: Novel Applications

Recommended Reading

Domingos, P. (1992). A unified bias-variance decomposition for zero-one and squared loss. In *Proceedings of national conference on artificial intelligence*. Austin, TX: AAAI Press.

Geman, S. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*(1)

Moore, D. S., & McCabe, G. P. (2002). Introduction to the practice of statistics. Michelle Julet

Bias-Variance Trade-offs: Novel Applications

Dev Rajnarayan, David Wolpert
NASA Ames Research Center, Moffett Field, CA, USA

Definition

Consider a given random variable $\underline{F}$ and a random variable that we can modify, $\underline{\hat{F}}$. We wish to use a sample of $\underline{\hat{F}}$ as an estimate of a sample of $\underline{F}$. The mean squared error (MSE) between such a pair of samples is a sum of four terms. The first term reflects the statistical coupling between $\underline{F}$ and $\underline{\hat{F}}$ and is conventionally ignored in bias-variance analysis. The second term reflects the inherent noise in $\underline{F}$ and is independent of the estimator $\underline{\hat{F}}$. Accordingly, we cannot affect this term. In contrast, the third and fourth terms depend on $\underline{\hat{F}}$. The third term, called the bias, is independent of the precise samples of both $\underline{F}$ and $\underline{\hat{F}}$, and reflects the difference between the means of $\underline{F}$ and $\underline{\hat{F}}$. The fourth term, called the variance, is independent of the precise sample of $\underline{F}$, and reflects the inherent noise in the estimator as one samples it. These last two terms can be modified by changing the choice of the estimator. In particular, on small sample sets, we can often decrease our mean squared error by, for instance, introducing a small bias that causes a large reduction the variance. While most commonly used in machine learning, this article shows that such bias-variance trade-offs are applicable in a much broader context and in a variety of situations. We also show, using experiments, how existing bias-variance trade-offs can be applied in novel circumstances to improve the performance of a class of optimization algorithms.

Motivation and Background

In its simplest form, the bias-variance decomposition is based on the following idea. Say we have a random variable $\underline{F}$ taking on values F distributed according to a density function $p(F)$. We want to estimate the value of a sample from $p(F)$. To form our estimate, we sample a different random variable $\underline{\hat{F}}$ taking on values $\hat{F}$ distributed according to $p(\hat{F})$. Assuming a quadratic loss function, the quality of our estimate is measured by its MSE:

$$\mathrm{MSE}(\hat{\underline{F}}) \equiv \int p(\hat{F}, F)\,(\hat{F} - F)^2 \mathrm{d}\hat{F}\,\mathrm{d}F.$$

In many situations, $\underline{F}$ and $\hat{\underline{F}}$ are dependent variables. For example, in supervised machine learning, $\underline{F}$ is a "target" conditional distribution, stochastically mapping elements of an input space X into a space Y of output variables. The associated distribution $p(F)$ is the "prior" of $\underline{F}$. A random sample $\mathcal{D}$ of $\underline{F}$, called "the training set," is generated, and $\mathcal{D}$ is used in a "learning algorithm" to produce $\hat{\underline{F}}$, which is our estimate of $\underline{F}$. Clearly, this $\underline{F}$ and $\hat{\underline{F}}$ are statistically dependent, via $\mathcal{D}$. Indeed, intuitively speaking, the goal in designing a learning algorithm is that the $\hat{\underline{F}}$'s it produces are positively correlated with $\underline{F}$'s.

In practice this coupling is simply ignored in analyses of bias plus variance, without any justification (one such justification could be that the coupling has little effect on the value of the MSE). We shall follow that practice here. Accordingly, our equation for MSE reduces to

$$\mathrm{MSE}(\hat{\underline{F}}) = \int p(\hat{F})p(F)\,(\hat{F} - F)^2 \mathrm{d}\hat{F}\,\mathrm{d}F. \qquad (1)$$

If we were to account for the coupling of $\hat{F}$ and $\hat{\underline{F}}$ an additive correction term would need to be added to the right-hand side. For instance, see Wolpert (1997).

Using simple algebra, the right hand side of (1) can be written as the sum of three terms. The first is the variance of $\underline{F}$. Since this is beyond our control in designing the estimator $\hat{\underline{F}}$, we ignore it for the rest of this article. The second term involves a mean that describes the deterministic component of the error. This term depends on both the distribution of $\underline{F}$ and that of $\hat{\underline{F}}$, and quantifies how close the means of those distributions are. The third term is a variance that describes stochastic variations from one sample to the next. This term is independent of the random variable being estimated. Formally, up to an overall additive constant, we can write

$$\begin{aligned}
\mathrm{MSE}(\hat{F}) &= \int p(\hat{F})(\hat{F}^2 - 2F\hat{F} + F^2)\,\mathrm{d}\hat{F} \\
&= \overbrace{\int p(\hat{F})\hat{F}^2\,\mathrm{d}\hat{F} - 2F \int p(\hat{F})\hat{F}\,\mathrm{d}\hat{F}} + F^2 \\
&= \mathbb{V}(\hat{F}) + [\mathbb{E}(\hat{F})]^2 - 2F\,\mathbb{E}(\hat{F}) + F^2 \\
&= \mathbb{V}(\hat{F}) + \underbrace{[F - \mathbb{E}(\hat{F})]^2} \\
&= \text{variance} + \text{bias}^2. \qquad (2)
\end{aligned}$$

In light of (2), one way to try to reduce expected quadratic error is to modify an estimator to trade-off bias and variance. Some of the most famous applications of such bias-variance trade-offs occur in parametric machine learning, where many techniques have been developed to exploit the trade-off. Nonetheless, the trade-off also arises in many other fields, including integral estimation and optimization. In the rest of this paper we present a few novel applications of bias-variance trade-off, and describe some interesting features in each case. A recurring theme is the following: whenever a bias-variance trade-off arises in a particular field, we can use many techniques from parametric machine learning that have been developed for exploiting this trade-off. See Wolpert and Rajnarayan (2007) for further details of many of these applications.

Applications

In this section, we describe some applications of the bias-variance tradeoff. First, we describe Monte Carlo (MC) techniques for the estimation of integrals, and provide a brief analysis of bias-variance trade-offs in this context. Next, we introduce the field of Monte Carlo optimization (MCO), and illustrate that there are more subtleties involved than in simple MC. Then, we describe the field of parametric machine learning, which, as will show, is formally identical to MCO. Finally, we describe the application of parametric learning (PL) techniques to improve the performance of MCO algorithms. We do this in the context of an MCO problem that addresses black-box optimization.

Monte Carlo Estimation of Integrals Using Importance Sampling

Monte Carlo methods are often the method of choice for estimating difficult high-dimensional integrals. Consider a function $f: X \to \mathbb{R}$, which we want to integrate over some region $\mathcal{X} \subseteq X$, yielding the value F, as given by

$$F = \int_{\mathcal{X}} \mathrm{d}x f(x).$$

We can view this as a random variable $\underline{F}$, with density function given by a Dirac delta function centered on F. Therefore, the variance of $\underline{F}$ is 0, and (2) is exact.

A popular MC method to estimate this integral is importance sampling (see Robert & Casella, 2004). This exploits the law of large numbers as follows: i.i.d. samples $x^{(i)}, i = 1, \ldots, m$ are generated from a so-called importance distribution $h(x)$ that we control, and the associated values of the integrand, $f(x^{(i)})$ are computed. Denote these "data" by

$$\mathcal{D} = \{(x^{(i)}, f(x^{(i)}),\ i = 1, \ldots, m\}. \tag{3}$$

Now,

$$\begin{aligned} F &= \int_{\mathcal{X}} \mathrm{d}x\, h(x) \frac{f(x)}{h(x)} \\ &= \lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} \frac{f(x^{(i)})}{h(x^{(i)})} \quad \text{with probability 1.} \end{aligned}$$

Denote by $\underline{\hat{F}}$ the random variable with value given by the sample average for $\mathcal{D}$:

$$\hat{F} = \frac{1}{m} \sum_{i=1}^{m} \frac{f(x^{(i)})}{h(x^{(i)})}.$$

We use $\underline{\hat{F}}$ as our statistical estimator for $\underline{F}$, as we broadly described in the introductory section. Assuming a quadratic loss function, $L(\hat{F}, F) = (F - \hat{F})^2$, the bias-variance decomposition described in (2) applies *exactly*. It can be shown that the estimator $\underline{\hat{F}}$ is unbiased, that is, $\mathbb{E}(\underline{\hat{F}}) = F$, where the mean is over samples of h. Consequently, the MSE of this estimator is just its variance. The choice of sampling distribution h that minimizes this variance is given by (see Robert & Casella, 2004)

$$h^{\star}(x) = \frac{|f(x)|}{\int_{\mathcal{X}} |f(x')|\, \mathrm{d}x'}.$$

By itself, this result is not very helpful, since the equation for the optimal importance distribution contains a similar integral to the one we are trying to estimate. For non-negative integrands $f(x)$, the VEGAS algorithm (Lepage, 1978) describes an adaptive method to find successively better importance distributions, by iteratively estimating $\underline{F}$, and then using that estimate to generate the next importance distribution h. In the case of these unbiased estimators, there is no trade-off between bias and variance, and minimizing MSE is achieved by minimizing variance.

Monte Carlo Optimization

Instead of a *fixed* integral to evaluate, consider a parametrized integral

$$F(\theta) = \int_{\mathcal{X}} \mathrm{d}x f_{\theta}(x).$$

Further, suppose we are interested in finding the value of the parameter $\theta \in \Theta$ that minimizes $F(\theta)$:

$$\theta^{\star} = \arg\min_{\theta\in\Theta} F(\theta).$$

In the case where the functional form of f_{θ} is not explicitly known, one approach to solve this problem is a technique called MCO (see Ermoliev & Norkin, 1998), involving repeated MC estimation of the integral in question with adaptive modification of the parameter θ.

We proceed by analogy to the case with MC. First, we introduce the θ-indexed random variable $\underline{F}(\theta)$, all of whose components have delta-function distributions about the associated values $F(\theta)$. Next, we introduce a θ-indexed vector random variable $\underline{\hat{F}}$ with values

$$\hat{F} \equiv \{\hat{F}(\theta)\ \forall\ \theta \in \Theta\}. \tag{4}$$

Each real-valued component $\underline{\hat{F}}(\theta)$ can be sampled and viewed as an estimate of $\underline{F}(\theta)$.

For example, let $\mathcal{D}$ be a data set as described in (3). Then for every θ, any sample of $\mathcal{D}$ provides an associated estimate

$$\hat{F}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{f_{\theta}(x^{(i)})}{h(x^{(i)})}.$$

That average serves as an estimate of $\underline{F}(\theta)$. Formally, $\underline{\hat{F}}$ is a function of the random variable $\mathcal{D}$, and is given by such averaging over the elements of $\mathcal{D}$. So, a sample of $\mathcal{D}$ provides a sample of $\underline{\hat{F}}$. A priori, we make no restrictions on $\underline{\hat{F}}$, and so, in general, its components may be statistically coupled with one another. Note that this coupling arises even though we are, for simplicity, treating each function $\underline{F}(\theta)$ as having a delta-function distribution, rather than as having a non-zero variance that would reflect our lack of knowledge of the $f(\theta)$ functions.

However $\underline{\hat{F}}$ is defined, given a sample of $\hat{F}$, one way to estimate $\theta^\star$ is

$$\hat{\theta}^\star = \arg\min_{\theta \in \Theta} \hat{F}(\theta).$$

We call this approach "natural" MCO. As an example, say that $\mathcal{D}$ is a set of m samples of h, and let

$$\hat{F}(\theta) \triangleq \frac{1}{m} \sum_{i=1}^{m} \frac{f_\theta(x^{(i)})}{h(x^{(i)})},$$

as above. Under this choice for $\underline{\hat{F}}$,

$$\hat{\theta}^\star = \arg\min_{\theta \in \Theta} \frac{1}{m} \sum_{i=1}^{m} \frac{f_\theta(x^{(i)})}{h(x^{(i)})}. \tag{5}$$

We call this approach "naive" MCO.

Consider *any* algorithm that estimates $\theta^\star$ as a single-valued function of $\hat{F}$. The estimate of $\theta^\star$ produced by that algorithm is itself a random variable, since it is a function of the random variable $\underline{\hat{F}}$. Call this random variable $\underline{\hat{\theta}}^\star$, taking on values $\hat{\theta}^\star$. Any MCO algorithm is defined by $\underline{\hat{\theta}}^\star$; that random variable encapsulates the output estimate made by the algorithm.

To analyze the error of such an algorithm, consider the associated random variable given by the true parametrized integral $F(\underline{\hat{\theta}}^\star)$. The difference between a sample of $F(\underline{\hat{\theta}}^\star)$ and the true minimal value of the integral, $F(\theta^\star) = \min_\theta F(\theta)$, is the error introduced by our estimating that optimal θ as a sample of $\underline{\hat{\theta}}^\star$. Since our aim in MCO is to minimize $F(\theta)$, we adopt the loss function $L(\underline{\hat{\theta}}^\star, \theta^\star) \triangleq F(\underline{\hat{\theta}}^\star) - F(\theta^\star)$. This is in contrast to our discussion on MC integration, which involved quadratic loss. The current loss function just equals $F(\underline{\hat{\theta}}^\star)$ up to an additive constant $F(\theta^\star)$ that is fixed by the MCO problem at hand and is beyond our control. Up to that additive constant, the associated expected loss is

$$\mathbb{E}(L) = \int d\hat{\theta}^\star p(\hat{\theta}^\star) F(\hat{\theta}^\star). \tag{6}$$

Now change coordinates in this integral from the values of the scalar random variable $\underline{\hat{\theta}}^\star$ to the values of the underlying vector random variable $\underline{\hat{F}}$. The expected loss now becomes

$$\mathbb{E}(L) = \int \mathrm{d}\hat{F}\, p(\hat{F}) F(\hat{\theta}^\star(\hat{F})).$$

The natural MCO algorithm provides some insight into these results. For that algorithm,

$$\begin{aligned}\mathbb{E}(L) &= \int \mathrm{d}\hat{F}\, p(\hat{F}) F\left(\arg\min_\theta \hat{F}(\theta)\right) \\ &= \int \mathrm{d}\hat{F}(\theta_1)\, \mathrm{d}\hat{F}(\theta_2) \ldots\ p(\hat{F}(\theta_1), \hat{F}(\theta_2), \ldots) \\ &\quad F\left(\arg\min_\theta \hat{F}(\theta)\right).\end{aligned} \tag{7}$$

For any fixed θ, there is an error between samples of $\hat{F}(\theta)$ and the true value $F(\theta)$. Bias-variance considerations apply to this error, exactly as in the discussion of MC above. We are not, however, concerned with $\hat{F}$ for a single component θ, but rather for a set Θ of θ's.

The simplest such case is where the components of $\hat{F}(\Theta)$ are independent. Even so, $\arg\min_\theta \hat{F}(\theta)$ is distributed according to the laws for extrema of multiple independent random variables, and this distribution depends on higher-order moments of each random variable $\hat{F}(\theta)$. This means that $\mathbb{E}[L]$ also depends on such higher-order moments. Only the first two moments, however, arise in the bias and variance for any single θ. Thus, even in the simplest possible case, the bias-variance considerations for the individual θ do not provide a complete analysis.

In most cases, the components of $\hat{F}$ are *not* independent. Therefore, in order to analyze $\mathbb{E}[L]$, in addition to higher moments of the distribution for each θ, we must now also consider higher-order moments coupling the estimates $\hat{F}(\theta)$ for different θ.

Due to these effects, it may be quite acceptable for all the components $\hat{F}(\theta)$ to have both a large bias and a large variance, as long as they still order the θ's correctly with respect to the true $F(\theta)$. In such a situation, large covariances could ensure that if some $\hat{F}(\theta)$ were incorrectly large, then $\hat{F}(\theta'), \theta' \neq \theta$ would also be incorrectly large. This coupling between the components of $\hat{F}$ would preserve the ordering of θ's under $\underline{F}$. So, even with large bias and variance for each θ, the estimator as a whole would still work well.

Nevertheless, it *is* sufficient to design estimators $\hat{F}(\theta)$ with sufficiently small bias plus variance for each single θ. More precisely, suppose that those terms are very small on the scale of differences $F(\theta) - F(\theta')$ for any θ and θ'. Then by Chebychev's inequality,

we know that the density functions of the random variables $\hat{\underline{F}}(\theta)$ and $\hat{\underline{F}}(\theta')$ have almost no overlap. Accordingly, the probability that a sample of $\hat{\underline{F}}(\theta) - \hat{\underline{F}}(\theta')$ has the opposite sign of $F(\theta) - F(\theta')$ is almost zero.

Evidently, $\mathbb{E}[L]$ is generally determined by a complicated relationship involving bias, variance, covariance, and higher moments. Natural MCO in general, and naive MCO in particular, ignore all of these effects, and consequently, often perform quite poorly in practice. In the next section we discuss some ways of addressing this problem.

Parametric Machine Learning

There are many versions of the basic MCO problem described in the previous section. Some of the best-explored arise in parametric density estimation and parametric supervised learning, which together comprise the field of parametric machine learning (PL).

In particular, parametric supervised learning attempts to solve

$$\arg\min_{\theta\in\Theta} \int \mathrm{d}x\, p(x) \int \mathrm{d}y\, p(y \mid x) f_\theta(x).$$

Here, the values x represent inputs, and the values y represent corresponding outputs, generated according to some stochastic process defined by a set of conditional distributions $\{p(y \mid x),\ x \in \mathcal{X}\}$. Typically, one tries to solve this problem by casting it as an MCO problem. For instance, say we adopt a quadratic loss between a predictor $z_\theta(x)$ and the true value of y. Using MCO notation, we can express the associated supervised learning problem as finding $\arg\min_\theta F(\theta)$, where

$$\begin{aligned} l_\theta(x) &= \int \mathrm{d}y\, p(y \mid x)\, (z_\theta(x) - y)^2, \\ f_\theta(x) &= p(x)\, l_\theta(x), \\ F(\theta) &= \int \mathrm{d}x f_\theta(x). \end{aligned} \tag{8}$$

Next, the argmin is estimated by minimizing a sample-based estimate of the $F(\theta)$'s. More precisely, we are given a "training set" of samples of $p(y \mid x)\, p(x)$, $\{(x^{(i)}, y^i) i = 1, \ldots, m\}$. This training set provides a set of associated estimates of $F(\theta)$:

$$\hat{F}(\theta) = \frac{1}{m} \sum_{i=1}^{m} l_\theta(x^{(i)}).$$

These are used to estimate $\arg\min_\theta F(\theta)$, exactly as in MCO. In particular, one could estimate the minimizer of $F(\theta)$ by finding the minimum of $\hat{F}(\theta)$, just as in natural MCO. As mentioned above, this MCO algorithm can perform very poorly in practice. In PL, this poor performance is called "overfitting the data."

There are several formal approaches that have been explored in PL to try to address this "overfitting the data." Interestingly, none are based on direct consideration of the random variable $F(\hat{\theta}^\star(\hat{\underline{F}}))$ and the ramifications of its distribution for expected loss (cf. (7)). In particular, no work has applied the mathematics of extrema of multiple random variables to analyze the bias-variance-covariance trade-offs encapsulated in (7).

The PL approach that perhaps comes closest to such direct consideration of the distribution of $F(\hat{\underline{\theta}}^\star)$ is uniform convergence theory, which is a central part of computational learning theory (see Angluin, 1992). Uniform convergence theory starts by crudely encapsulating the quadratic loss formula for expected loss under natural MCO (7). It does this by considering the worst-case bound, over possible $p(x)$ and $p(y \mid x)$, of the probability that $F(\underline{\theta}^\star)$ exceeds $\min_\theta F(\theta)$ by more than κ. It then examines how that bound varies with κ. In particular, it relates such variation to characteristics of the set of functions $\{f_\theta : \theta \in \Theta\}$, e.g., the "VC dimension" of that set (see Vapnik, 1982, 1995).

Another, historically earlier approach, is to apply bias-plus-variance considerations to the *entire* PL algorithm $\hat{\underline{\theta}}^\star$, rather than to each $\hat{\underline{F}}(\theta)$ separately. This approach is applicable for algorithms that do not use natural MCO, and even for non-parametric supervised learning. As formulated for parameteric supervised learning, this approach combines the formulas in (8) to write

$$F(\theta) = \int \mathrm{d}x\, \mathrm{d}y\, p(x) p(y \mid x) (z_\theta(x) - y)^2.$$

This is then substituted into (6), giving

$$\begin{aligned} \mathbb{E}[L] &= \int \mathrm{d}\hat{\theta}^\star \mathrm{d}x\, \mathrm{d}y\, p(x)\, p(y \mid x)\, p(\hat{\theta}^\star) (z_{\hat{\theta}^\star}(x) - y)^2 \\ &= \int \mathrm{d}x\, p(x) \Big[\int \mathrm{d}\hat{\theta}^\star \mathrm{d}y\, p(x) p(y \mid x) p(\hat{\theta}^\star) \\ &\quad (z_{\hat{\theta}^\star}(x) - y)^2 \Big]. \end{aligned} \tag{9}$$

The term in square brackets is an x-parameterized expected quadratic loss, which can be decomposed into

a bias, variance, etc., in the usual way. This formulation eliminates any direct concern for issues like the distribution of extrema of multiple random variables, covariances between $\underline{\hat{F}}(\theta)$ and $\underline{\hat{F}}(\theta')$ for different values of θ, and so on.

There are numerous other approaches for addressing the problems of natural MCO that have been explored in PL. Particularly important among these are Bayesian approaches, e.g., Buntine and Weigend (1991), Berger (1985), and Mackay (2003). Based on these approaches, as well as on intuition, many powerful techniques for addressing data-overfitting have been explored in PL, including regularization, cross-validation, stacking, bagging, etc. Essentially all of these techniques can be applied to *any* MCO problem, not just PL problems. Since many of these techniques can be justified using (9), they provide a way to exploit the bias-variance trade-off in other domains besides PL.

PLMCO

In this section, we illustrate how PL techniques that exploit the bias-variance decomposition of (9) can be used to improve an MCO algorithm used in a domain outside of PL. This MCO algorithm is a version of adaptive importance sampling, somewhat similar to the CE method (Rubinstein & Kroese, 2004), and is related to function smoothing on continuous spaces. The PL techniques described are applicable to any other MCO problem, and this particular one is chosen just as an example.

MCO Problem Description The problem is to find the θ-parameterized distribution q_θ that minimizes the associated expected value of a function $G: \mathbb{R}^n \to \mathbb{R}$, i.e., find

$$\arg\min_\theta \mathbb{E}_{q_\theta}[G].$$

We are interested in versions of this problem where we do not know the functional form of G, but can obtain its value $G(x)$ at any $x \in \mathcal{X}$. Similarly we cannot assume that G is smooth, nor can we evaluate its derivatives directly. This scenario arises in many fields, including blackbox optimization (see Wolpert, Strauss, & Rajnarayan, 2006), and risk minimization (see Ermoliev & Norkin, 1998).

We begin by expressing this minimization problem as an MCO problem. We know that

$$\mathbb{E}_{q_\theta}[G] = \int_{\mathcal{X}} dx\, q_\theta(x) G(x)$$

Using MCO terminology, $f_\theta(x) = q_\theta(x)G(x)$ and $F(\theta) = \mathbb{E}_{q_\theta}[G]$. To apply MCO, we must define a vector-valued random variable $\underline{\hat{F}}$ with components indexed by θ, and then use a sample of $\underline{\hat{F}}$ to estimate $\arg\min_\theta \mathbb{E}_{q_\theta}[G]$. In particular, to apply naive MCO to estimate $\arg\min_\theta \mathbb{E}_{q_\theta}(G)$, we first i.i.d. sample a density function $h(x)$. By evaluating the associated values of $G(x)$ we get a data set

$$\begin{aligned}\mathcal{D} &\equiv (\mathcal{D}_{\mathcal{X}}, \mathcal{D}_G) \\ &= (\{x^{(i)} : i = 1, \ldots, m\}, \{G(x^{(i)}) : i = 1, \ldots, m\}).\end{aligned}$$

The associated estimates of $F(\theta)$ for each θ are

$$\hat{F}(\theta) \triangleq \frac{1}{m} \sum_{i=1}^{m} \frac{q_\theta(x^{(i)}) G(x^{(i)})}{h(x^{(i)})}. \tag{10}$$

The associated naive MCO estimate of $\arg\min_\theta \mathbb{E}_{q_\theta}[G]$ is

$$\hat{\theta}^\star \equiv \arg\min_\theta \hat{F}(\theta).$$

Suppose Θ includes all possible density functions over x's. Then the q_θ minimizing our estimate is a delta function about the $x^{(i)} \in \mathcal{D}_{\mathcal{X}}$ with the lowest associated value of $G(x^{(i)})/h(x^{(i)})$. This is clearly a poor estimate in general; it suffers from "data-overfitting." Proceeding as in PL, one way to address this data-overfitting is to use regularization. In particular, we can use the entropic regularizer, given by the negative of the Shannon entropy $S(q_\theta)$. So we now want to find the minimizer of $\mathbb{E}_{q_\theta}[G(x)] - TS(q_\theta)$, where T is the regularization parameter. Equivalently, we can minimize $\beta \mathbb{E}_{q_\theta}[G(x)] - S(q_\theta)$, where $\beta = 1/T$. This changes the definition of $\hat{F}$ from the function given in (10) to

$$\hat{F}(\theta) \triangleq \frac{1}{m} \sum_{i=1}^{m} \frac{\beta\, q_\theta(x^{(i)}) G(x^{(i)})}{h(x^{(i)})} - S(q_\theta).$$

Solution Methodology Unfortunately, it can be difficult to find the θ globally minimizing this new $\hat{F}$ for an arbitrary $\mathcal{D}$. An alternative is to find a close approximation

to that optimal θ. One way to do this is as follows. First, we find minimizer of

$$\frac{1}{m}\sum_{i=1}^{m}\frac{\beta\, p(x^{(i)})G(x^{(i)})}{h(x^{(i)})} - S(p) \tag{11}$$

over the set of *all* possible distributions $p(x)$ with domain $\mathcal{X}$. We then find the q_θ that has minimal Kullback–Leibler (KL) divergence from this p, evaluated over $\mathcal{D}_\mathcal{X}$. That serves as our approximation to $\arg\min_\theta \hat{F}(\theta)$, and therefore as our estimate of the θ that minimizes $\mathbb{E}_{q_\theta}(G)$.

The minimizer p of (11) can be found in closed form; over $\mathcal{D}_\mathcal{X}$ it is the Boltzmann distribution $p^\beta(x^{(i)}) \propto \exp(-\beta\, G(x^{(i)}))$. The KL divergence in $\mathcal{D}_\mathcal{X}$ from this Boltzmann distribution to q_θ is

$$F(\theta) = \mathrm{KL}(p^\beta \| q_\theta) = \int_\mathcal{X} \mathrm{d}x\, p^\beta(x) \log\left(\frac{p^\beta(x)}{q_\theta(x)}\right).$$

The minimizer of this KL divergence is given by

$$\theta^\dagger = \arg\min_\theta -\sum_{i=1}^{m}\frac{\exp(-\beta G(x^{(i)}))}{h(x^{(i)})}\log(q_\theta(x^{(i)})). \tag{12}$$

This approach is an approximation to a regularized version of the naive MCO estimate of the θ that minimizes $\mathbb{E}_{q_\theta}(G)$. The application of the technique of regularization in this context has the same motivation as it does in PL: to reduce bias plus variance.

Log-Concave Densities If q_θ is log-concave in its parameters θ, then the minimization problem in (12) is a convex optimization problem, and the optimal parameters can be found closed-form. Denote the likelihood ratios by $s^{(i)} = \exp(-\beta G(x^{(i)}))/h(x^{(i)})$. Differentiating (12) with respect to the parameters μ and Σ^{-1} and setting them to zero yields

$$\mu^\star = \frac{\sum_\mathcal{D} s^{(i)} x^{(i)}}{\sum_\mathcal{D} s^{(i)}}$$
$$\Sigma^\star = \frac{\sum_\mathcal{D} s^{(i)}(x^{(i)} - \mu^\star)(x^{(i)} - \mu^\star)^T}{\sum_\mathcal{D} s^{(i)}}$$

Mixture Models The single Gaussian is a fairly restrictive class of models. Mixture models (see ▶Mixture Modeling) can significantly improve flexibility, but at the cost of convexity of the KL distance minimization problem. However, a plethora of techniques from supervised learning, in particular the expectation maximization (EM) algorithm, can be applied with minor modifications.

Suppose q_θ is a mixture of M Gaussians, that is, $\theta = (\mu, \Sigma, \phi)$ where ϕ is the mixing p.m.f, we can view the problem as one where a hidden variable z decides which mixture component each sample is drawn from. We then have the optimization problem

$$\text{minimize } -\sum_\mathcal{D}\frac{p(x^{(i)})}{h(x^{(i)})}\log\left(q_\theta(x^{(i)}, z^{(i)})\right).$$

Following the standard EM procedure, we get the algorithm described in (13). Since this is a nonconvex problem, one typically runs the algorithm multiple times with random initializations of the parameters.

E-step: For each i, set $Q_i(z^{(i)}) = p(z^{(i)}|x^{(i)})$,
that is, $w_j^{(i)} = q_{\mu,\Sigma,\phi}(z^{(i)} = j|x^{(i)}),\quad j = 1,\ldots,M.$

M-step: Set
$$\mu_j = \frac{\sum_\mathcal{D} w_j^{(i)} s^{(i)}\, x^{(i)}}{\sum_\mathcal{D} w_j^{(i)} s^{(i)}},$$
$$\Sigma_j = \frac{\sum_\mathcal{D} w_j^{(i)} s^{(i)}\,(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_\mathcal{D} w_j^{(i)} s^{(i)}},$$
$$\phi_j = \frac{\sum_\mathcal{D} w_j^{(i)} s^{(i)}}{\sum_\mathcal{D} s^{(i)}}.$$

Test Problems To compare the performance of this algorithm with and without the use of PL techniques, we use a couple of very simple academic problems in two and four dimensions – the Rosenbrock function in two dimensions, given by

$$G_R(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

and the Woods function in four dimensions, given by given by

$$\begin{aligned} G_{\text{Woods}}(x) = {} & 100(x_2 - x_1)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 \\ & + (1 - x_3)^2 \\ & + 10.1[(1 - x_2)^2 + (1 - x_4)^2] \\ & + 19.8(1 - x_2)(1 - x_4). \end{aligned}$$

For the Rosenbrock, the optimum value of 0 is achieved at $x = (1,1)$, and for the Woods problem, the optimum value of 0 is achieved at $x = (1,1,1,1)$.

Application of PL Techniques As mentioned above, there are many PL techniques beyond regularization that are designed to optimize the trade-off between bias and variance. So having cast the solution of $\arg\min_{q_\theta} \mathbb{E}(G)$ as an MCO problem, we can apply those other PL techniques instead of (or in addition to) entropic regularization. This should improve the performance of our MCO algorithm, for the exact same reason that using those techniques to trade off bias and variance improves performance in PL. We briefly mention some of those alternative techniques here.

The overall MCO algorithm is broadly described in Algorithm 1. For the Woods problem, 20 samples of x are drawn from the updated q_θ at each iteration, and for the Rosenbrock, 10 samples. For comparing various methods and plotting purposes, 1,000 samples of $G(x)$ are drawn to evaluate $\mathbb{E}_{q_\theta}[G(x)]$. Note: in an actual optimization, we will not be drawing these test samples! All the performance results in Fig. 1 are based on 50 runs of the PC algorithm, randomly initialized each time. The sample mean performance across these runs is plotted along with 95% confidence intervals for this sample mean (shaded regions).

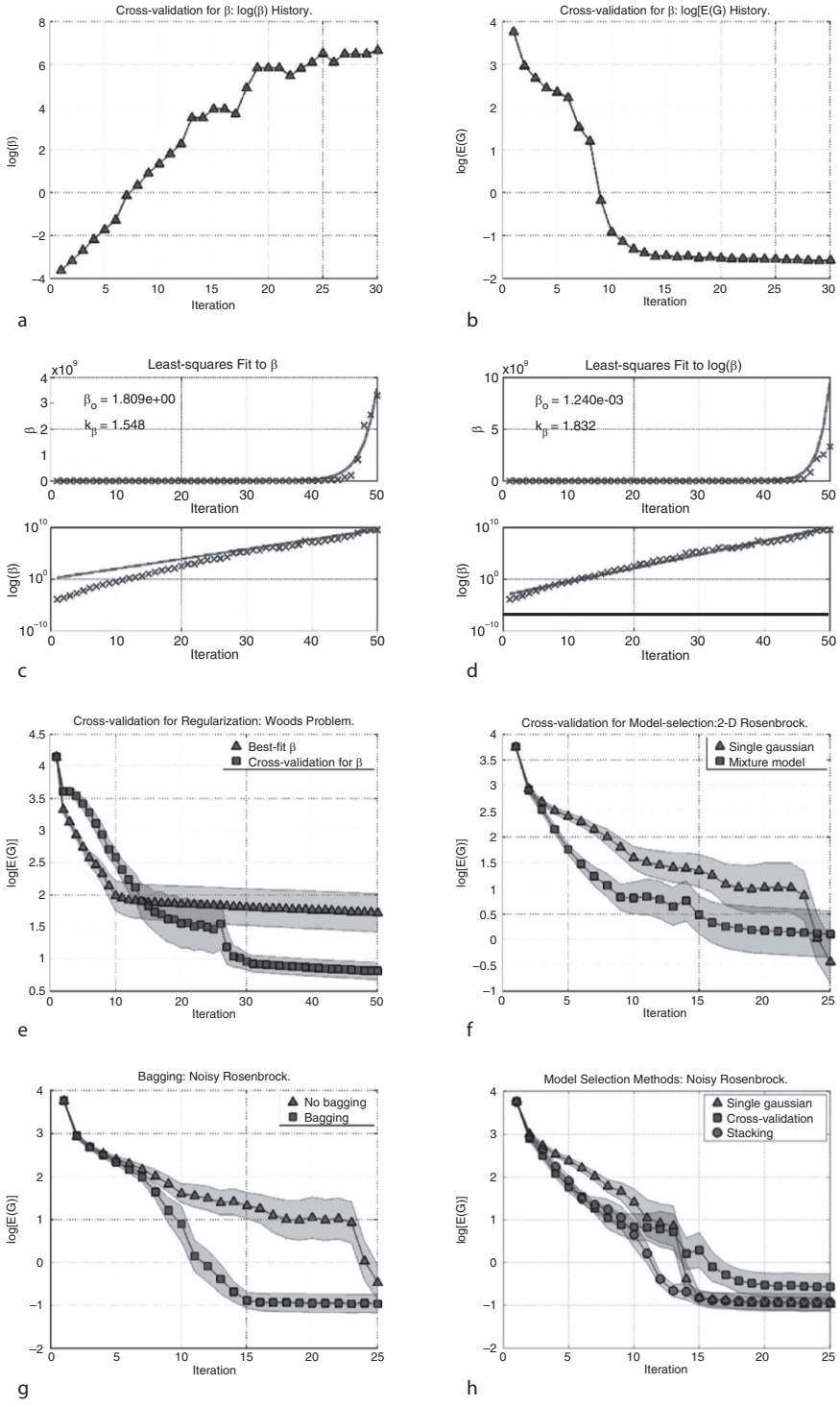
►*Cross-Validation for Regularization:* We note that we are using regularization to reduce variance, but that regularization introduces bias. As is done in PL, we use standard k-fold cross-validation to tradeoff this bias and variance. We do this by partitioning the data into k disjoint sets. The held-out data for the ith fold is just the ith partition, and the held-in data is the union of all other partitions. First, we "train" the regularized algorithm on the held-in data $\mathcal{D}_t$ to get an optimal set of parameters $\theta^\star$, then "test" this $\theta^\star$ by considering unregularized performance on the held-out data $\mathcal{D}_v$. In our context, "training" refers to finding optimal parameters by KL distance minimization using the held-in data, and "testing" refers to estimating $\mathbb{E}_{q_\theta}[G(x)]$ on the held-out data using the following formula (Robert & Casella, 2004).

$$\widehat{g}(\theta) = \frac{\sum_{\mathcal{D}_v} \frac{q_\theta(x^{(i)})G(x^{(i)})}{h(x^{(i)})}}{\sum_{\mathcal{D}_v} \frac{q_\theta(x^{(i)})}{h(x^{(i)})}}.$$

We do this for several values of the regularization parameter β in the interval $k_1\beta < \beta < k_2\beta$, and choose the one that yield the best held-out performance, averaged over all folds. For our experiments, $k_1 = 0.5, k_2 = 3$, and we use five equally-spaced values in this interval. Having found the best regularization parameter in this range, we then use *all* the data to minimize KL distance using this optimal value of β. Note that all cross-validation is done *without* any additional evaluations of $G(x)$. Cross-validation for β in PC is similar to optimizing the annealing schedule in simulated annealing. This "auto-annealing" is seen in Fig. 1a, which shows the variation of β with iterations of the Rosenbrock problem. It can be seen that β value sometimes decreases from one iteration to the next. This can never happen in any kind of "geometric annealing schedule," $\beta \leftarrow k_\beta\beta$, $k_\beta > 1$, of the sort that is often used in most algorithms in the literature. In fact, we ran 50 trials of this algorithm on the Rosenbrock and then computed a best-fit geometric variation for β, that is, a nonlinear least squares fit to variation of β, and a linear least squares fit to the variation of $\log(\beta)$. These are shown in Fig. 1c and d. As can be seen, neither is a very good fit. We then ran 50 trials of the algorithm with the fixed update rule obtained by best-fit to $\log(\beta)$, and found that the adaptive setting of β using cross-validation performed an order of magnitude better, as shown in Fig. 1e.

Algorithm 1 Overview of pq minimization using Gaussian mixtures

1: Draw uniform random samples on X
2: Initialize regularization parameter β
3: Compute $G(x)$ values for those samples
4: **repeat**
5: Find a mixture distribution q_θ to minimize sampled pq KL distance
6: Sample from q_θ
7: Compute $G(x)$ for those samples
8: Update β
9: **until** Termination
10: Sample final q_θ to get solution(s).



Bias-Variance Trade-offs: Novel Applications. Figure 1. Various PL techniques improve MCO performance

Cross-Validation for Model Selection: Given a set Θ (sometimes called a model class) to choose θ from, we can find an optimal $\theta \in \Theta$. But how do we choose the set Θ? In PL, this is done using cross-validation. We choose that set Θ such that $\arg\min_{\theta\in\Theta} \hat{F}(\theta)$ has the best held-out performance. As before, we use that model class Θ that yields the lowest estimate of $\mathbb{E}_{q_\theta}[G(x)]$ on the held-out data. We demonstrate the use of this PL technique for minimizing the Rosenbrock problem, which has a long curved valley that is poorly approximated by a single Gaussian. We use cross-validation to choose between a Gaussian mixture with up to four components. The improvement in performance is shown in Fig. 1d.

Bagging: In bagging (Breiman, 1996a), we generate multiple data sets by resampling the given data set with replacement. These new data sets will, in general, contain replicates. We "train" the learning algorithm on each of these resampled data sets, and average the results. In our case, we average the q_θ got by our KL divergence minimization on each data set. PC works even on stochastic objective functions, and on the noisy Rosenbrock, we implemented PC with bagging by resampling ten times, and obtained significant performance gains, as seen in Fig. 1g.

Stacking: In bagging, we combine estimates of the same learning algorithm on different data sets generated by resampling, whereas in stacking (Breiman, 1996b; Smyth & Wolpert, 1999), we combine estimates of different learning algorithms on the same data set. These combined estimated are often better than any of the single estimates. In our case, we combine the q_θ obtained from our KL divergence minimization algorithm using multiple models Θ. Again, Fig. 1h shows that cross-validation for model selection performs better than a single model, and stacking performs slightly better than cross-validation.

Conclusions

The conventional goal of reducing bias plus variance has interesting applications in a variety of fields. In straightforward applications, the bias-variance trade-offs can decrease the MSE of estimators, reduce the generalization error of learning algorithms, and so on. In this article, we described a novel application of bias-variance trade-offs: we placed bias-variance trade-offs in the context of MCO, and discussed the need for higher moments in the trade-off, such as a bias-variance-covariance trade-off. We also showed a way of applying just a bias-variance trade-off, as used in Parametric Learning, to improve the performance of MCO algorithms.

Recommended Reading

Angluin, D. (1992). Computational learning theory: Survey and selected bibliography. In *Proceedings of the twenty-fourth annual ACM symposium on theory of computing*. New York: ACM.

Berger, J. O. (1985). *Statistical decision theory and bayesian analysis*. New York: Springer.

Breiman, L. (1996a). Bagging predictors. *Machine Learning, 24*(2), 123–140.

Breiman, L. (1996b). Stacked regression. *Machine Learning, 24*(1), 49–64.

Buntine, W., & Weigend, A. (1991). Bayesian back-propagation. *Complex Systems, 5*, 603–643.

Ermoliev, Y. M., & Norkin, V. I. (1998). *Monte carlo optimization and path dependent nonstationary laws of large numbers*. Technical Report IR-98-009. International Institute for Applied Systems Analysis, Austria.

Lepage, G. P. (1978). A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics, 27*, 192–203.

Mackay, D. (2003). *Information theory, inference, and learning algorithms*. Cambridge, UK: Cambridge University Press.

Robert, C. P., & Casella, G. (2004). *Monte Carlo statistical methods*. New York: Springer.

Rubinstein, R., & Kroese, D. (2004). *The cross-entropy method*. New York: Springer.

Smyth, P., & Wolpert, D. (1999). Linearly combining density estimators via stacking. *Machine Learning, 36*(1–2), 59–83.

Vapnik, V. N. (1982). *Estimation of dependences based on empirical data*. New York: Springer.

Vapnik, V. N. (1995). *The nature of statistical learning theory*. New York: Springer.

Wolpert, D. H. (1997). On bias plus variance. *Neural Computation, 9*, 1211–1244.

Wolpert, D. H., & Rajnarayan, D. (2007). Parametric learning and monte carlo optimization. arXiv:0704.1274v1 [cs.LG].

Wolpert, D. H., Strauss, C. E. M., & Rajnarayan, D. (2006). Advances in distributed optimization using probability collectives. *Advances in Complex Systems, 9*(4), 383–436.

Bias-Variance Trade-offs

►Bias-Variance

Bias-Variance-Covariance Decomposition

The bias-variance-covarianc delcomposition is a theoretical result underlying ►ensemble learning algorithms. It is an extension of the ►bias-variance decomposition, for linear combinations of models. The expected squared error of the ensemble $\bar{f}(\mathbf{x})$ from a target d is:

$$\mathcal{E}_{\mathcal{D}}\{(\bar{f}(\mathbf{x}) - d)^2\} = \overline{\text{bias}}^2 + \frac{1}{T}\overline{\text{var}} + \left(1 - \frac{1}{T}\right)\overline{\text{covar}}.$$

The error is composed of the average bias of the models, plus a term involving their average variance, and a final term involving their average *pairwise covariance*. This shows that while a single model has a two-way bias-variance tradeoff, an ensemble is controlled by a three-way tradeoff. This ensemble tradeoff is often referred to as the *accuracy-diversity* dilemma for an ensemble. See ►ensemble learning for more details.

Bilingual Lexicon Extraction

Bilingual lexicon extraction is the task of automatically identifying a terms in a first language and terms in a second language which are translation f one another. In this context, a term can be either a single word or an expression composed of several words the full meaning of which cannot be derived compositionally from the meaning of the individual words. Bilingual lexicon extraction is itself a form of ►cross-lingual text mining and is an essential preliminary step in many approaches for performing other ►cross-lingual text mining tasks.

Binning

►Discretization

Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Wulfram Gerstner
Brain Mind Institute, Lausanne EPFL, Switzerland

Synonyms

Correlation-based learning; Hebb rule; Hebbian learning

Definition

The brain of humans and animals consists of a large number of interconnected neurons. Learning in biological neural systems is thought to take place by changes in the connections between these neurons. Since the contact points between two neurons are called synapses, the change in the connection strength is called synaptic plasticity. The mathematical description of synaptic plasticity is called a (biological) learning rule. Most of these biological learning rules can be categorized in the context of machine learning as unsupervised learning rules, and the remaining ones as reward-based or reinforcement learning. The Hebb rule is an example of an unsupervised correlation-based learning rule formulated on the level of neuronal firing rates. Spike-timing-dependent plasticity (STDP) is an unsupervised learning rule formulated on the level of spikes. Modulation of learning rates in a Hebb rule or STDP rule by a diffusive signal carrying reward-related information yields a biologically plausible form of a reinforcement learning rule.

Motivation and Background

Humans and animals can adapt to environmental conditions and learn new tasks. Learning becomes measurable by changes in the behavior: humans and animals get better at seeing and distinguishing visual objects with experience; animals can learn to go to a target location; humans can memorize a list of words and recall the items 2 days later. How learning is implemented in the biological substrate is only partially known.

The brain consists of billions of neurons. Each neuron has long wire-like extensions and makes contacts with thousands of other neurons. This network of neurons is not fixed but constantly changes. Connections

can be formed or can disappear, and existing connections can be strengthened or weakened. Neuroscientists have shown in numerous experiments that changes can be induced by stimulating neuronal activity in an appropriate fashion. Moreover, changes in synaptic connections that have been induced in one or a few seconds can persist for hours or days, an effect called long-term potentiation (LTP) or long-term depression (LTD) of synapses.

The question arises of whether such long-lasting changes in connections are useful for learning. To answer this question, research in theoretical and computational neuroscience needs to solve two problems: First, develop a compact but realistic description of the phenomenon of synaptic plasticity observed in biology, i.e., extract learning rules from the biological data; and second, study the functional consequences of these learning rules. An important insight from experiments on LTP is that the activation of a synaptic connection alone does not lead to a long-lasting change; however, if the activation of the synapses by presynaptic signals is combined with some activation of the postsynaptic neuron, then a long-lasting change of the synapse may occur. The coactivation of presynaptic and postsynaptic neurons as a condition for learning is the key ingredient of Hebbian learning rules. Here, activation of the presynaptic neuron means that it fires one or several action potentials; activation of the postsynaptic neuron can be represented by high firing rates, a few well-timed action potentials or input from other neurons that lead to an increase in the membrane voltage.

Structure of the Learning System

The Hebb Rule

Hebbian learning rules are local, i.e., they depend only on the state of the presynaptic and postsynaptic neurons plus possibly the current value of the synaptic weight itself. Let w_{ij} denotes the weight between a presynaptic neuron j and a postsynaptic neuron i, and let us describe the activity (e.g., the firing rate) of each neuron by a continuous variable v_j and v_i, respectively. Mathematically, we may therefore write for a local learning rule

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = F(w_{ij}; v_i, v_j) \tag{1}$$

where F is an unknown function. In addition to locality, Hebbian learning requires some kind of cooperation or correlation between the activity of the presynaptic neuron and that of the postsynaptic neuron. At the moment we restrict ourselves to the requirement of *simultaneous activity* of presynaptic and postsynaptic neurons. Since F is a function of the rates v_i and v_j, we may expand F about $v_i = v_j = 0$. An expansion to second order of the rates yields

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \approx\ & c_0(w_{ij}) + c_1^{\mathrm{pre}}(w_{ij})\, v_j + c_1^{\mathrm{post}}(w_{ij}) v_i \\ & + c_2^{\mathrm{corr}}(w_{ij})\, v_i\, v_j + c_2^{\mathrm{post}}(w_{ij})\, v_i^2 \\ & + c_2^{\mathrm{pre}}(w_{ij})\, v_j^2 + O(v^3).\end{aligned} \tag{2}$$

Here, v_i and v_j are functions of time, i.e., $v_i(t)$ and $v_j(t)$ and so is the weight w_{ij}. The bilinear term $v_i(t)\, v_j(t)$ is sensitive to the instantaneous *correlations* between presynaptic and postsynaptic activities. It is this term that makes Hebbian learning a useful concept. The simplest implementation of Hebbian plasticity would be to require $c_2^{\mathrm{corr}} > 0$ and set all other parameters in the expansion (2) to zero

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = c_2^{\mathrm{corr}}(w_{ij})\, v_i\, v_j\,. \tag{3}$$

Equation (3) with fixed parameter $c_2^{\mathrm{corr}} > 0$ is the prototype of Hebbian learning. However, since the activity variables v_i and v_j are always positive, such a rule will lead eventually to an increase of all weights in a network. Hence, some of the other terms (e.g., c_0 or c_1^{pre}) need to have a negative coefficient to make Hebbian learning stable. In passing we note that a learning rule with $c_2^{\mathrm{corr}} < 0$ is usually called anti-Hebbian.

Oja's rule. A particular interesting case is a model with coefficients $c_2^{\mathrm{corr}} > 0$ and $c_2^{\mathrm{post}} < 0$, since it guarantees the normalization of the set of weights $w_{i1}, \ldots w_{iN}$ converging onto the same postsynaptic neuron i.

BCM rule. The Bienenstock–Cooper–Munro learning rule (also called BCM rule) with

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = a(w_{ij})\Phi(v_i - \vartheta)\, v_j \tag{4}$$

where Φ is some nonlinear function with $\Phi(0) = 0$ is a special case of (1). The parameter ϑ depends on the average firing rate.

Temporally asymmetric Hebbian learning. In the Taylor expansion (2) we focused on *instantaneous* correlations. More generally, we can use a Volterra expansion so as to also include temporal correlations with

nonzero time lag. With the additional assumptions that changes are instantaneous, a Volterra expansion generates terms of the form

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} \propto \int_0^\infty [W_+(s) v_i(t)\, v_j(t-s) + W_-(s) v_j(t)\, v_i(t-s)]\mathrm{d}s \qquad (5)$$

with some functions W_+ and W_-. For reasons of causality, W_+ and W_- must vanish for $s < 0$. Since $W_+(s) \neq W_-(s)$, learning is asymmetric in time so that learning rules of the form (5) are called temporally asymmetric Hebbian learning. In the special case $W_+(s) = -W_-(s)$, we have antisymmetric Hebbian learning. The functions W_+ and W_- may depend on the present weight value.

STDP rule. STDP is a form of Hebbian learning with increased temporal resolution. In contrast to rate-based Hebb models, neuronal activity is described by the firing times of the neuron, i.e., the moments when the presynaptic and postsynaptic neurons emit action potentials. Let t_j^f denote the fth spike of the presynaptic neuron j and t_i^n the nth spike of the postsynaptic neuron i. The weight change in an STDP rule depends on the exact timing of presynaptic and postsynaptic spikes

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = \sum_n \sum_f [A(w_{ij}; t - t_j^f)\delta(t - t_i^n) + B(w_{ij}; t - t_i^f)\delta(t - t_j^f)] \qquad (6)$$

where $A(x)$ and $B(x)$ are some real-valued functions with $A(w_{ij}, x) = B(w_{ij}, x) = 0$ for $x < 0$. Thus, at the moment of a postsynaptic spike the synaptic weight is updated by an amount that depends on the time $t_i^f - t_j^f$ since a previous presynaptic spike t_j^f. Similarly, at the moment of a presynaptic spike the synaptic weight is updated by an amount that depends on the time $t_j^f - t_i^f$ since a previous postsynaptic spike t_i^f. The dependence on the present value w_{ij} can be used to keep the weight in a desired range $0 < w_{ij} < w^{\max}$. A standard choice for the functions A and B is $A(w_{ij}); t - t_j^f = A_+(w_{ij}) \exp[-(t - t_j^f)/\tau_+]$ for $t - t_j^f > 0$ and zero otherwise. Similarly, $B(w_{ij}; t - t_i^n) = B_-(w_{ij}) \exp[-(t - t_i^n)/\tau_-]$ for $t - t_i^f > 0$ and zero otherwise. Here, τ_+ and τ_- are time constants in the range of 10–50 ms. The case $A_+(x) = (w^{\max} - x)\, c_+$ and $B_x(x) = -\, c_- x$ is called soft bounds. The choice $A_+(x) = c_+ \Theta(w^{\max} - x)$ and $B_x = -\, c_- \Theta(x)$ is called hard bounds. Here, c_+ and c_- are positive constants. The term proportional to A_+ causes potentiation (weight increase), the one proportional to A_- causes depression (weight decrease) of synapses. Note that the STDP rule (6) can be interpreted as a spike-based form of temporally asymmetric Hebbian learning.

Functional Consequences of Hebbian Learning

Sensitivity to correlations. All Hebbian learning rules are sensitive to the correlations between the activity of the presynaptic neuron j and that of the postsynaptic neuron i. If the activity of the postsynaptic neuron is given by a linear sum of all inputs rates, i.e., $v_i = \gamma \sum_j w_{ij} v_j$, then correlations between presynaptic and postsynaptic activities can be traced back to correlations in the input. A particular clear example of learning driven by correlations in the input is Oja's learning rule applied to a statistical ensemble of inputs with zero mean. In this case, the postsynaptic neuron becomes sensitive to the dominant principal component of the input ensemble. If the neuron model is nonlinear, Hebbian learning extracts the independent components of the statistical input ensemble. These two examples show that learning by a Hebbian learning rule makes neurons adapt to the statistics of the input. While the condition of zero-mean input is biologically not realistic (because neuronal firing rates are always positive), this condition can be relaxed so that the same result is also applicable to biologically plausible learning rules.

Receptive fields and cortical maps. Neurons in the primary visual cortex of cats and monkeys respond to visual stimuli in a localized region of the visual field. This small sensitive zone is called the receptive field of the neuron. Neighboring neurons normally have very similar receptive fields. The exact location and properties of the receptive field are not fixed, but can be influenced by sensory stimulation. Models of unsupervised Hebbian learning can explain the development of receptive fields and the adaptation of cortical maps to the statistics of the ensemble of stimuli.

Beyond the Hebb rule. Standard models of Hebbian learning are formulated on the level of neuronal firing rates, a graded variable characterizing neuronal activity. However, real neurons communicate by spikes, short electrical pulses or "action potentials" with a rather

stereotyped time course. Experiments have shown that the changes of synaptic efficacy depend not only on the mean firing rate of action potentials but on the relative timing of presynaptic and postsynaptic spikes on the level of milliseconds. This Spike-Timing Dependent Synaptic Plasticity (STDP) can be considered a temporally more precise form of Hebbian learning. The STDP rule indicated above supposes that pairs of spikes (one presynaptic and one postsynaptic action potential) within some time window cause a weight change. However, experimentally it was shown that at least three spikes are necessary (one presynaptic and two postsynaptic spikes). Moreover, the voltage of the postsynaptic neuron matters even in the absence of spikes.

In most models of Hebbian learning and STDP, the factors $c_0, c_1^{\text{pre}} \ldots$ are constant or depend only on the synaptic weight. However, in biological context the speed of learning is often gated by neuromodulators. Since some of these neuromodulators contain reward-related information, one can think of learning as a three-factor rule where weight changes depend on presynaptic activity, postsynaptic activity, and the presence of a reward-related factor. A prominent neuromodulator linked to reward information is dopamine. Three factor learning rules fall in the class of reinforcement learning algorithms.

Cross References

►Dimensionality Reduction
►Reinforcement Learning
►Self-Organizing Maps

Recommended Reading

Bliss, T., & Gardner-Medwin, A. (1973). Long-lasting potentation of synaptic transmission in the dendate area of unanaesthetized rabbit following stimulation of the perforant path. *The Journal of Physiology, 232*, 357–374.

Bliss, T., Collingridge, G., & Morris, R. (2003). Long-term potentiation: Enhancing neuroscience for 30 years - introduction. *Philosophical Transactions of the Royal Society of London. Series B : Biological Sciences, 358*, 607–611.

Cooper, L., Intrator, N., Blais, B., & Shouval, H. Z. (2004). *Theory of cortical plasticity.* Singapore: World Scientific.

Dayan, P., & Abbott, L. F. (2001). *Theoretical Neuroscience*. Cambridge, MA: MIT Press.

Gerstner, W., & Kistler, W. K. (2002). *Spiking neuron models*. Cambridgess, UK: Cambridge University Press.

Gerstner, W., Kempter, R., van Hemmen, J. L., & Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature, 383*, 76–78.

Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.

Lisman, J. (2003). Long-term potentiation: Outstanding questions and attempted synthesis. *Philosophical Transactions of the Royal Society of London Series B, Biological Sciences, 358*, 829–842.

Malenka, R. C., & Nicoll, R. A. (1999). Long-term potentiation–a decade of progress? *Science, 285*, 1870–1874.

Markram, H., Lübke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postysnaptic AP and EPSP. *Science, 275*, 213–215.

Schultz, W., Dayan, P., & Montague, R. (1997). A neural substrate for prediction and reward. *Science, 275*, 1593–1599.

Biomedical Informatics

C. David Page, Sriraam Natarajan
University of Wisconsin Medical School, Madison, USA

Introduction

Recent years have witnessed a tremendous increase in the use of machine learning for biomedical applications. This surge in interest has several causes. One is the successful application of machine learning technologies in other fields such as web search, speech and handwriting recognition, agent design, spatial modeling, etc. Another is the development of technologies that enable the production of large amounts of data in the time it used to take to generate a single data point (run a single experiment). A third most recent development is the advent of Electronic Medical/Health Records (EMRs/EHRs). The drastic increase in the amount of data generated has led the biologists and clinical researchers to adopt algorithms that can construct predictive models from large amounts of data. Naturally, machine learning is emerging as a tool of choice.

In this article, we will present a few data types and tasks involving such large-scale biological data, where machine learning techniques have been applied. For each of these data types and tasks, we first present the required background, followed by the challenges involved in addressing the tasks. Then, we present the machine learning techniques that have been applied to these data sets. Finally and most importantly, we

present the lessons learned in these tasks. We hope that these lessons will be helpful to researchers who aim to apply machine learning algorithms to biological applications and equip them with useful knowledge when they collaborate with biological scientists.

Some of the data types that we present in this work are:

- Gene expression microarrays
- SNPs and genetic data
- Mass spectrometry and other proteomic data
- High-throughput screening data for drug design
- Electronic Medical Records (EMR) and personalized medicine

Some of the key lessons learned from all these data types include the following: (1) We can often do surprisingly well with far more features than data points if there are many highly predictive features (e.g., predicting cancer from microarray data) and if we use methods that are robust to overfitting such as *Voted Decision Stumps* (Hardin et al., 2004; Waddell et al., 2005) (►Ensemble Learning and ►Decision Stumps), ►Naive Bayes (Golub et al., 1999; Listgarten et al., 2004), or *Linear Support Vector Machines (SVMs)* (see ►Support Vector Machine) (Furey et al., 2000; Hardin et al., 2004). (2) Bayes Net learning (Friedman, 2000) (see ►Bayesian Methods) often does not give us causality, but ►Active Learning and ►Time-Series data help if available (Pe'er, Regev, Elidan, & Friedman, 2001; Ong, Glassner, & Page, 2002; Tucker, Vinciotti, Hoen, Liu, & Famili, 2005; Zou & Conzen, 2005). (3) Multi-relational methods are useful for EMRs or molecular data as the data in these cases are very highly relational (see ►Multi-relational Data Mining). (4) There are more important issues than just increasing the accuracy of the learned model on these data sets. Such issues include how data was created, its comprehensibility (physicians typically want to understand the model that has been learned), and its privacy (some data sets contain private information that cannot be posted on public web sites and cannot even be downloaded off site).

The rest of the paper is organized as follows: First we present gene expression microarrays, followed by SNPs and other genetic data. We then present mass spectrometry (MS) and related proteomic data. Next, we present high-throughput screening data for drug design, followed by EMR data and personalized medicine. For each of these data types, we motivate the problem and survey the different machine learning solutions. Finally, we conclude by outlining the lessons learned from all these data types and presenting some interesting and exciting directions for future research.

Gene Expression Microarrays

This data type was presented in detail in *AI Magazine* (Molla et al., 2004) and hence we will brief it in this section. We encourage the reader to read Molla et al. (2004) for more details on this data type. Genes are contained in the DNA of an organism. The mechanism by which proteins are produced from their corresponding genes is a two-step process. The first step is the *transcription* of a gene into a messenger RNA (mRNA) and in the second step called as *translation*, a protein is built using mRNA as a blueprint.

One property that DNA and RNA have in common is that each is a chain of chemicals called as *bases*. In the case of DNA, these bases are *Adenine*, *Cytosine*, *Guanine*, and *Thymine*, commonly referred to as A, C, G, and T, respectively. RNA has the same set of four bases, except Thymine; RNA has *Uracil*, commonly referred as U. An important characteristic of DNA and RNA is *complementarity*, that is, each base only binds well with its complement: A with T (or U) and G with C. As a result of complementarity, a strand of either DNA or RNA has a strong affinity toward what is known as its *reverse complement*, which is a strand of either DNA or RNA that has bases exactly complementary to the original strand. Complementarity is central to the processes of replication of the DNA and transcription.

In addition, complementarity can be used to detect specific sequences of bases within strands of DNA and RNA. This is done by first synthesizing a *probe*, a piece of DNA that is the complement of a sequence that one wants to detect, and then introducing this probe to a solution containing the genetic material (DNA or RNA) to be searched. This solution of genetic material is called the *sample*. In theory, the probe will bind to the sample if and only if the probe finds its complement in the sample (in reality, this process is often imperfect). The act of binding between a sample and probe is called *hybridization*. Prior to the experiment, a biologist labels the probe using a florescent flag. After the

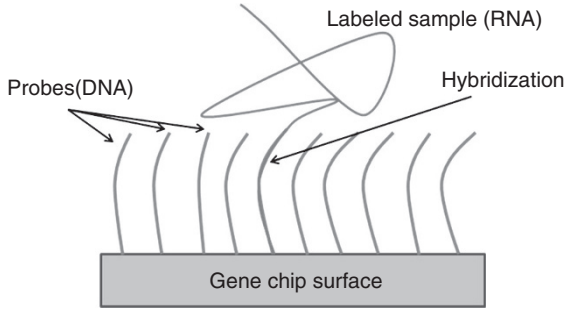
hybridization experiment, one can easily scan to see if the probe has hybridized to its reverse complement in the sample. This allows the molecular biologist to determine the presence or absence of the sequence in the sample.

Gene Chips

DNA probe technology has been adapted for detection of tens of thousands of sequences simultaneously. This has become possible due to the device called a *microarray* or *gene chip*, the working of which is illustrated in Fig. 1. When using the chips it is more common to label (luminescently) the samples than the probe. Thousands of copies of this labeled sample are spread across the probe, followed by washing away any copies that do not remain bound. Since the probes are attached at specific locations on the chip, if a labeled sample is detected at any position in the chip, the probe that is hybridized to its complement can be easily determined. The most common use of these gene chips is to measure the expression levels of various genes in the organism.

Probes are typically on the order of 25-bases long, whereas samples are usually about 10 times, as long, with a large variation due to the process that breaks up long sequences of RNA into small samples (Molla et al., 2004).

To understand about the biology of an organism, say to understand human biology to design new drugs or lower the blood pressure or to cure diabetes, there is a necessity to understand the degree to which different genes get expressed as proteins under different conditions and different cell types. It is much easier to estimate the amount of mRNA for a gene than the protein-production rate. Microarrays provide the measurement of RNAs corresponding to the given gene rather than the amounts of protein.



Biomedical Informatics. Figure 1. Hybridization of sample to probe

In brief, experiments with the microarrays are performed as follows: As can be seen from the figure, probes are DNA strands attached to the gene chip surface. A typical probe length is 25 bases (i.e., 25 letters from *A*, *C*, *G*, *T* to represent a gene). There may be several different subsequences of these 25 bases. Then the mRNA (which is the labeled sample) is passed over the microarrays and the mRNA will bind to the complementary DNA corresponding to the gene better than the other DNA strings. Then the florescence levels of the different gene chips segments are measured, which in turn measures the amount of mRNA on that surface. This mRNA measurement serves as a surrogate to the expression level of the gene.

Machine Learning for Microarrays

The data from microarrays (gene chips) have been analyzed and used by machine learning researchers in two different ways:

1. Data points are genes. This is the case where the examples are genes while the features are the samples (measured expression levels of a single gene under a variety of conditions). The goal of this view is to categorize new genes based on the current set of examples.
2. Data points are samples (e.g., patients). This is the case where the examples are patients and the features are the measured expression levels of genes under one condition.

The problems have been approached in two different ways. In the ►Unsupervised Learning approach, the goal is to cluster the genes according to their expression levels or to cluster the patients (samples) based on their gene expression levels, or both. Hierarchical clustering is especially widely applied. As one of many examples, see Perou et al. (1999). In the ►Supervised Learning setting, the Class labels are the category of the genes or the samples. The latter is the more common supervised task, each sample being mRNA from a different patient (with the same cell type from each patient) or an organism under different conditions to learn a model that accurately predicts the class based on the features. The features could be the patient's expression values for each

gene, while the class labels might be the patient's disease state. We discuss this task further in the subsequent paragraphs.

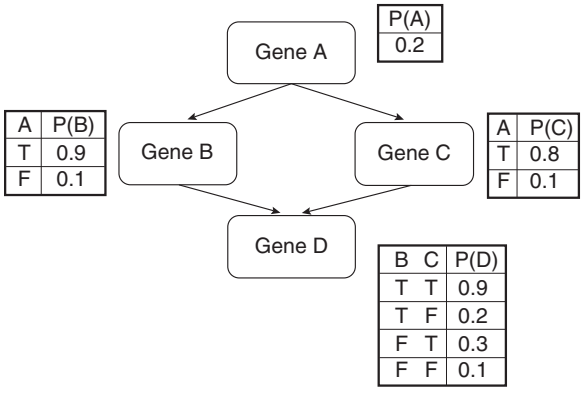
Yet another widely studied supervised learning task is to predict cancer vs. normal for a wide variety of cancer types. One of the significant lessons learned is that it is easy to predict cancer vs. normal in patients based on the gene expression by several machine learning techniques, largely regardless of the type of cancer. The main reason for this is that if cancer is present, many genes in the cancer cells "go haywire" and hence are very predictive of the cancer. The primary challenge in this prediction problem is the noise in the data (impure RNA, cross-hybridization, etc.).

Other related tasks that have been addressed include distinguishing related cancer types and distinguishing cancer from a related benign condition. An early success was a work by Golub et al. (1999), distinguishing acute myeloid leukemia and acute lymphoblastic leukemia (ALL). They used a weighted voting algorithm similar to Naive Bayes and achieved a very high accuracy. This result has been repeated on this data with many other machine learning (ML) approaches. Other work examined multiple myeloma vs. benign condition. This task is challenging because the benign condition is very similar to the cancer, and hence the machine learning algorithms had a difficult time predicting accurately. We refer to Hardin et al. (2004) for more details on the experiments.

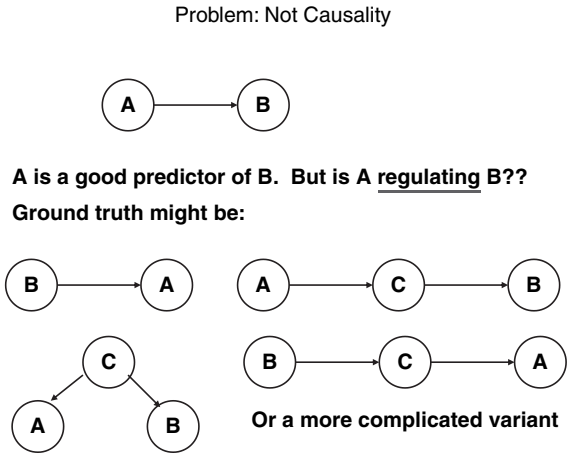
Another important lesson for machine learning researchers from this data type is that the biologists often do not want one predictive model, but a rank-ordered list of genes that a biologist can explore further with additional lab tests on certain genes. Hence, there is a need to present a small set of highly interesting genes to perform follow-up experiments on. Toward this end, statisticians have used mutual information or a t-test to rank the genes. When using a t-test, they check if the mean expression levels are different under the two conditions (cancer vs. normal), yielding a p-value. But the issue is that when working with a large number of genes (typically in the order of 30,000), there could be some genes with lower p-value by chance. This is known as the "multiple comparisons problem." One solution is to do a Bonferoni correction (multiply p-values by the number of genes), but this can be a drastic step and may eliminate all the genes. There are other methods such as false discovery rate (Storey & Tibshirani, 2003) that uses the notion of q-values. We do not go into detail of this method. But the key recommendation we make is that such a method should be used along with the supervised learning method, as the biological collaborators might be interested in the ranking of genes.

One of the most important research directions for the use of microarray data lies in the prognosis and treatment. The features are the same as those of diagnosis, but the class value becomes life expectancy for a given treatment (or a positive response vs. no response to a given treatment). The goal is to use the person's genes to make these predictions. An example of this is the breast cancer prognosis study (Van't Veer et al., 2002), where the goal is to predict good prognosis (no metastastis within 5 years of initial diagnosis) vs. poor prognosis. They used an ensemble of voting algorithms and obtained very good results. Nevertheless, an important lesson learned from this experiment and others was that when using ▶cross-validation, there is a need to tune parameters and perform feature selection *independently* on *each fold* of the cross-validation. There can be a large number of features, and it is natural to want to reduce the size of the data set before working with it. But reducing the number of features by some measure of correlation with the class, such as information gain, using the entire data set means that on each fold of cross-validation, information has leaked from the labeled test set into the training process – labels of test cases were used to eliminate many features from the training set. Hence, selecting features by looking at the entire data set can partially negate the effect of cross-validation, sometimes yielding accuracy estimates that are more than 10% points overly optimistic. Hence the entire training process of selecting features, tuning parameters, and learning a model must be repeated for every fold in cross-validation by looking only at the training data for that fold.

An important use of microarrays for prognosis and therapy is in the area of predictive personalized medicine (PPM). While we present the idea of PPM later in the paper, it must be mentioned that combining gene expression data with clinical trials of the patients to recommend the best treatment for the patients is a very exciting problem with promising impact in the area of PPM.



Biomedical Informatics. Figure 2. A simple Bayes net. The actual learning task typically involves thousands of variables



Biomedical Informatics. Figure 3. Why a learned Bayesian network may not be representing regulation of one gene by another

Bayesian Networks for Regulatory Pathways: ▶Bayesian Networks have been one of the successful machine learning methods used for the analysis of microarray data. Recall that a Bayes net is a directed acyclic graph, such as the one shown in Fig. 2 that defines a joint distribution over the variables using a set of conditional distributions. Friedman and Halpern (Friedman & Halpern, 1999) were the first to use Bayes nets for the microarrays data type. In particular, the problem that was considered was finding regulatory pathways in genes. This problem can be posed as a supervised learning task as follows:

- *Given*: A set of microarray experiments for a single organism under different conditions.
- *Do*: Learn a graphical model that accurately predicts expression of some genes in terms of others.

Friedman and Halpern showed that using statistical methods, a Bayes net representing the observations (expression levels of different genes) can be learned automatically. A main advantage of Bayes nets is that they can (potentially) provide insight into the interaction networks within cells that regulate the expression of genes. But one has to exercise caution, interpreting the arcs of a learned Bayes net as representing causality. For example in Fig. 2, one might interpret the network to mean that gene A causes gene B and gene C to be expressed, in turn influencing gene D. Note that however, the Bayes net in this case just denotes the correlation and not the causality, that is, the direction of an arc merely represents the fact that one variable is a good predictor of the other, as illustrated in Fig. 3.

One possible method of learning causality is to use *knock-out* methods [Pe'er, Regev, Elidan, & Friedman, 2001], where for 300 of the genes in *S. cerevisiae* (bakers' yeast), biologists have created a *knock-out mutant* or a genetic mutant lacking that gene. If the parent of a gene in the Bayes net is knocked out and the child's status remains unchanged, then it is unlikely that the arc from the parent to the child captures causality. A key limitation is that the mutants are not available for many organisms. Some other approaches such as RNAi have been proposed for more efficiently doing knock-outs, but a limitation is that RNAi typically reduces rather than eliminates expression of a gene.

Ong, Glassner, and Page (2002) used time-series data (data from the same organism at various time points) to partially address the issue of causality. They used these data to learn dynamic Bayesian networks in order to infer temporal direction for gene interactions, thereby getting a potentially better handle on causality. DBNs have been employed by other researchers for time-series gene expression data, and the approach has been extended to learn DBNs with continuous variables (Segal, Pe'er, Regev, Koller, & Friedman, 2005).

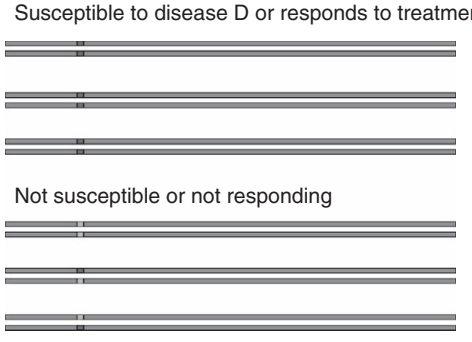
Single Nucleotide Polymorphisms

Single-Nucleotide Polymorphisms (SNPs) are individual base positions (i.e., single-nucleotide positions)

in DNA, where people (or the organism of interest) vary. Most of the variation in human DNA is due to SNPs variations. (There are other variations such as copy number, insertions and deletions that we do not consider in this article.) There are well over three million known SNPs in humans. Technologies such as *Illumina* or *Affymetrix whole-genome* scan can measure a million SNPs in short time. The measurement of these variations is an order of magnitude faster, easier, and cheaper than sequencing all the genes of the person.

It is believed that in the next decade, it will be possible to obtain the entire genome sequence for an individual human for under $1,000 (Mardis, 2006). If we had every human's entire sequence, it could be used to predict the susceptibility of diseases for humans or the adverse reactions to drugs for a certain subset of patients. The idea is illustrated in Fig. 4. Suppose the red dots in the figure are two copies of nucleotide *A*, and the green dots denote a different nucleotide, say *C*. As can be seen from the figure, people who respond to a treatment *T* (top half of the figure) have two copies of *A* (for instance, these could be the positive examples), while the people who do not respond to the treatment have at most one copy of *A* (negative examples and are presented in the bottom half of the figure). Now, we can imagine modeling the sequence to predict the susceptibility to a disease or responsiveness to a treatment.

SNP data can serve as a surrogate for the above problem. SNPs allow us to detect the variations among humans. An example of SNP data is presented in Fig. 5



Biomedical Informatics. Figure 4. Example application of sequencing human genes. The top half is the case, where patients respond to a treatment and the bottom is the case, where three patients do not respond to the treatment

for the prediction of *myeloma* cancer that is common with older people (with age > 70) and is very rare in younger people (age < 40). This data set consists of 40 people diagnosed with myeloma at young age and 40 people who weren't diagnosed till they were 70 when the disease is more common. Most SNP positions represent a pair of nucleotides and are typically restricted in the combinations of values they may assume. For example, in the figure, SNP 1 can take values from the three possible combinations $< C\ T,\ C\ C,\ T\ T >$ for its two positions. The goal is to use the feature values of the different SNPs to predict the class label which could be the susceptibility. That is, the goal is to determine genetic difference between people who got the disease at a young age vs. people who did not until they were old.

There is also the possibility of two patients having the same SNP pattern in the data but not the identical DNA. Patients 1 and 2 may have CT for the SNP1 and GA for SNP2, where both SNPs are on chromosome 1. But, Patient 1 has *C* on SNP1 in the same copy of chromosome 1 as the *G* in SNP2, whereas Patient 2 has *C* on the same copy as an *A*. Hence, while they have the same SNP pattern of CT and GA, they do not have identical DNA. The process of converting the data from the form in the Figure 5 below to the form above is called *Phasing*. From a machine learning perspective, there is a choice of either working with the *unphased* data or to use an algorithm for phasing. It turns out that phasing is very difficult and is an active research area. If there are a number of unrelated patients phasing is very hard. Hence many machine learning researchers work mainly with unphased data. Admittedly, there is a small loss of information with the unphased data that compensates for the difficulty of phasing.

Most biologists and statisticians using SNP data perform genome-wide associations studies (GWAS). The goal in this work is to find individual SNPs that are significantly associated with disease, that is, such that one of the SNP values, or alleles, raises the risk of disease. This is typically measured by "relative risk" or by "odds ratio," and significance is typically measured by statistical tests such as Wald test, Score test, or LRLR (►logistic regression log likelihood, where each SNP is used individually to predict disease, and log likelihood of the predictive model is compared to guessing under the null hypothesis that the SNP is not associated).

Person ▼ SNP ►	1		2		3		...	Class
Person 1	C	T	A	G	T	T	...	Old
Person 2	C	C	A	G	C	T	...	Young
Person 3	T	T	A	A	C	C	...	Old
Person 4	C	T	G	G	T	T	...	Young
. . .	.	.	.	...	.			
. . .	.	.	.	...	.			
. . .	.	.	.	...	.			

Biomedical Informatics. Figure 5. Example of SNP data

One of many examples is the use of SNPs to predict susceptibility to breast cancer (Easton et al., 2007).

The advantages of SNP data compared to microarray data are the following: (1) Because SNP analysis is typically performed on DNA from saliva or peripheral blood cells, a person's SNP pattern does not change with time or disease. If the SNPs are collected from a blood sample of a person aged 40 years, the SNP patterns are probably the same as when they were born. This gives more insight to the susceptibility of the person to many diseases. Hence, we do not see the widespread changes in SNP pattern with cancer, for example, that we see in microarray data from tumor samples. (2) It is easier to collect the samples. These can be obtained from the blood samples as against obtaining say, the biopsy of other tissue types.

The challenges of SNP data are as follows: (1) As explained earlier, the data is unphased. Algorithms exist for phasing (haplotyping), but they are error prone and do not work well with unrelated patient samples. They require the data to consist of related individuals in order to have a dense coverage. (2) ►Missing Values are more common than in microarray data. The good news is that the amount of missing values is decreasing substantially (down from 30–40% a few years ago to 1–2%). (3) The sheer volume of measurements – currently, it is possible to measure a million SNPs out of over three million SNPs in the human genome. While this provides a tremendous amount of potential information, the resulting high dimensionality causes problems for machine learning. As with gene expression microarray data, we have a multiple comparisons problem, so approaches such as Bonferoni correction or q-values from False Discovery Rate can again be applied. But even when a significant SNP is found, it usually only increases our accuracy at predicting disease by 2% or 3% points, because a single SNP typically either has a small effect or small penetrance (the variation is fairly rare – one value of the SNP is strongly predominant). So GWAS are missing a major opportunity to build predictive models by combining multiple SNPs with small effects – this is an exciting opportunity for machine learning.

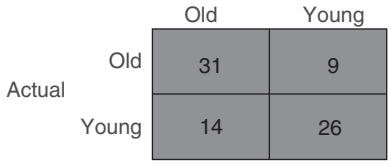
The supervised learning task can be defined as follows:

- *Given*: A set of SNP profiles each from a different patient.

  Phased: Nucleotides at each SNP position on each copy of *each chromosome* constitute the *features* and patient's *disease susceptibility* or *drug response* constitutes the *class*.

  Unphased: Unordered pair of nucleotides at each SNP position constitutes the *features* and patient's *disease susceptibility* or *drug response* constitutes the *class*.
- *Do*: Learn a model to predict the class based on the features.

We now briefly present one example of supervised learning from SNP data. (Waddell, Page, and Shaughnessy (2005)) found that there was evidence of a genetic component in predicting the blood cancer *multiple myeloma* as it was possible to distinguish the two cases significantly better than chance (71% accuracy). The results from using Support Vector Machines (SVMs) are



Biomedical Informatics. Figure 6. Results on predicting multiple myeloma, young (susceptible) vs. old (less susceptible), 3,000 SNPs

presented in Fig. 6. Similar results were obtained using a Naive Bayes model as well. Listgarten et al. (2004) also used the SNP data with the goal of predicting *lung cancer*. The accuracy of 69% obtained by them was remarkably similar to the task of predicting *multiple myeloma*. The best models for predicting lung cancer were also Naive Bayes and SVMs. There is a striking similarity between the two experiments on unrelated tasks using SNPs. When only the individual SNPs were considered, the accuracy for both the experiments fell to 60%.

The lessons learned from SNP data are the following: (1) ►Supervised learning algorithms such as ►Naive Bayes and ►SVM that can handle large number of features in the presence of smaller number of training examples can predict disease susceptibility at rates better than chance and better than individual SNPs. (2) Accuracies are much lower than the ones with microarray data. This is mainly due to the fact that we are predicting the susceptibility to the diseases (or the response to a drug) as against predicting whether a person already has the disease (as with the microarray data). While we are predicting using the genetic component, there are also many environmental components that are responsible for the diseases and the response. We are not considering such components in our model and hence the accuracies are often not very high. In spite of relatively lower accuracies, they give a different valuable insight to the human gene.

We now briefly outline a couple of exciting future directions for the use of SNP data. *Pharmacogenetics* is the problem of predicting drug response from SNP profile and has been gaining momentum over the past few years. This includes predicting drug efficacy and adverse reactions to certain drugs, given a person's SNP profile. A recent New England Journal of Medicine article showed that the analysis of SNPs can significantly improve the dosing model for the most widely used orally available blood thinner, Warfarin (IWPC, 2009). Another exciting direction is the combination of SNP data with other data types such as clinical data that includes the history of the patient and the lab tests and microarray data. The combination of these different data sets will not only improve the accuracy of the learned model but also provide a deeper insight to the different kinds of interactions that occur within a human, such as gene interactions with other drugs.

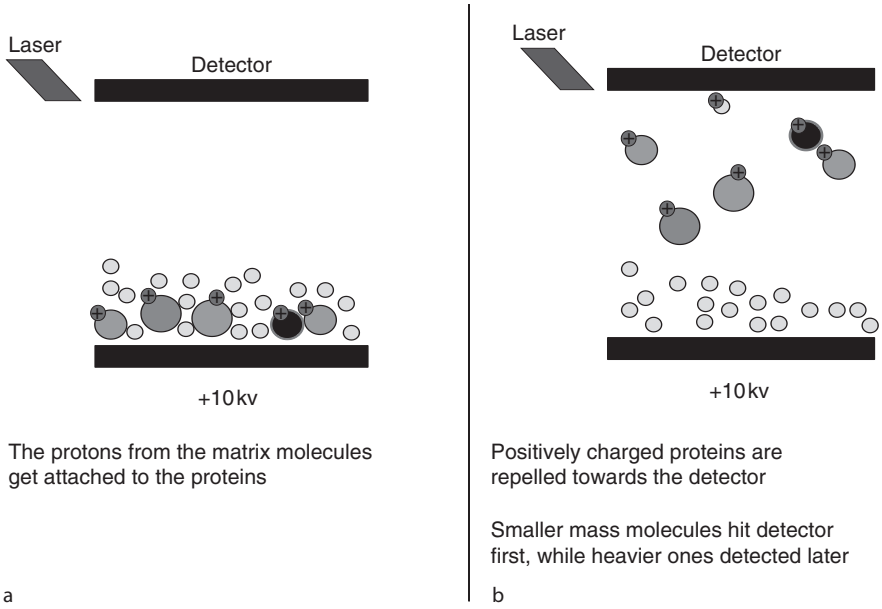
It should be mentioned that other genetic data types are becoming available and may be useful for supervised learning as well. These data types can provide additional information about DNA sequence beyond SNPs but without the expense of full genome sequencing. They include copy-number variations and exon-sequencing.

Mass Spectrometry and Proteomics

Microarrays are useful primarily because mRNA concentrations can serve as surrogates for protein concentrations and they are easier to measure. Though measuring protein concentrations directly is possible, it cannot be done in the same high-throughput manner as measuring mRNA. Recently, techniques such as *Mass Spectrometry* (MS or mass spec) have been successful in high-throughput measuring of proteins. Mass spec still does not given the complete coverage that microarrays provide, nor as good a quantitation.

Mass spectometry is improving on many fronts, using many technologies. As one example, we present *Time-Of-Flight (TOF) Mass Spectometry* illustrated in Fig. 7. This measures the time required for an ionized particle starting from the sample plate (bottom of the figure) to hit the detector. The key idea is to place some proteins (indicated as larger circles) into a matrix (smaller circles are the matrix molecules). Because of mass spec limitations, the proteins typically are digested (broken into smaller peptides), for example, by the compound trypsin. When struck by a laser, the matrix molecules release protons that attach themselves to the peptides or protein fragments (shown in (a)). Note that the plate where the peptides are present is positively charged. This causes the peptides to migrate toward the detector.

As can be seen in (b) of the figure, the molecules with smaller mass move faster toward the detector. The idea is to detect the number of molecules that hit the



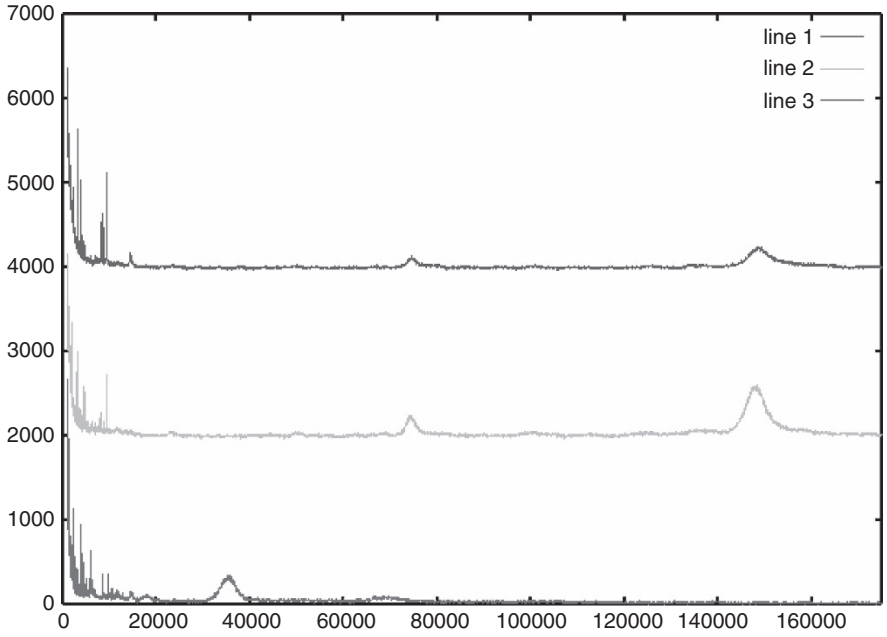
Biomedical Informatics. Figure 7. Time-Of-Flight mass spectrometry

detector at any given time. This makes it possible to use time as a surrogate for mass of the protein. The experiment is repeated a number of times, counting frequencies of "flight-times." Plotting time vs. the number of particles hitting the detector yields a spectrum as presented in Fig. 8. The figure shows three different fractions from the same sample. These kinds of spectra provide us an insight about the different types of proteins in a given sample. A technical detail is that sometimes molecules receive additional charge (additional protons) and hence fly faster. Therefore, the horizontal mass axis in a spectrum is actually a mass/charge ratio.

The main issues for machine learning researchers working with mass spectrometry data compared to microarray data are as follows: (1) There is a lot of ►Noise in the data. The noise is due to extra peaks from handling of sample, from machine and environment (e.g., electrical noise). Also the mass to charge values may not exactly align across the spectra; the accuracy of the mass/charge values is the resolution of the mass spec. (2) Intensities (peak heights) are not calibrated across the spectra, making quantification difficult. This is to say that if one spectrum is compared to another, and if one of them has more intensity at a particular mass/charge, it does not necessarily mean that the levels of the peptide at that mass/charge are higher in that spectrum. (3) Another issue is that the mass spectrometry data is not as comprehensive as microarray data, in that it is not possible to measure all peptides (typically only several hundred of them can be obtained). To get the best results, there is a need to fractionate the sample beforehand, getting different groups of proteins in different subsamples (fractions). (4) As already mentioned, the proteins themselves typically must be broken down (digested) into smaller peptides in order to get accurate readings from the mass spec. But this means processing is needed afterward not only to determine from a spectrum which peptides are present but also from that determination which proteins are present. It is worth noting that some of these challenges are being partially addressed by ongoing improvements in mass spectrometry technologies, including the use of "tandem mass spectrometry."

This data type opens up a lot of possibilities for machine learning research. Some of the learning tasks include:

- Learn to predict proteins from spectra, when the organism's proteome (full set of proteins) is known.
- Learn to identify isotopic distributions (combinations of multiple peaks for a given molecule



Biomedical Informatics. Figure 8. Example spectra from a competition by Lin et al.

arising from different isotypes of carbon, nitrogen. and oxygen).

- Learn to predict disease from either proteins, peaks or isotopic distributions as features.
- Construct pathway models.

We will now present one case study that was successful and generated a lot of interest – *Early Detection of Ovarian Cancer* (Petricoin et al., 2002). Ovarian cancer is difficult to detect early, often leading to poor prognosis. The goal of this work was to predict ovarian cancer from blood samples. To this effect, the researchers trained and tested on mass spectra from blood serum. They used 100 training cases (50 positive) and used a held-out test set of 116 cases (50 positive). The results were extremely impressive (100% sensitivity, 95% specificity).

While the results were extremely impressive and while the machine learning methodology seemed very sound, it turns out that the preprocessing stage of the data may have introduced errors (Baggerly, Morris, & Combes, 2004). Mass spectrometry is very sensitive to the external factors as well. For instance, if we run cancer samples on Monday and normal samples on Wednesday, it is possible that we could get differences from variations in the machine or nearby electrical equipment that is running on Monday but not Wednesday. Hence, one of the important lessons learned from this data type is the need for *careful randomization* of the data samples. This is to say that we should sample the positive and negative samples under identical conditions. It should not be the case that the positive examples are run through the machine on one day and the negatives on the other day. Any preprocessing of the data must be performed similarly.

While mass spectrometry is a widely used type of high-throughput proteomic data, other types of data are also important and are briefly covered next.

Protein Structures

X-ray crystallography and nuclear magnetic resonance are widely used to determine the three-dimensional structures of proteins. Predicting protein structures has been a very fertile field for machine learning research for several decades.

While the amino acid sequence of a protein is called its primary structure, it is more difficult to determine secondary structure and tertiary (3D) structure. Secondary structure maps subsequences of the primary

structure in the three classes of alpha helix (helical structures akin to a telephone cord, often denoted by A), beta strand (which comes together with other strand sections to form planar structures called beta sheets, often denoted by B), and less descript regions referred to as coil, or loop regions, often denoted by C.

Predicting secondary structure and tertiary structure has been a popular topic for machine learning for many years, because training data exists yet it is difficult and expensive to experimentally determine structures. We will not attempt to survey all the work in this area. Waltz and colleagues (Zhang, Mesirov, & Waltz, 1992) showed the benefit of applying neural networks to the task of secondary structure prediction, and the best secondary structure predictors (e.g., Rost & Sander, 1993) have continued to be constructed by machine learning over the years. Approaches for predicting the tertiary structure have also relied heavily on machine learning and include ab initio prediction (e.g., Bonneau & Baker, 2001), prediction aided by crystallography data (e.g., DiMaio et al., 2007), and homology-based prediction (by finding similar proteins). For over a decade, there has been a regular competition in the prediction of protein structures (Critical Assessment of Structure Prediction [CASP]).

Protein–Protein Interactions

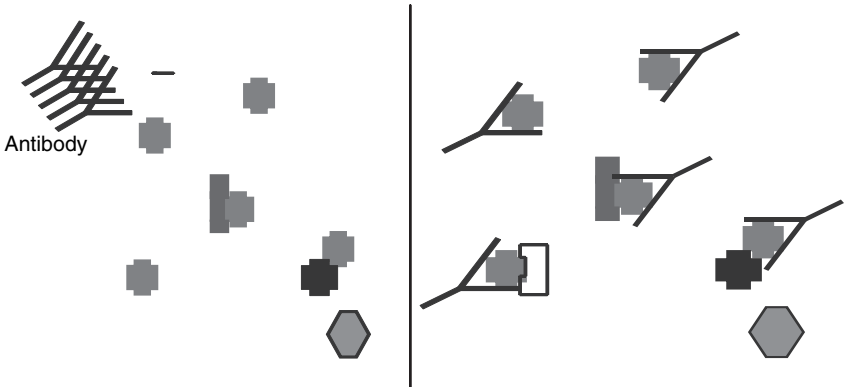
Another proteomics data type is protein–protein interactions. This is illustrated in Fig. 9. The idea is to identify proteins that interact with the current protein say *P*. Generally, this is performed as follows: In the sample, there are some proteins of type *X* (shown in pink in the figure) and other types of proteins. Proteins that interact with *X* are bonded to *X*. Then antibodies (shown as Y-shaped green objects) are introduced in the sample. The idea of antibodies is to collect the proteins of type *X*. Once the antibodies have collected all protein *X*'s in the sample, they can be analyzed through mass spectrometry presented earlier.

A particularly high-throughput way of measuring protein–protein interactions is through "ChIP-chip" data. The supervised learning tasks for this task include:

- Learn to predict protein–protein interactions: Protein three-dimensional structures may be critical.
- Use protein–protein interactions in construction of pathway models.
- Learn to predict protein function from interaction data.

Related Data Types

- *Metabolomics* measures concentration of each low-molecular-weight molecule in sample. These typically are *metabolites*, or small molecules produced or consumed by reactions in biochemical pathways. These reactions are typically catalyzed by proteins (specifically, enzymes). This data typically uses mass spectrometry.



Biomedical Informatics. Figure 9. Schematic of antibody-based identification of protein–protein interactions

- *ChIP-chip* data measures protein–DNA interactions. For example, transcription factors are proteins that interact with DNA in specific locations to alter transcription of a nearby gene.
- *Lipomics* is analogous to metabolomics, but measuring concentrations of Lipids rather than metabolites. These potentially help induce biochemical pathway information or to help disease diagnosis or treatment choice.

High-Throughput Screening Data for Drug Design

The typical steps in designing a drug are: (1) Identifying a target protein – for example, while developing an antibiotic, it will be useful to find a protein that belongs to the bacteria that we are interested in and find a small molecule that will bind to that protein. In order to perform this, we need the knowledge of proteome/genome and the relevant biological path ways. (2) Determining the target site structure once the protein has been identified – this is typically performed using crystallography. (3) Finding a molecule that will bind to the target site. These steps are presented in Fig. 10.

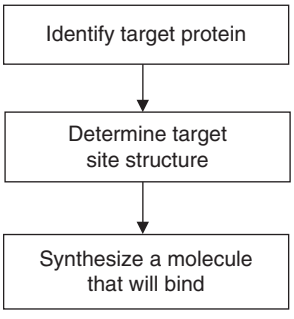
The molecules that bind to the target may have a number of other problems and hence they cannot directly be used as a drug. Some common problems are as follows: (1) They may bind too tightly or not tightly enough. (2) They may be toxic. (3) They may have unanticipated side effects in the body. (4) They may break down as soon as they get into the body or may not leave the body soon enough. (5) They may not get to the right target in the body (e.g., cross blood–brain barrier). (6) They may not diffuse from gut to bloodstream. Also, since the organisms are different, even if a molecule works in the test tube and in animal studies, it may fail in clinical trials. Also while a molecule may work for some people, it may not work for others. Conversely, while some molecules may cause harmful side effects in some people, they may not do so in others.

Often pharmaceutical companies will use robotic high-throughput screening assays to test many thousands of molecules to see if they bind to the target protein, and then computational chemists will work to determine the commonalities that allow them to bind to the target as often the structure of the target protein cannot be determined. The process of discovering the commonalities across the different molecules presents a great opportunity for machine learning research. The first study of this task using machine learning was by Dietterich, Lathrop, and Lozano-Perez and led to the formulation of Multi-Instance Learning. Yet, another machine learning task could be to predict the reactions of the patients to the drugs.

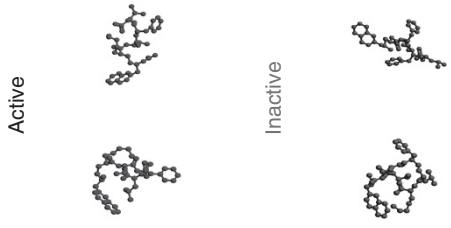
High-Throughput Screening: When the target structure is unknown, it is a common practice to test many molecules (1,000,000) to find some that bind to the target. This is called as *High-Throughput Screening*. Hence, it is important to infer the shape of the target from three-dimensional structural similarities. The shared three-dimensional structure is called as *pharmacophore*. This is a perfect example of a machine learning task with a spatial target and is presented in Fig. 11.

Given: A set of molecules, each labeled by activity (binding affinity for a target protein) and a set of low-energy conformers for each molecule

Do: Learn a model that accurately predicts the activity (may be Boolean or real valued).



Biomedical Informatics. Figure 10. Steps involved in drug design



Biomedical Informatics. Figure 11. An example of structure learning

The common machine learning approaches taken toward solving this problem are:

1. Representing a molecule by thousands to millions of features and use standard techniques (KDD, 2001)
2. Representing each low-energy conformer by feature vector and use multiple-instance learning (Jain et al., 1994)
3. Relational learning – using either Inductive Logic Programming techniques (Finn, Muggleton, Page, & Srinivasan, 1998) or Graph Mining

Thermolysin Inhibitors: We present some results of relational learning algorithms on thermolysin inhibitors data set (Davis, 2007a). Thermolysin belongs to the family of metalloproteases and plays roles in physiological processes such as digestion and blood pressure regulation. The molecules in the data set are known inhibitors of thermolysin. Activity for these molecules is measured in $pKi = -log\ Ki$, where Ki is a dissociation constant, measuring the ratio of the concentrations of bound product to unbound constituents. A higher value indicates a stronger affinity for binding. The data set that was used had the ten lowest energy conformations (as computed by the SYBYL software package [www.tripos.com]) for each of 31 thermolysin inhibitors along with their activity levels.

The key results for this data set using the relational algorithm SAYU (Davis, 2007b) were:

- Ten five-point pharmacophore identified, falling into two groups (7/10 molecules):
  - Three "acceptors," one hydrophobe, and one donor
  - Four "acceptors," and one donor
- Common core of Zn ligands, Arg203, and Asn112 interactions identified
- Correct assignments of functional groups
- Correct geometry to 1 Å tolerance
- Increasing tolerance to 1.5 Å finds common six-point pharmacophore including one extra interaction

Antibacterial Peptides: This is a data set of 11 pentapeptides showing activity against Pseudomonas aeruginosa (Spatola, Page, Vogel, Blondell, & Crozet, 1999). There are six active pharmacophores with < 64 μg/ml of IC_{50} and five inactives. The pharmacophore that has been identified is presented in Table 1.

Biomedical Informatics. Table 1 Identified Pharmacophore

A molecule M is active against *Pseudomonas aeruginosa* if it has a conformation B such that
M has a hydrophobic group C
M has a hydrogen acceptor D
The distance between C and D in conformation B is 11.7 Å
M has a positively charged atom E
The distance between C and E in conformation B is 4 Å
The distance between D and E in conformation B is 9.4 Å
M has a positively charged atom F
The distance between C and F in conformation B is 11.1 Å
The distance between D and F in conformation B is 12.6 Å
The distance between E and F in conformation B is 8.7 Å
Tolerance 1.5 Å

Dopamine Agonists: The last data set that we present here consists of dopamine agonists (Martin et al., 1993). Dopamine works as a neurotransmitter in the brain, where it plays a major role in the movement control. Dopamine agonists are molecules that function like dopamine and produce dopamine-like effects and can potentially be used to treat diseases such as Parkinson's disease. The data set had 23 dopamine agonists along with their activity levels. The pharmacophore identified using Inductive Logic Programming is presented in Table 2.

Electronic Medical Records (EMR) and Personalized Medicine

Predictive personalized medicine (PPM) is a vision of the future, whose parts are beginning to come into place now. Under this vision, physicians can construct safer and more effective prevention and treatment plans for

each patient. This is rendered possible by predicting the impact of treatments on patients – their effectiveness for different classes of patients, adverse reactions of certain drugs that are prescribed to the patients, and susceptibility of different types of patients to diseases. PPM can become a reality due to three reasons: The first is the widespread use by many clinics of *Electronic Medical Records* (EMR also called as *Electronic Health Records* – EHR). The second is that whole-genome scan technology makes it possible in one experiment, for well under $1,000, to measure for one patient a half million to one million SNPs, or individual positions in the DNA where humans vary. The third key reason is the advancement of statistical modeling (machine learning) methods in the past decade that can handle large relational longitudinal databases with significant amount of noise. The first two reasons make it possible for the clinics to have a relational database of the form presented in Fig. 12.

Biomedical Informatics. Table 2 Pharmacophore Identified for Dopamine Agonists

Molecule A has the desired activity if
• In conformation B molecule A contains a hydrogen acceptor at C
• In conformation B molecule A contains a basic nitrogen group at D
• The distance between C and D is 7.05966 ± 0.75 Å
• In conformation B molecule A contains a hydrogen acceptor at E
• The distance between C and E is 2.80871 ± 0.75 Å
• The distance between D and E is 6.36846 ± 0.75 Å
• In conformation B molecule A contains a hydrophobic group at F
• The distance between C and F is 2.68136 ± 0.75 Å
• The distance between D and F is 4.80399 ± 0.75 Å
• The distance between E and F is 2.74602 ± 0.75 Å

Given such a database, it is conceivable to use existing machine learning algorithms for achieving the goal of PPM. These algorithms could focus on predicting which patients are at risk (pos and neg examples). Another task is predicting which patients will respond to a specific treatment – a set of patients who have undergone specific treatments in order to learn predictive models that could be extended to similar patients of the population. Similarly, it is possible to focus on certain drugs and their adverse reactions and use them to predict the adverse reactions of similar drugs that are released in the market. In this work, we focus on the machine learning solutions to predicting adverse drug reactions for different drugs.

There are actually at least three different tasks for machine learning in predicting Adverse Drug Events (ADEs).

Patient ID	Gender	Birthdate
P1	M	3/22/63

Patient ID	Date	Physician	Symptoms	Diagnosis
P1	1/1/01	Smith	Palpitations	Hypoglycemic
P1	2/1/03	Jones	Fever, Aches	influenza

Patient ID	Date	Lab Test	Result
P1	1/1/01	blood glucose	42
P1	1/9/01	blood glucose	45

Patient ID	SNP1	SNP2	⋯	SNP500K
P1	AA	AB		BB
P2	AB	BB		AA

Patient ID	Date Prescribed	Date Filled	Physician	Medication	Dose	Duration
P1	5/17/98	5/18/98	Jones	Prilosec	10 mg	3 months

Biomedical Informatics. Figure 12. Electronic Health Records (dramatically simplified) – most data currently do not include SNP information but are anticipated in the future

Task 1:

Given: Patient data (from claims databases and/or EMRs) and a drug D

Do: Construct a model to predict a minimum efficacious dose of drug D, because a minimum dose is less likely to induce an ADE.

An example of this task is predicting the "stable dose" of the blood-thinner Warfarin (Coumadin) for a patient (McCarty, Wilke, Giampietro, Wesbrook, & Caldwell, 2005). A stable dose of Warfarin yields the desired degree of anticoagulation, whereas a higher dose can lead to bleeding ADEs; the stable dose for a patient is currently found by trial and error, modifying the dose and measuring the degree of anticoagulation. The cited study shows that a learned dosing model can predict a significantly better starting dose (significantly closer to the final "stable dose") than the 5 mg/day starting dose currently used in many clinics.

Task 2:

Given: Patient data (from claims databases and/or EMRs), a drug D, and an adverse event E

Do: Construct a model to predict which patients are likely to suffer the adverse event E if they take D.

In this second task, we assume that the association between D and E already has been hypothesized. We seek to construct models that can predict who will suffer a given event if they take the drug. Here, whether the patient will suffer adverse event *E* is the class variable to be predicted. This task is important for *personalized medicine*, as accurate models for this task can be used to identify patients who should not be given a particular drug. An earlier study has demonstrated the benefit of a Statistical Relational Learning (SRL) system called SAYU (Davis, 2007b) over standard machine learning approaches with a feature-vector representation of the EHR, for the task of predicting which users of cox2 inhibitors would have an MI.

Task 3:

Given: Patient data (from claims databases and/or EMRs) and a drug D

Do: Determine if evidence exists that associates D with a previously unanticipated adverse event.

This third task is the most challenging because no associated event has been hypothesized. There is a need to identify the response variable to be predicted. In brief, the major approach for this task is to use machine learning "in reverse." We seek a model that can predict which patients are on drug *D* using the data after they start the drug (left censored) and also censoring the indications of the drug. If a model can predict (with accuracy better than chance on held-aside data) which patients are taking the drug, there must be some combination of variable settings more common among patients on the drug. Because we have left censored, in theory, this commonality should not consist of common symptoms, but common effects, presumably from the drug. The model can then be examined by the experts to see if it might indicate a possible new adverse event for the drug.

The preceding use of machine learning "in reverse" actually can be viewed as Subgroup Discovery (Wrobel, 1997; Klösgen, 2002), finding a subgroup of patients on drug *D* who share some subsequent clinical events. The learned model – say an IF-THEN rule – need not correctly identify everyone on the drug but rather merely a subgroup of those on the drug, while not generating many false positives (individuals not on the drug). This task poses several different challenges that traditional ML methods will find difficult to handle.

First, the data is *multi-relational*. There are several objects such as doctors, patients, drugs, diseases, and labs that are connected through relations such as visits, prescriptions, diagnoses, etc. If traditional machine learning (ML) techniques are to be employed on this problem, they require flattening the data into a single table. All known flattening techniques such as computing a join or summary features result in either (1) changes in frequencies on which machine learning algorithms critically depend or (2) loss of information. They also typically result in loss of some correlations between the objects and explosion in database size. Second, the data is *non-i.i.d.*, as there are relationships between the objects and between different rows within a table. Third, there are *arbitrary* numbers of patient visits, diagnoses, and prescriptions for different patients. This is to say that there is no fixed pattern in the diagnoses and prescriptions of the patients. It is incorrect to assume that the patients are diagnosed a fixed number of times or to assume only the last diagnosis is relevant. To predict the adverse reactions to a drug, it is important to consider the other drugs that the patient is prescribed or has been prescribed in the past, as well as past diagnoses and laboratory results. To capture

these interactions, it is critical to explicitly model time since the interactions are highly *temporal*. Some drugs taken at the same time can lead to side effects while in some cases, drugs taken after one another cause side effects. It is important to capture such interactions to be able to make useful predictions for the physicians and the Federal Drug Authority (FDA). In this work, we focus on this hardest task and present the results on two data sets.

Cox2 Inhibitors: Recently, a study was performed to see if there were any unanticipated adverse events that occurred when subjects used cox2 inhibitors (Vioxx, Celebrex, and Bextra). Cox2 inhibitors are a nonsteroidal anti-inflammatory class of drugs that were used to reduce joint pain. Vioxx, Celebrex, and Bextra were approved for use in the late 1990s and were ranked as one of the top therapeutic drugs in the USA. Several clinical trials were conducted, and the APPROVe trial (focused on Vioxx outcomes) showed an increase of adverse events from myocardial infarction, stroke, and vascular thrombosis. The manufacturer withdrew Vioxx from the market shortly after the results were published. The other cox2 inhibitor drugs were discontinued shortly thereafter.

This study utilized the Marshfield Clinic's Personalized Medicine Research Project (McCarty, Wilke, Giampietro, Wesbrook, & Caldwell, 2005) (PMRP) cohort consisting of approximately 19,700+ subjects. The PMRP cohort included adults aged 18 years and older, who reside in the Marshfield Epidemiology Study Area (MESA). Marshfield has one of the oldest internally developed Electronic Medical Records (Cattails MD) in the USA, with coded diagnoses dating back to the early 1960s. Cattails MD has over 13,000 users throughout central and northern Wisconsin.

Since the data is multi-relational, an Inductive Logic Programming (Muggleton & Raedt, 1994) system, *Aleph* (Srinivasan, 2001) was used to learn the models. Aleph learns rules in the form of Prolog clauses and scores rules by positive examples covered (P) minus negative examples covered (N). Seventy-five percent of the data was used for training and rule development, while the remaining 25% was used for testing. There were 14,654 subjects within the PMRP cohort that had medication records. Within this cohort, almost 20% of the subjects indicated use of a cox2 inhibitor, and more specifically, 8.5% indicated the use of Vioxx. Approximately, 3.5% of this cohort had an indicated use of clopidogrel biosulfate (Plavix).

Biomedical Informatics. Table 3 Cox2 Inhibitor Test Data Results

	Actual		
Rule	+	–	
+	438	158	596
–	269	549	818
	707	707	1,414
Accuracy	0.69801		

Aleph generated thousands of rules and selected a subset of the "best" rules that were based on the scoring algorithm. The authors also developed specific hypotheses to test for known adverse events to validate the approach (indicated by # A). This rule was: cox2(A):- diagnoses(A, _,'410'). It states that if finding (A): the subject would have the diagnosis coded as *410* (*myocardial infarction*). Aleph also provided summary statistics on model performance for identifying subjects on cox2 inhibitors, as indicated in Table 3. If we assume that the probability of being on the cox2 inhibitor is greater than. 5 (the common threshold), then the model has a predictive probability of 69% to predict cox2 inhibitor use.

OMOP Challenge: Observational Medical Outcomes Partnership (OMOP) designed and developed an automated procedure to construct simulated data sets to identify adverse drug events. The simulated data sets are modeled after real observational data sources but are comprised of hypothetical persons with fictional drug exposure and health outcomes occurrence. The data sets are constructed such that the relationships between the fictional drugs and fictional outcomes are well characterized as *true* and *false* associations. That is, hypothetical persons are created and assigned fictional drug exposure periods and instances of health outcomes based on random sampling from probability distributions that define the relationships between the fictional drugs and outcomes. The relationships created within the simulated data sets are contrived but are representative of the types of relationships observed within real observational data sources. OMOP has made a

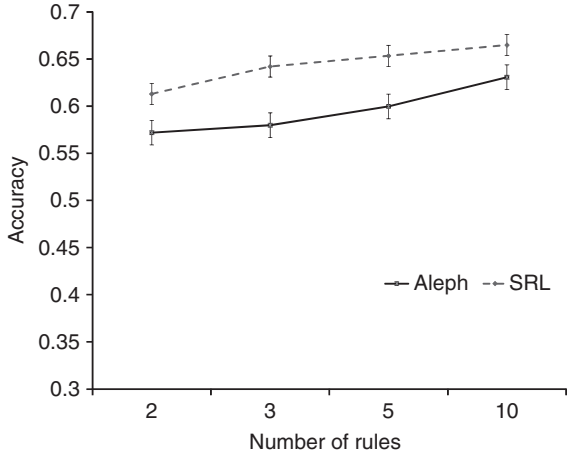
simulated data set and the simulator itself publicly available as part of the OMOP Cup Data Mining Competition (http://omopcup.orwik.com).

Aleph was used to learn rules from a subset of the data (about 10,000 patients). Each patient had a record of drugs and diagnoses (conditions) with dates attached. A few examples of the rules learned by Aleph in this data set are:

```
on_drug(A):-  condition_occurrence(B,C,A,D,
                E,3450,F,G,H)
on_drug(A):-  condition_occurrence(B,C,A,D,E,
                140,F,G,H)
              condition_occurrence(I,J,A,K,L,
                1487,M,N,O)
```

The first rule identifies drug *3450* as interesting, while the second rule identifies two other drugs as interesting when predicting the reaction for person *A*. With about 150 rules, Aleph was able to achieve a 67% coverage. The results were compared against a Statistical Relational Learning technique (SRL) (Getoor & Taskar, 2007) that uses a probability distribution on the rules. The results are presented in Fig. 13. As expected, with a small number of rules, SRL has a better performance than Aleph, but as the number of rules increase, they converge on the same performance.

The leading approaches in the first OMOP Cup include a machine learning approach based on random forests as well as several approaches based on techniques from epidemiology such as disproportionality analysis. At the time of this writing further details, as well as plans for future competitions, are available at http://omopcup.orwik.com/.



Biomedical Informatics. Figure 13. Results of OMOP data

Identifying previously unanticipated ADEs, predicting who is most at risk for an ADE, and predicting safe and efficacious doses of drugs for particular patients are all important needs for society. With the recent advent of "paperless" medical record systems, the pieces are in place for machine learning to help meet these important needs.

Conclusion

In this work, we aim to survey the abundant opportunities in biomedical applications to machine learning researchers by presenting several data types to which machine learning techniques have been applied successfully or showing tremendous promise. One of the most important developments in biology and medicine over the last few years is the availability of technologies that can produce large volumes of data. This in turn has necessitated the need for processing large volumes of data in a reasonable amount of time, presenting the perfect setting for machine learning algorithms to have an impact. We outlined several data types including gene expression microarrays (measuring mRNA), mass spectrometry (measuring proteins), SNP chips (measuring genetic variation), and Electronic Medical/Health Records (EMR/EHRs).

The key lessons learned from all these data types are as follows: (1) Even if the number of features is greater than the number of data points (e.g., predicting cancer from microarray data), we can do well provided the features are highly predictive. (2) Careful randomization of data samples is necessary. (3) It is very easy to overfit the data and hence robust techniques such as voted ▶decision stumps, ▶naive Bayes or linear ▶SVMs are in general very useful tools for such data sets. (4) ▶Bayes nets do not give us causality and hence knock-out experiments (▶active learning) and ▶DBNs with ▶time-series data can help. (5) Multi-relational methods such as SRL and ILP are helpful for predictive personalized medicine due to the relational nature of the data. (6) Mostly, the collaborators are interested in measures other than just accuracy. Comprehensibility, privacy, and ranking are other criteria that are important to biologists.

This chapter is necessarily incomplete because so many exciting tasks and data types exist within biology

and medicine. While we have touched on many of the leading such data types, other related ones also exist. For example, there are many opportunities in analyzing genomic and protein sequences (Learning Models of Biological Sequences). Other opportunities exist within phylogenetics, for example, see work by Heckerman and colleagues on HIV (Carlson et al., 2009). New technologies such as optical mapping are constantly being developed and refined (Ananiev et al., 2008). Machine learning has great potential for developing models for computer-aided diagnosis (CAD), for example, for mammography (Burnside et al., 2009). Data types such as metabolomics and auxotropic growth experiments raise opportunities for active learning and for automatic revision of biological network models, for example, as in the Robot Scientist projects (Jones et al., 2004; Oliver et al., 2009). Incorporation of multiple data types can further help in mapping out the regulatory entities and networks of an organism (Noto & Craven, 2006). It is our hope that this article will encourage some machine learning researchers to delve deeper into these and other related opportunities.

Acknowledgment

We would like to thank Elizabeth Burnside, Michael Caldwell, Mark Craven, Jesse Davis, Lingjun Li, David Madigan, Sean McIlwain, Michael Molla, Irene Ong, Peggy Peissig, Patrick Ryan, Jude Shavlik, Michael Sussman, Humberto Vidaillet, Michael Waddell and Steve Wesbrook.

Cross References

►Learning Models of Biological Sequences

Recommended Reading

Ananiev, G. E., Goldstein, S., Runnheim, R., Forrest, D. K., Zhou, S., Potamousis, K., Churas, C. P., Bergendah, V., Thomson, J. A., & David, C. (2008). Schwartz1. Optical mapping discerns genome wide DNA methylation profiles. *BMC Molecular Biology*, *9*, doi:10.1186/1471-2199-9-68.

Baggerly, K., Morris, J. S., & Combes, K. R. (2004). *Reproducibility of seldi-tof protein patterns in serum: Comparing datasets from different experiments. Bioinformatics*, *20*, 777–785.

Bonneau, R., & Baker, D. (2001). Ab initio protein structure prediction: Progress and prospects. *Annual Review of Biophysics and Biomolecular Structure*, *30*, 173–189.

Burnside, E. S., Davis, J., Chhatwal, J., Alagoz, O., Lindstrom, M. J., Geller, B. M., Littenberg, B., Kahn, C. E., Shaffer, K., & Page, D. (2009). Unique features of hla-mediated hiv evolution in a mexican cohort: A comparative study. *Radiology*, *251*, 663–672.

Carlson, J., Valenzuela-Ponce, H., Blanco-Heredia, J., Garrido-Rodriguez, D., Garcia-Morales, C., Heckerman, D., et al. (2009). Unique features of hla-mediated hiv evolution in a mexican cohort: A comparative study. *Retrovirology*, *6*(72), 39.

Davis, J., Costa, V. S., Ray, S., & Page, D. (2007a). An integrated approach to feature construction and model building for drug activity prediction. In *Proceedings of the 24th international conference on machine learning (ICML)*.

Davis, J., Ong, I., Struyf, J., Burnside, E., Page, D., & Costa, V. S. (2007b). Change of representation for statistical relational learning. In *Proceedings of the 20th international joint conference on artificial intelligence (IJCAI)*.

DiMaio, F., Kondrashov, D., Bitto, E., Soni, A., Bingman, C., Phillips, G., & Shavlik, J. (2007). Creating protein models from electron-density maps using particle-filtering methods. *Bioinformatics*, *23*, 2851–2858.

Easton, D. F., Pooley, K. A., Dunning, A. M., Pharoah, P. D., et al. (2007). Genome-wide association study identifies novel breast cancer susceptibility loci. *Nature*, *447*, 1087–1093.

Finn, P., Muggleton, S., Page, D., & Srinivasan, A. (1998). Discovery of pharmacophores using the inductive logic programming system progol. *Machine Learning, 30*(1, 2), 241–270.

Friedman, N. (2000). Being Bayesian about network structure. In *Machine Learning*, *50*, 95–125.

Friedman, N., & Halpern, J. (1999). Modeling beliefs in dynamic systems. part ii: Revision and update. *Journal of AI Research*, *10*, 117–167.

Furey, T. S., Cristianini, N., Duffy, N., Bednarski, B. W., Schummer, M., & Haussler, D. (2000). Support vector classification and validation of cancer tissue samples using microarray expression. *Bioinformatics*, *16*(10), 906–914.

Getoor, L., & Taskar, B. (2007). *Introduction to statistical relational learning*. Cambridge, MA: MIT Press.

Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., et al. (1999). Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, *286*, 531–537.

Hardin, J., Waddell, M., Page, C. D., Zhan, F., Barlogie, B., Shaughnessy, J., et al. (2004). Evaluation of multiple models to distinguish closely related forms of disease using DNA microarray data: An application to multiple myeloma. *Statistical Applications in Genetics and Molecular Biology*, *3*(1).

Jain, A. N., Dietterich, T. G., Lathrop, R. H., Chapman, D., Critchlow, R. E., Bauer, B. E., et al. (1994). Compass: A shape-based machine learning tool for drug design. *Aided Molecular Design*, *8*(6), 635–652.

Jones, K. E., Reiser, F. M., Bryant, P. G. K., Muggleton, C. H., Kell, S., King, D. B., et al. (2004). Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427, 247–252.

KDD cup (2001). http://pages.cs.wisc.edu/ dpage/kddcup2001/.

Klösgen, W. (2002). *Handbook of data mining and knowledge discovery, chapter 16.3: Subgroup discovery*. New York: Oxford University Press.

Listgarten, J., Damaraju, S., Poulin, B., Cook, L., Dufour, J., Driga, A., et al. (2004). Predictive models for breast cancer

susceptibility from multiple single nucleotide polymorphisms. *Clinical Cancer Research*, 10, 2725–2737.
Mardis, E. R. (2006). Anticipating the 1,000 dollar genome. *Genome Biology*, *7*(7), 112.
Martin, Y. C., Bures, M. G., Danaher, E. A., DeLazzer, J., Lico, I. I., & Pavlik, P. A. (1993). A fast new approach to pharmacophore mapping and its application to dopaminergic and benzodiazepine agonists. *Journal of Computer Aided Molecular Design*, *8*, 751–758.
McCarty, C., Wilke, R. A., Giampietro, P. F, Wesbrook, S. D., & Caldwell, M. D. (2005). Personalized Medicine Research Project (PMRP): Design, methods and recruitment for a large population-based biobank. *Personalized Medicine*, *2*, 49–79.
Molla, M., Waddell, M., Page, D., & Shavlik, J. (2004). Using machine learning to design and interpret gene expression microarrays. *AI Magazine*, *25*(1), 23–44.
Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *19*(20), 629–679.
Noto, K., & Craven, M. (2006). A specialized learner for inferring structured cis-regulatory modules. *BMC Bioinformatics*, *7*(528), doi:10.1186/1471-2105-7-528.
Oliver, S. G., Young, M., Aubrey, W., Byrne, E., Liakata, M., Markham, M., et al. (2009). The automation of science. *Science*, *324*, 85–89.
Ong, I., Glassner, J., & Page, D. (2002). Modelling regulatory pathways in e.coli from time series expression profiles. *Bioinformatics*, *18*, 241S–248S.
Pe'er, D., Regev, A., Elidan, G., & Friedman, N. (2001). Inferring subnetworks from perturbed expression profiles. *Bioinformatics*, *17*, 215–224.
Perou, C., Jeffrey, S., Van De Rijn, M., Rees, C. A., Eisen, M. B., Ross, D. T., et al. (1999). Distinctive gene expression patterns in human mammary epithelial cells and breast cancers. *Proccedings of National Academy of Science*, *96*, 9212–9217.
Petricoin, E. F., III, Ardekani, A. M., Hitt, B. A., Levine, P. J., Fusaro, V. A., Steinberg, S. M., et al. (2002). Use of proteomic patterns in serum to identify ovarian cancer. *Lancet*, *359*, 572–577.
Rost, B., & Sander, C. (1993). Prediction of protein secondary structure at better than 70 accuracy. *Journal of Molecular Biology*, *232*, 584–599.
Segal, E., Pe'er, D., Regev, A., Koller, D., & Friedman, N. (April 2005). Learning module networks. *Journal of Machine Learning Research*, *6*, 557–588.
Spatola, A., Page, D., Vogel, D., Blondell, S., & Crozet, Y. (1999). Can machine learning and combinatorial chemistry co-exist? In *Proceedings of the American Peptide Symposium*. Kluwer Academic Publishers.
Srinivasan, A. (2001). The aleph manual. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/.
Storey, J. D., & Tibshirani, R. (2003). Statistical significance for genome-wide studies. *Proceedings of the National Academy of Sciences*, *100*, 9440–9445.
The International Warfarin Pharmacogenetics Consortium (IWPC) (2009). Estimation of the Warfarin Dose with Clinical and Pharmacogenetic Data. *The New England Journal of Medicine*, 360:753–764.
Tucker, A., Vinciotti, V., Hoen, P. A. C., Liu, X., & Famili, A. F. (2005). Bayesian network classifiers for time-series microarray data. *Advances in Intelligent Data Analysis VI, 3646*, 475–485.
Van't Veer, L. L., Dai, H., van de Vijver, M. M., He, Y., Hart, A., Mao, M., et al. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, *415*, 530–536.
Waddell, M., Page, D., & Shaughnessy, J., Jr. (2005). Predicting cancer susceptibility from single-nucleotide polymorphism data: A case study in multiple myeloma. *BIOKDD'05: Proceedings of the fifth international workshop on bioinformatics*, Chicago, IL.
Wrobel, S. (1997). An algorithm for multi-relational discovery of subgroups. In *European symposium on principles of kdd* (pp. 78–87). Lecture notes in computer science, Springer, Norway.
Zhang, X., Mesirov, J. P., & Waltz, D. L. (1992). Hybrid system for protein secondary structure prediction. *Journal of Molecular Biology*, *225*, 81–92.
Zou, M., & Conzen, S. D. (2005). A new dynamic Bayesian network approach for identifying gene regulatory networks from time course microarray data. *Bioinformatics*, *21*, 71–79.

Blog Mining

Blog mining is the application of data mining (in particular, Web mining) techniques on blogs, adapted to the content, format, and language of the medium blog. A *blog* is a (more or less) frequently updated publication on the Web, sorted in (usually reverse) chronological order of the constituent blog posts. As in other areas of the Web, mining is applied to the content of blogs, to the various types of links between blogs, and to blog-related behavior. The latter comprises blog authoring including link setting, blog reading and commenting, and querying (often in blog search engines). For more details on blogs and on mining them, see ►text mining for news and blogs analysis.

Boltzmann Machines

Geoffrey Hinton
University of Toronto, ON, Canada

Synonyms

Boltzmann machines

Definition

A Boltzmann machine is a network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be on or off. Boltzmann machines have a simple learning algorithm (Hinton &

Sejnowski, 1983) that allows them to discover interesting features that represent complex regularities in the training data. The learning algorithm is very slow in networks with many layers of feature detectors, but it is fast in "restricted Boltzmann machines" that have a single layer of feature detectors. Many hidden layers can be learned efficiently by composing restricted Boltzmann machines, using the feature activations of one as the training data for the next.

Boltzmann machines are used to solve two quite different computational problems. For a search problem, the weights on the connections are fixed and are used to represent a cost function. The stochastic dynamics of a Boltzmann machine then allow it to sample binary state vectors that have low values of the cost function. For a learning problem, the Boltzmann machine is shown a set of binary data vectors and it must learn to generate these vectors with high probability. To do this, it must find weights on the connections so that relative to other possible binary vectors, the data vectors have low values of the cost function. To solve a learning problem, Boltzmann machines make many small updates to their weights, and each update requires them to solve many different search problems.

Motivation and Background

The brain is very good at settling on a sensible interpretation of its sensory input within a few hundred milliseconds, and it is also very good, over a much longer timescale, at learning the code that is used to express its interpretations. It achieves both the settling and the learning using spiking neurons which, over a period of a few milliseconds, have a state of 1 or 0. These neurons have intrinsic noise caused by the quantal release of vesicles of neurotransmitter at the synapses between the neurons.

Boltzmann machines were designed to model both the settling and the learning, and were based on two seminal ideas that appeared in 1982. Hopfield (1982) showed that a neural network composed of binary units would settle to a minimum of a simple, quadratic energy function provided that the units were updated asynchronously and the pairwise connections between units were symmetrically weighted. Kirkpatrick et al. (1982) showed that systems that were settling to energy minima could find deeper minima if noise was added to the update rule so that the system could occasionally increase its energy to escape from poor local minima.

Adding noise to a Hopfield net allows it to find deeper minima that represent more probable interpretations of the sensory data. More significantly, by using the right kind of noise, it is possible to make the log probability of finding the system in a particular global configuration be a linear function of its energy. This makes it possible to manipulate log probabilities by manipulating energies, and since energies are simple local functions of the connection weights, this leads to a simple, local learning rule.

Structure of Learning System

The learning procedure for updating the connection weights of a Boltzmann machine is very simple, but to understand why it works it is first necessary to understand how a Boltzmann machine models a probability distribution over a set of binary vectors and how it samples from this distribution.

The stochastic Dynamics of a Boltzmann Machine

When unit i is given the opportunity to update its binary state, it first computes its total input, x_i, which is the sum of its own bias, b_i, and the weights on connections coming from other active units:

$$x_i = b_i + \sum_j s_j w_{ij} \tag{1}$$

where w_{ij} is the weight on the connection between i and j, and s_j is 1 if unit j is on and 0, otherwise. Unit i then turns on with a probability given by the logistic function:

$$\text{prob}(s_i = 1) = \frac{1}{1 + e^{-x_i}} \tag{2}$$

If the units are updated sequentially in any order that does not depend on their total inputs, the network will eventually reach a Boltzmann distribution (also called its equilibrium or stationary distribution) in which the probability of a state vector, $\mathbf{v}$, is determined solely by the "energy" of that state vector relative to the energies of all possible binary state vectors:

$$P(\mathbf{v}) = e^{-E(\mathbf{v})} / \sum_{\mathbf{u}} e^{-E(\mathbf{u})} \tag{3}$$

As in Hopfield nets, the energy of state vector $\mathbf{v}$ is defined as

$$E(\mathbf{v}) = -\sum_i s_i^{\mathbf{v}} b_i - \sum_{i<j} s_i^{\mathbf{v}} s_j^{\mathbf{v}} w_{ij} \tag{4}$$

where $s_i^{\mathbf{v}}$ is the binary state assigned to unit i by state vector $\mathbf{v}$.

If the weights on the connections are chosen so that the energies of state vectors represent the cost of those state vectors, then the stochastic dynamics of a Boltzmann machine can be viewed as a way of escaping from poor local optima while searching for low-cost solutions. The total input to unit i, x_i, represents the difference in energy depending on whether the unit is off or on, and the fact that unit i occasionally turns on even if x_i is negative means that the energy can occasionally increase during the search, thus allowing the search to jump over energy barriers.

The search can be improved by using simulated annealing. This scales down all of the weights and energies by a factor, T, which is analogous to the temperature of a physical system. By reducing T from a large initial value to a small final value, it is possible to benefit from the fast equilibration at high temperatures and still have a final equilibrium distribution that makes low-cost solutions much more probable than high-cost ones. At a temperature of 0, the update rule becomes deterministic and a Boltzmann machine turns into a Hopfield network.

Learning in Boltzmann Machines Without Hidden Units

Given a training set of state vectors (the data), the learning consists of finding weights and biases (the parameters) that make those state vectors good. More specifically, the aim is to find weights and biases that define a Boltzmann distribution in which the training vectors have high probability. By differentiating (3) and using the fact that:

$$\partial E(\mathbf{v})/\partial w_{ij} = -s_i^{\mathbf{v}} s_j^{\mathbf{v}} \tag{5}$$

it can be shown that:

$$\left\langle \frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} \right\rangle_{\text{data}} = \langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{model}} \tag{6}$$

where $\langle \cdot \rangle_{\text{data}}$ is an expected value in the data distribution and $\langle \cdot \rangle_{\text{model}}$ is an expected value when the Boltzmann machine samples state vectors from its equilibrium distribution at a temperature of 1. To perform gradient ascent in the log probability that the Boltzmann machine would generate the observed data when sampling from its equilibrium distribution, w_{ij} is incremented by a small learning rate times the RHS of (6). The learning rule for the bias, b_i, is the same as (6), but with s_j omitted.

If the observed data specifies a binary state for every unit in the Boltzmann machine, the learning problem is convex: There are no nonglobal optima in the parameter space. However, sampling from $\langle \cdot \rangle_{\text{model}}$ may involve overcoming energy barriers in the binary state space.

Learning with Hidden Units

Learning becomes much more interesting if the Boltzmann machine consists of some "visible" units whose states can be observed, and some "hidden" units whose states are not specified by the observed data. The hidden units act as latent variables (features) that allow the Boltzmann machine to model distributions over visible state vectors that cannot be modeled by direct pairwise interactions between the visible units. A surprising property of Boltzmann machines is that, even with hidden units, the learning rule remains unchanged. This makes it possible to learn binary features that capture higher-order structure in the data. With hidden units, the expectation $\langle s_i s_j \rangle_{\text{data}}$ is the average, over all data vectors, of the expected value of $s_i s_j$ when a data vector is clamped on the visible units and the hidden units are repeatedly updated until they reach equilibrium with the clamped data vector.

It is surprising that the learning rule is so simple because $\partial \log P(\mathbf{v})/\partial w_{ij}$ depends on all the other weights in the network. Fortunately, the locally available difference in the two correlations in (6) tells w_{ij} everything it needs to know about the other weights. This makes it unnecessary to explicitly propagate error derivatives, as in the backpropagation algorithm.

Different Types of Boltzmann Machine

The stochastic dynamics and the learning rule can accommodate more complicated energy functions (Sejnowski, 1986). For example, the quadratic energy function in (4) can be replaced by an energy function

that has typical term $s_i s_j s_k w_{ijk}$. The total input to unit i that is used in the update rule must then be replaced by

$$x_i = b_i + \sum_{j<k} s_j s_k w_{ijk}. \quad (7)$$

The only change in the learning rule is that $s_i s_j$ is replaced by $s_i s_j s_k$.

Boltzmann machines model the distribution of the data vectors, but there is a simple extension, the "conditional Boltzmann machine" for modeling conditional distributions (Ackley, Hinton, & Sejnowski, 1985). The only difference between the visible and the hidden units is that, when sampling $\langle s_i s_j \rangle_{\text{data}}$, the visible units are clamped and the hidden units are not. If a subset of the visible units are also clamped when sampling $\langle s_i s_j \rangle_{\text{model}}$ this subset acts as "input" units and the remaining visible units act as "output" units. The same learning rule applies, but now it maximizes the log probabilities of the observed output vectors conditional on the input vectors.

Instead of using units that have stochastic binary states, it is possible to use "mean field" units that have deterministic, real-valued states between 0 and 1, as in an analog Hopfield net. Equation (2) is used to compute an "ideal" value for a unit's state, given the current states of the other units, and the actual value is moved toward the ideal value by some fraction of the difference. If this fraction is small, all the units can be updated in parallel. The same learning rules can be used by simply replacing the stochastic, binary values by the deterministic real values (Peterson & Anderson, 1987), but the learning algorithm is hard to justify and the mean field nets have problems in modeling multimodal distributions.

The binary stochastic units used in Boltzmann machines can be generalized to "softmax" units that have more than two discrete values, Gaussian units whose output is simply their total input plus Gaussian noise, binomial units, Poisson units, and any other type of unit that falls in the exponential family (Welling, Rosen-Zvi, & Hinton, 2005). This family is characterized by the fact that the adjustable parameters have linear effects on the log probabilities. The general form of the gradient required for learning is simply the change in the sufficient statistics caused by clamping data on the visible units.

The speed of Learning

Learning is typically very slow in Boltzmann machines with many hidden layers because large networks can take a long time to approach their equilibrium distribution, especially when the weights are large and the equilibrium distribution is highly multimodal, as it usually is when the visible units are unclamped. Even if samples from the equilibrium distribution can be obtained, the learning signal is very noisy because it is the difference of two sampled expectations. These difficulties can be overcome by restricting the connectivity, simplifying the learning algorithm, and learning one hidden layer at a time.

Restricted Boltzmann Machines

A restricted Boltzmann machine (Smolensky, 1986) consists of a layer of visible units and a layer of hidden units with no visible-visible or hidden-hidden connections. With these restrictions, the hidden units are conditionally independent given a visible vector, so unbiased samples from $\langle s_i s_j \rangle_{\text{data}}$ can be obtained in one parallel step. To sample from $\langle s_i s_j \rangle_{\text{model}}$ still requires multiple iterations that alternate between updating all the hidden units in parallel and updating all of the visible units in parallel. However, learning still works well if $\langle s_i s_j \rangle_{\text{model}}$ is replaced by $\langle s_i s_j \rangle_{\text{reconstruction}}$ which is obtained as follows:

1. Starting with a data vector on the visible units, update all of the hidden units in parallel.
2. Update all of the visible units in parallel to get a "reconstruction."
3. Update all of the hidden units again.

This efficient learning procedure approximates gradient descent in a quantity called "contrastive divergence" and works well in practice (Hinton, 2002).

Learning Deep Networks by Composing Restricted Boltzmann Machines

After learning one hidden layer, the activity vectors of the hidden units, when they are being driven by the real data, can be treated as "data" for training another restricted Boltzmann machine. This can be repeated to learn as many hidden layers as desired. After learning multiple hidden layers in this way, the whole network can be viewed as a single, multilayer generative model,

and each additional hidden layer improves a lower bound on the probability that the multilayer model would generate the training data (Hinton, Osindero, & Teh, 2006).

Learning one hidden layer at a time is a very effective way to learn deep neural networks with many hidden layers and millions of weights. Even though the learning is unsupervised, the highest level features are typically much more useful for classification than the raw data vectors. These deep networks can be fine-tuned to be better at classification or dimensionality reduction using the backpropagation algorithm (Hinton & Salakhutdinov, 2006). Alternatively, they can be fine-tuned to be better generative models using a version of the "wake-sleep" algorithm Hinton et al. (2006).

Relationships to Other Models

Boltzmann machines are a type of Markov random field (see ►Graphical Models), but most Markov random fields have simple, local interaction weights which are designed by hand rather than being learned. Boltzmann machines are also like Ising models, but Ising models typically use random or hand-designed interaction weights. The search procedure for Boltzmann machines is an early example of Gibbs sampling, a ►Markov chain Monte Carlo method which was invented independently (Geman & Geman, 1984) and was also inspired by simulated annealing.

Boltzmann machines are a simple type of undirected graphical model. The learning algorithm for Boltzmann machines was the first learning algorithm for undirected graphical models with hidden variables (Jordan, 1998). When restricted Boltzmann machines are composed to learn a deep network, the top two layers of the resulting graphical model form an undirected Boltzmann machine, but the lower layers form a directed acyclic graph with directed connections from higher layers to lower layers, Hinton et al. (2006).

Conditional random fields (Lafferty, McCallum, & Pereira, 2001) can be viewed as simplified versions of higher-order, conditional Boltzmann machines in which the hidden units have been eliminated. This makes the learning problem convex, but removes the ability to learn new features.

Recommended Reading

Ackley, D., Hinton, G., & Sejnowski, T. (1985). A Learning algorithm for boltzmann machines. *Cognitive Science, 9*(1), 147–169.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 6*(6), 721–741.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA, 79*, 2554–2558.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation, 14*(8), 1711–1800.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*, 1527–1554.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*, 504–507.

Hinton, G. E., & Sejnowski, T. J. (1983). Optimal perceptual inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition, Washington, DC* (pp. 448–453).

Jordan, M. I. (1998). *Learning in graphical models*. Cambridge, MA MIT press.

Kirkpatrick, S., Gelatt, D. D., & Vecci, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th international conference on machine learning* (pp. 282–289). San Francisco, Morgan Kaufmann.

Peterson, C., & Anderson, J. R. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems, 1*(5), 995–1019.

Sejnowski, T. J. (1986). Higher-order boltzmann machines. *AIP Conference Proceedings, 151*(1), 398–403.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, & J. L. McClelland (Eds.), *Parallel distributed processing: Vol. 1: Foundations* (pp. 194–281). Cambridge, MA: MIT Press.

Welling, M., Rosen-Zvi, M., & Hinton, G. E. (2005). Exponential family harmoniums with an application to information retrieval. In *Advances in neural information processing systems* (vol. 17, pp. 1481–1488). Cambridge, MA: MIT Press.

Boosting

Boosting is a family of ►ensemble learning methods. The Boosting framework is an answer to a question posed on whether two complexity classes of learning problems are equivalent: *strongly learnable*, and *weakly learnable*. The Boosting framework is a proof by construction that the answer is positive, they are equivalent. The framework allows a "weak" model, only slightly

better than random guessing, to be *boosted* into an arbitrarily accurate *strong* model. ►Adaboost is the most well known and successful of the Boosting family, though there exist many variants specialized for particular tasks, such as cost-sensitive and noise-tolerant versions. See ►ensemble learning for full details.

Bootstrap Sampling

Definition

Bootstrap sampling is a process for creating a distribution of datasets out of a single dataset. It is used in the ►ensemble learning algorithm ►Bagging. It can also be used in ►algorithm evaluation to create a distribution of training sets from which to estimate properties of an algorithm.

Recommended Reading

Davison, A. C., & Hinkley, D. (2006). *Bootstrap methods and their applications* (8th ed.). Cambridge: Cambridge Series in Statistical and Probabilistic Mathematics.

Bottom Clause

Synonyms

Saturation; Starting clause

Definition

The bottom clause is a notion from the field of ►inductive logic programming. It is used to refer to the most specific hypothesis covering a particular example when ►learning from entailment. When learning from entailment, a hypothesis H covers an example e relative to the background theory B if and only if $B \wedge H \models e$, that is, B together with H ►entails the example e. The bottom clause is now the most specific clause satisfying this relationship w.r.t the background theory B and a given example e.

For instance, given the background theory B

```
bird :- blackbird.
bird :- ostrich.
```

and the example e:

```
flies :- blackbird, normal.
```

the bottom clause is H

```
flies :- bird, blackbird, normal.
```

The bottom clause can be used to constrain the search for clauses covering the given example because all clauses covering the example relative to the background theory should be more general than the bottom clause. The bottom clause can be computed using ►inverse entailment.

Cross References

►Entailment
►Inductive Logic Programming
►Inverse Entailment
►Logic of Generality

Bounded Differences Inequality

►McDiarmid's Inequality

BP

►Backpropagation

Breakeven Point

More accurately described as *precision–recall* BEP, it is an evaluation measure originally introduced in the field of information retrieval to evaluate retrieval systems that return a list of documents ordered by their supposed relevance to the user's information need (see also ►Document Classification). It can also be used to evaluate any classification model f that addresses a two-class classification problem but outputs real-valued predictions $f(x)$ instead of binary ones. To use such a classifier in practice, one would select a threshold θ and predict an instance x to be positive if $f(x) > \theta$ and negative otherwise. Thus, the ►precision and ►recall of this system depend on the choice of the threshold θ. A lower threshold means higher recall, but usually also lower precision. At some point (when the number of instances predicted to be positive is the same as the actual number

of positive instances), precision and recall are equal; this value of precision and recall is known as the *precision–recall BEP*. It is a useful measure of the quality of our classifier because it gives us guidance into what sort of tradeoffs are available to the user of such a classifier via the choice of threshold: if we want a precision above the BEP, we must accept that our recall will be below the BEP, and vice versa. A different meaning of the term "breakeven point" is sometimes used in ROC (►ROC Analysis), where the *ROC breakeven* is defined as the point where the true positive rate and the false positive rate sum to 1; smaller values of the ROC breakeven are better than larger ones. Informally, the ROC breakeven measures how close the ROC curve gets to the "ROC sweet spot" in the top left corner (where the ►true positive rate is 1 and the ►false positive rate is 0).

C

C4.5

►Decision Tree

Cannot-Link Constraint

A pairwise constraint between two items indicating that they should be placed into different clusters in the final partition.

Candidate-Elimination Algorithm

Mitchell's, (1982, 1997) candidate-elimination algorithm performs a bidirectional search in the ►hypothesis space. It maintains a set, *S*, of most specific hypotheses that are consistent with the training data and a set, *G*, of most general hypotheses consistent with the training data. These two sets form two boundaries on the version space. See ►Learning as Search.

Recommended Reading

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence, 18*(2), 203–226.
Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.

Cascade-Correlation

Thomas R. Shultz[1], Scott E. Fahlman[2]
[1]McGill University, Montréal, QC, Canada
[2]Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Cascor; CC

Definition

Cascade-Correlation (often abbreviated as "Cascor" or "CC") is a ►supervised learning algorithm for ►artificial neural networks. It is related to the ►back-propagation algorithm ("backprop"). CC differs from backprop in that a CC network begins with no hidden units, and then adds units one-by-one, as needed during learning.

Each new hidden unit is trained to correlate with residual error in the network built so far. When it is added to the network, the new unit is frozen, in the sense that its input weights are fixed. The hidden units form a *cascade*: each new unit receives weighted input from all the original network inputs and from the output of every previously created hidden unit. This cascading creates a network that is as deep as the number of hidden units. Stated another way, the CC algorithm is capable of efficiently creating complex, higher-order nonlinear basis functions – the hidden units – which are then combined to form the desired outputs.

The result is an algorithm that learns complex input/output mappings very fast compared to backprop, and that builds a multi-layer network structure that is customized for the problem at hand.

Motivation and Background

Cascade-Correlation was designed (Fahlman & Lebiere, 1990) to address two well-known problems with the popular back-propagation algorithm ("backprop"). First, a backprop user has to guess what network structure – the number of hidden layers and the number of units in each layer – would be best for a given learning problem. If the network is too small or too shallow, it won't solve the problem; if it is too large or too deep, training is very slow, and the network is prone to overfitting the training data. Because there is no reliable way to choose a good structure before training begins, most backprop users have to train many different structures before finding one that is well-matched to the task.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Second, even if a backprop user manages to choose a good network structure, training is generally very slow. That is particularly true in networks with many hidden units or with more than one hidden layer. One cause of slow learning in backprop is the use of a uniform learning-rate parameter for updating network weights. This problem was addressed with the Quickprop algorithm (Fahlman, 1988), an approximation to Newton's method that adapts the learning rate for each weight parameter depending on the first two derivatives of the local error surface. Quickprop improved learning speed, sometimes dramatically, but learning was still too slow in large or deep networks.

Another cause of slow learning in backprop is the "herd effect" (Fahlman & Lebiere, 1990). If the solution to a network problem requires, say, 30 hidden units, each of these units must be trained to do a different job – that is, to compute a different nonlinear basis function. Each hidden unit starts with a different and randomly chosen set of input weights; but if the units are all trained at once, they all see the same error signal. There is no central authority telling each unit to do a separate job, so they tend to drift toward the same part of parameter space, forming a herd that moves around together. Eventually, the units may drift apart and begin to differentiate, but there is nothing to compel this, so the process is slow and unreliable. Usually, in selecting an initial topology for a backprop net, it is necessary to include many extra hidden units to increase the odds that each job will be done by some unit.

CC addresses this problem by introducing and training hidden units one by one. Each hidden unit sees a strong, clear error gradient, not confused by the simultaneous movement of other hidden units. A new hidden unit can thus move quickly and decisively to a position in parameter space where it can perform a useful function, reducing the residual error. One by one, cascor-hidden units take up distinct jobs, instead of milling about together competing to do the same job.

Structure of Learning System

The Algorithm

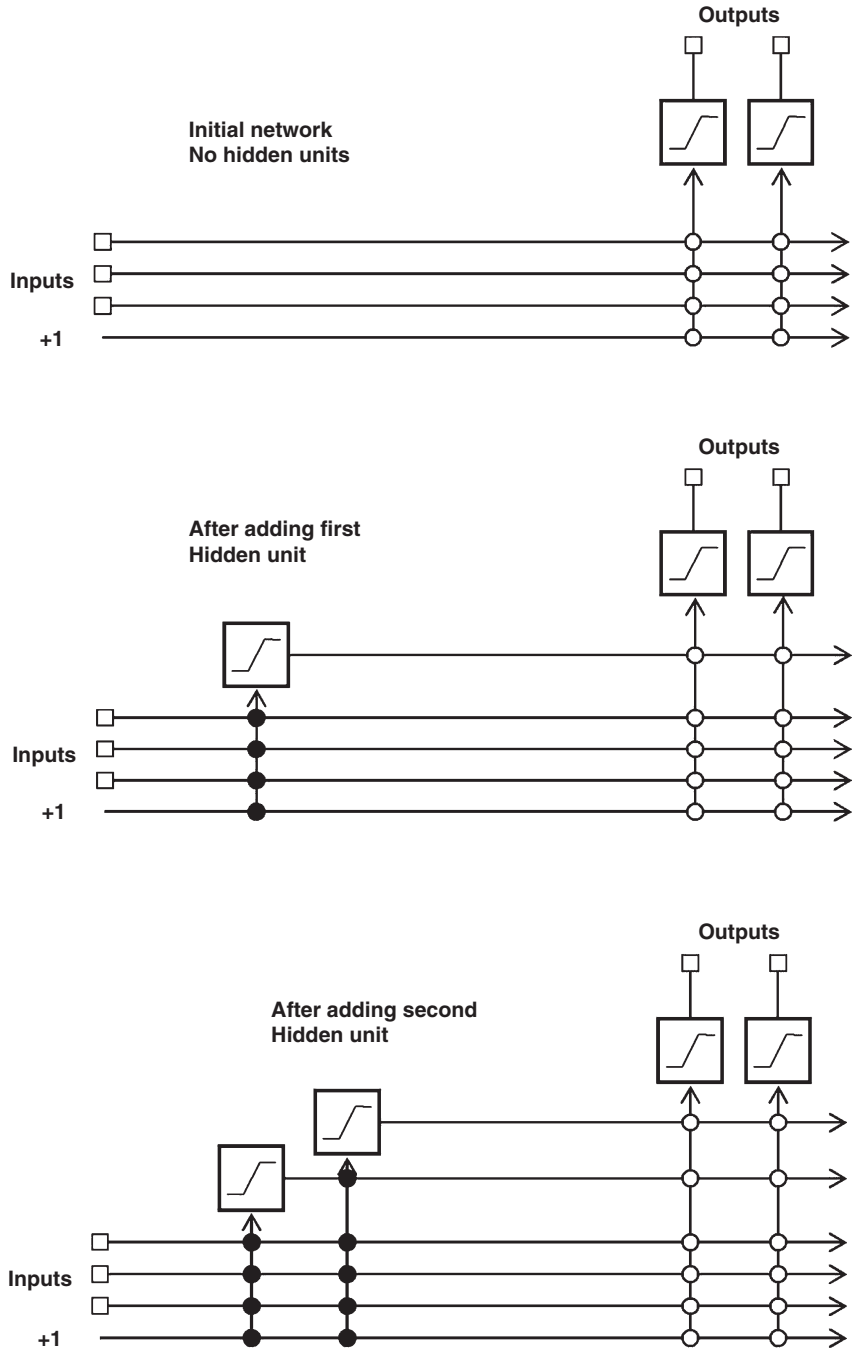
The CC architecture is illustrated in Fig. 1. It begins with some inputs and one or more output units, but no hidden units. The numbers of inputs and outputs are dictated by the problem. As in backprop, the output units generally have a sigmoid activation function, but could alternatively have a linear activation function. Every input is connected to every output unit by a connection with an adjustable weight. There is also a *bias* input, permanently set to +1.

Hidden units are added to the network one by one. Each new hidden unit receives a weighted connection from each of the network's original inputs and also from every existing hidden unit. Each new unit therefore adds a new single-unit layer to the network. This makes it possible to create high-order nonlinear feature detectors, customized for the problem at hand.

As noted, learning begins without hidden units. The direct input–output connections are trained as well as possible over the entire set of training examples, using Quickprop. At some point, this training approaches an asymptote. When no significant error reduction has occurred after a certain number of training cycles, this output phase is terminated and there is a shift to input phase to recruit a new hidden unit, using the unit-creation algorithm to be described. The new unit is added to the net, its input weights are frozen, and all the output weights are once again trained using Quickprop. This cycle repeats until the error is acceptably small, in the sense that all network outputs for all training patterns are within a specified threshold of their target values.

To create a new hidden unit, input phase begins with several *candidate units* that receive trainable input connections from all of the network inputs and from all existing hidden units. The outputs of these candidates are not yet connected to the network. There are a number of passes over the examples of the training set, adjusting the candidate unit's input weights after each pass. The goal of these adjustments, using Quickprop, is to maximize the correlation between each candidate's output and the residual error.

When these correlation measures show no further significant improvement, input phase stops, the best-correlating candidate's input weights are frozen, and that unit is installed in the network. The remaining candidates are discarded and the algorithm then retrains the output weights, making use of this new feature as well as all the old ones. As the new unit's output correlates well with some component of the residual error, its output weights can be quickly adjusted to reduce that



Cascade-Correlation. Figure 1. The Cascade–Correlation (CC) architecture, as new hidden units are added. Black circles are frozen connection weights, white circles are weights trained during output-training phase. The vertical lines sum all incoming activation

component. So after adding each new hidden unit, the network's residual error should be smaller than before.

Using several candidates, each with differently-initialized input weights, greatly reduces the chances of installing a bad hidden unit that gets the network stuck in a local optimum far from the global optimum value. All candidates receive the same input signals and see the same residual error for each training pattern. Because they do not interact with one another or affect the network during training, these candidates can be trained in parallel. In a pool of four to eight candidates, there are almost always several high-quality candidates with nearly equal correlation values.

Hidden units continue to be recruited until network error reaches an acceptable level, or until cross-validation signals a stop. Because only a single layer of weights is adjusted at a time, rather than back-propagating an error signal through several layers of shifting units, CC training proceeds very quickly.

Performance

CC is designed to produce a network just large enough to solve the problem, and to do so much faster than backprop and related algorithms. In many reported cases that require hidden units, CC learns the desired behavior 10–100 times faster than standard backprop (Fahlman & Lebiere, 1990). One striking example of this is the *two-spirals problem*, an artificial benchmark designed to be very difficult for neural networks with sigmoid units. At the time CC was developed, the best known backprop solutions for two-spirals required a network with three hidden layers of five units each. CC typically solves this problem with 12 hidden units, and has found solutions with as few as nine hidden units. In terms of runtime, CC training was about 50 times faster than standard backprop and 23 times faster than Quickprop used within a static network.

Variants of Cascade-Correlation

Flat Cascade-Correlation In standard CC, each new hidden unit receives inputs from every existing unit, so the net becomes one level deeper every time a unit is added. This is a powerful mechanism, creating increasingly complex feature detectors as the network learns. But sometimes this added depth is not required for the problem, creating a very deep network that performs no better than a shallow one. The resulting network might have more weights than are required for the problem, raising concern about over-fitting. Another concern was that the cascaded non-linearity of CC might also compromise generalization. To address these concerns, a flat variant of cascor adds new recruited units onto a single layer (i.e., cascaded connections are eliminated), limiting the depth of the network and eliminating all cascaded weights between hidden units.

Comparison of flat to standard CC on generalization in particular learning problems yielded inconsistent results, but a more problem–neutral, student–teacher approach found no generalization differences between flat and standard versions of CC (Dandurand, Berthiaume, & Shultz, 2007). Here, flat and standard student CC networks learned the input–output mappings of other, randomly initialized flat and standard CC teacher networks, where task complexity was systematically manipulated. Both standard and flat CC student networks learned and generalized well on problems of varying complexity. In low-complexity tasks, there were no significant performance differences between flat and standard CC student networks. For high-complexity tasks, flat CC student networks required fewer connection weights and learned with less computational cost than did standard CC student networks.

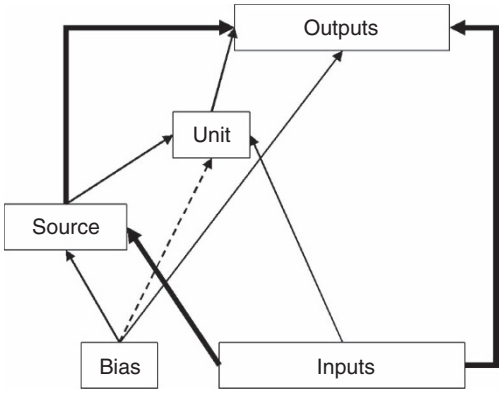
Sibling-Descendant Cascade-Correlation (SDCC) SDCC (Baluja & Fahlman, 1994) provides a more general solution to the problem of network depth. In the candidate pool there are two kinds of candidate units: *descendant* units that receive inputs from all existing hidden units, and *sibling* units that receive the same inputs as the deepest hidden units in the current net. When the time comes to choose a winning candidate, the candidate with the best correlation wins, but there is a slight preference for sibling units. So unless a descendant unit is clearly superior, a sibling unit will be recruited, making the active network larger, but not deeper. In problems where standard CC produces a network with 15 or 20 hidden units and an equal number of layers, SDCC often produces a network with only two or three hidden layers.

Recurrent Cascade-Correlation (RCC) Standard CC produces a network that maps its *current* inputs to outputs. The network has no memory of recent inputs, so this

architecture is not able to learn to recognize a sequence of inputs. In the RCC algorithm, each candidate and hidden unit takes the same inputs as in standard CC, but it also takes an additional input: the unit's own previous output, delayed by one time interval (Fahlman, 1991). The weight on this time-delayed input is trained by the same algorithm as all the other inputs.

This delayed loop gives RCC networks a way of remembering past inputs and internal states, so they can learn to recognize sequences of input patterns. In effect, the architecture builds a finite-state machine tailored specifically to recognize the pattern sequences in the training set. For example, an RCC net learned to recognize characters in Morse code.

Knowledge-Based Cascade-Correlation (KBCC) KBCC is a variant that can recruit previously-learned networks or indeed any differentiable function, in competition with single hidden units (Shultz & Rivest, 2001; Shultz, Rivest, Egri, Thivierge, & Dandurand, 2007). The recruit is the candidate whose output correlates best with residual network error, just as in ordinary CC. The candidate pool usually has a number of randomly initialized sigmoid units and a number of candidate source networks, i.e., networks previously trained on other tasks. The input weights to multiple copies of the source networks are usually randomly initialized to improve optimization. Of these copies, one is typically connected with an identity matrix with off-diagonal zeros, to enable quick learning of the target task when exact knowledge is available. A hypothetical KBCC network is shown in Fig. 2.



Cascade-Correlation. Figure 2. Hypothetical knowledge-based cascade-correlation (KBCC) network that has recruited a source network and then a sigmoid unit, each installed on a separate layer. The dashed line represents a single connection weight, thin solid lines represent weight vectors, and thick solid lines represent weight matrices

Software Most CC algorithms are available in a variety of formats and languages, including:

CASCOR: Lisp and C implementations of Cascade-correlation
http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/systems/cascor/0.html
Free Lisp and C implementations of cascade-correlation.

Cascade Neural Network Simulator
http://www.cs.cmu.edu/~sef/sefSoft.htm
A public domain C program that implements cascade-correlation and recurrent cascade-correlation, plus experimental versions of cascade 2 and recurrent cascade 2.

LNSC Cascade-correlation Simulator Applet
http://www.psych.mcgill.ca/perpg/fac/shultz/cdp/lnsc_applet.htm
A Java applet allowing direct comparisons of cascade-correlation and back-propagation algorithms on some benchmark problems, also permitting entry of text-edited custom training and test patterns.

LNSC Java Code Library
http://www.lnsclab.org/
Free compiled Java versions of BP, CC, SDCC, and KBCC neural-network software, along with a tutorial

Applications

CC

Partly because of its ability to grow its own networks and build new learning on top of existing knowledge, CC has been used to simulate many phenomena in cognitive development. These characteristics embody the constructivism that developmental psychologists often discussed but did not formulate precisely. Simulations are typically evaluated by how well they capture the various psychological phenomena that characterize a particular domain.

The balance-scale task involves presenting a child with a rigid beam balanced on a fulcrum with pegs spaced at equal intervals to the left and right of the fulcrum. A number of identical weights are placed on a peg on the left side and a peg on the right side, and the child is asked to predict which side will descend when the beam is released from its moorings. CC networks passed through the stages observed with children and captured the so-called torque-difference effect, the tendency to do better on problems with large absolute torque differences than on problems with small torque differences (Shultz, Mareschal, & Schmidt, 1994; Shultz and Takane, 2007).

The conservation task presents a child with two quantities of objects that the child judges to be equal and then transforms one set in a way that either changes that relationship or conserves it. CC networks captured four important conservation regularities (Shultz, 1998):

1. A shift from nonconservation to conservation beliefs
2. A sudden spurt in performance during acquisition
3. Emergence of correct conservation judgments for small quantities before larger quantities
4. Young children's choice of the longer row as having more items than the shorter row

Analysis of network solutions at various points in development revealed a gradual shift from perceptual (how the sets of items look) to cognitive (whether or not the transformation changed a quantity) solutions, similar to what had been found with children.

The seriation task requires a child to order a disordered collection of sticks of different lengths. CC networks passed through the four stages seen in children (total failure, partial sort, trial-and-error sort, and systematic sort) and captured the tendency for sets with smaller differences to be more difficult to sort (Mareschal & Shultz, 1999). Analysis of network solutions revealed early success at the short end of the series that was gradually extended to the longer end, as in children.

The transitivity problem typically also employs sticks of different length. Here the child is trained on all pairs of sticks that are adjacent in length and then is asked to infer the relative length of untrained pairs. Five psychological regularities were captured when CC networks were trained to compare the relative sizes of adjacent pairs (Shultz & Vogel, 2004):

1. Learning short or long adjacent pairs before adjacent pairs of medium length.
2. Faster inferences with pairs farther apart in length than with pairs close together in length, an effect that diminished with age. A constraint-satisfaction network module simulated reaction times by inputting the output of a CC network and settling over time cycles into a low-energy solution that satisfied the constraints supplied by connection weights and inputs, effectively cleaning up the output of the CC network.
3. Faster inferences with pairs containing the shortest or longest stick.
4. Faster inferences when the expression used in the question (e.g., shorter) is compatible with an end stick (e.g., the shortest stick) in the compared pair than when the question term (e.g., shorter) is incompatible with an end stick (e.g., the longest stick) in the compared pair.
5. Older children learned adjacent pairs faster and made inference comparisons faster and more accurately than did young children.

The computational bases for these effects were revealed by examining the pattern of connection weights within the CC network module. The pattern of these weights formed a cubic shape, symmetrical for the two sticks being compared, in which discrimination was better at the ends of the array than in the middle and became sharper with deeper learning.

Another task calls for integration of cues for moving objects, governed by the equation velocity = distance/time. Children were presented with information on two of those quantities and asked to infer the third. Three stages involved first using the quantity that varied positively with the quantity to be inferred, second adding or subtracting the known quantities, and finally multiplying or dividing the known quantities. Already documented stages were captured and others were correctly predicted by CC networks (Buckingham & Shultz, 2000).

Semantic rules for deictic personal pronouns specify that *me* refers to the person using the pronoun and *you* refers to the person who is being addressed. Although

most children acquire these pronouns without notable errors, a few reverse these pronouns, persistently calling themselves *you* and the mother *me*. Such reversals in children are produced by lack of opportunity to overhear these pronouns used by other people, where the shifting reference can be observed. CC networks covered these phenomena and generated predictions for effective therapy to correct reversal errors (Oshima-Takane, Takane, & Shultz, 1999).

Discrimination shift learning tasks repeatedly present pairs of stimuli with mutually exclusive attributes on several binary dimensions, such as color, shape, and position, and a child learns to select the correct stimulus in each pair, e.g., *square*. Feedback is given and learning continues until the child reaches a success criterion, e.g., 8/10 correct. Then reward contingencies shift, usually without warning. A *reversal* shift stays within the initially relevant dimension, e.g., from *square* to *circle*. A *nonreversal* shift is to another dimension, such as from *square* to *blue*. There are related tasks that use new stimulus values in the shift phase. These are called *intradimensional* shifts if the shift remains within the initial dimension, e.g., *square* to *triangle*, or *extradimensional* if there is a change to another dimension, e.g., from *square* to *yellow*. The *optional shift* task presents only two stimulus pairs in the shift phase, making it ambiguous whether the shift is a reversal or nonreversal shift. The pattern of subsequent choices allows determination of whether the child interprets this as a reversal or a nonreversal shift.

Age differences in the large literature on these shifts indicate that older children learn a reversal shift faster than a nonreversal shift, learn an intradimensional shift faster than an extradimensional shift, make a reversal shift in the optional task, and are initially impaired on unchanged pairs during a nonreversal shift. Younger children learn reversal and nonreversal shifts equally fast, learn an intra-dimensional shift faster than an extra-dimensional shift, make a nonreversal shift in the optional task, and are unimpaired on unchanged pairs during a nonreversal shift. These findings were simulated by CC networks (Sirois & Shultz, 1998), which also generated predictions that were later confirmed.

When infants repeatedly experience stimuli from a particular class, their attention decreases, but it recovers to stimuli from a different class. This familiarize-and-test paradigm is responsible for most of the discoveries of infant psychological abilities. CC networks simulated findings on infant attention to syntactic patterns in an artificial language (Shultz & Bale, 2006) and age differences in infant categorization of visual stimuli (Shultz & Cohen, 2004), and generated several predictions, some of which were tested and confirmed.

SDCC

Because of SDCC's ability to create a variety of network topologies, it is beginning to be used in psychology simulations: infant learning of word-stress patterns in artificial languages (Shultz & Bale, 2006), syllable boundaries (Shultz & Bale, 2006), visual concepts (Shultz, 2006), and false-belief tasks; learning the structure of mathematical groups (Schlimm & Shultz, 2009); replication of the results of the CC simulation of conservation acquisition (Shultz, 2006); and concept acquisition.

CC and SDCC networks capture developmental stages by growing in computational power and by being sensitive to statistical patterns in the training environment (Shultz, 2003). The importance of growth was demonstrated by comparisons with static backprop networks, designed with the same final topology as successful CC networks, that learn only by adjusting connection weights (Shultz, 2006). Coupled with the variety of successful SDCC topologies, this suggests that the constructive process is more important than precise network topologies. Capturing stages is challenging because the system has to not only succeed on the task but also make the same mistakes on the road to success that children do. CC and SDCC arguably produced the best data coverage of any models applied to the foregoing phenomena. Both static and constructive networks capture various perceptual effects by virtue of their sensitivity to quantitative variation in stimulus inputs (Shultz, 2003).

Comparison of the two algorithms in psychological modeling indicates that SDCC provides the same functionality as CC but with fewer connection weights and shallower and more variable network topologies (Shultz, 2006).

KBCC

KBCC also has potential for simulating psychological development, but it has so far been applied mainly to toy and engineering problems. Exploration of a

variety of toy problems was important in understanding the behavior of this complex algorithm. Some toy problems involved learning about two-dimensional geometric shapes under various transformations such as translation, rotation, and size changes, as well as compositions of complex shapes from simpler shapes (Shultz & Rivest, 2001). Networks had to learn to distinguish points within a target shape from points outside the shape. Learning time without relevant knowledge was up to 16 times longer than with relevant knowledge on these problems. There was a strong tendency to recruit relevant knowledge whenever it was available. Direct comparison revealed that KBCC learned spatial translation problems faster than Multitask Learning networks did.

Parity problems require a network to activate an output unit only when an odd number of binary inputs are activated. When parity-4 networks were included in the candidate source pool, KBCC learned parity-8 problems (with eight binary inputs) faster and with fewer recruits than did CC networks. Parity-4 networks were recruited by these KBCC target networks whenever available.

KBCC also learned complex chessboard shapes from knowledge of simpler chessboards. As with parity, networks used simpler previous knowledge to compose a solution to a more complex problem and learning was speeded accordingly.

In a more realistic vein, KBCC networks recruiting knowledge of vowels from one sort of speaker (e.g., adult females) learned to recognize vowels spoken by other sets of speakers (e.g., children and adult males) faster than did knowledge-free networks.

KBCC learned an efficient algorithm for detecting prime numbers by recruiting previously-learned knowledge of divisibility (Shultz et al., 2007). This well-known detection algorithm tests the primality of an integer n by checking if n is divisible by any integers between 2 and the integer part of $\sqrt{n}$. Starting with small primes is efficient because the smaller the prime divisor, the more composites are detected in a fixed range of integers. The candidate pool contained networks that had learned whether an integer could be divided by each of a range of integers, e.g., a divide-by-2 network, a divide-by-3 network, etc., up to a divisor of 20. KBCC target networks trained on 306 randomly-selected integers from 21 to 360 recruited only source networks involving prime divisors below the square root of 360, in order from small to large divisors. KBCC avoided recruiting single hidden units, source networks with composite divisors, any divisors greater than the square root of 360 even if prime, and divisor networks with randomized connection weights. KBCC never recruited a divide-by-2 source network because it instead learned to check the last binary digit of n to determine if n was odd or even, an effective shortcut to dividing by 2. Such KBCC networks learned the training patterns in about one third the time required by knowledge-free networks, with fewer recruits on fewer network layers, and they generalized almost perfectly to novel test integers. In contrast, even after mastering the training patterns, CC networks generalized less well than automatic guessing that the integer was composite, which was true for 81% of integers in this range. As predicted by the simulation, adults testing primality also used mainly prime divisors below $\sqrt{n}$ and ordered divisors from small to large.

This work underscores the possibility of neural-network approaches to compositionality because KBCC effectively composed a solution to prime-number detection by recruiting and organizing previously learned parts of the problem, in the form of divisibility networks.

Future Directions

One new trend is to inject symbolic rules or functions into KBCC source networks. This is similar to KBANN, but more flexible because a KBCC target network decides whether and how to recruit these functions. This provides one method of integrating symbolic and neural computation and allows for simulation of the effects of direct instruction.

Cross References

►Artificial Neural Networks
►Backpropagation

Recommended Reading

Baluja, S., & Fahlman, S. E. (1994). *Reducing network depth in the cascade-correlation learning architecture.* Pittsburgh, PA: School of Computer Science, Carnegie Mellon University.

Buckingham, D., & Shultz, T. R. (2000). The developmental course of distance, time, and velocity concepts: A generative connectionist model. *Journal of Cognition and Development, 1,* 305–345.

Dandurand, F., Berthiaume, V., & Shultz, T. R. (2007). A systematic comparison of flat and standard cascade-correlation using a student-teacher network approximation task. *Connection Science, 19*, 223–244.

Fahlman, S. E. (1988). Faster-learning variations on back-propagation: An empirical study. In D. S. Touretzky, G. E. Hinton, & T. J. Sejnowski (Eds.), *Proceedings of the 1988 connectionist models summer school* (pp. 38–51). Los Altos, CA: Morgan Kaufmann.

Fahlman, S. E. (1991). The recurrent cascade-correlation architecture. In D. S. Touretzky (Ed.), *Advances in neural information processing systems*. (Vol. 3) Los Altos CA: Morgan Kaufmann.

Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), *Advances in neural information processing systems* (Vol. 2, pp. 524–532). Los Altos, CA: Morgan Kaufmann.

Mareschal, D., & Shultz, T. R. (1999). Development of children's seriation: A connectionist approach. *Connection Science, 11*, 149–186.

Oshima-Takane, Y., Takane, Y., & Shultz, T. R. (1999). The learning of first and second pronouns in English: Network models and analysis. *Journal of Child Language, 26*, 545–575.

Schlimm, D., & Shultz, T. R. (2009). Learning the structure of abstract groups. In N. A. Taatgen & H. V. Rijn (Eds.), *Proceedings of the 31st annual conference of the cognitive science society* (pp. 2950–2955). Austin, TX: Cognitive Science Society.

Shultz, T. R. (1998). A computational analysis of conservation. *Developmental Science, 1*, 103–126.

Shultz, T. R. (2003). *Computational developmental psychology*. Cambridge, MA: MIT Press.

Shultz, T. R. (2006). Constructive learning in the modeling of psychological development. In Y. Munakata & M. H. Johnson (Eds.), *Processes of change in brain and cognitive development: Attention and performance XXI* (pp. 61–86). Oxford, UK: Oxford University Press.

Shultz, T. R., & Bale, A. C. (2006). Neural networks discover a near-identity relation to distinguish simple syntactic forms. *Minds and Machines, 16*, 107–139.

Shultz, T. R., & Cohen, L. B. (2004). Modeling age differences in infant category learning. *Infancy, 5*, 153–171.

Shultz, T. R., Mareschal, D., & Schmidt, W. C. (1994). Modeling cognitive development on balance scale phenomena. *Machine Learning, 16*, 57–86.

Shultz, T. R., & Rivest, F. (2001). Knowledge-based cascade-correlation: Using knowledge to speed learning. *Connection Science, 13*, 1–30.

Shultz, T. R., Rivest, F., Egri, L., Thivierge, J.-P., & Dandurand, F. (2007). Could knowledge-based neural learning be useful in developmental robotics? The case of KBCC. *International Journal of Humanoid Robotics, 4*, 245–279.

Shultz, T. R., & Takane, Y. (2007). Rule following and rule use in simulations of the balance-scale task. *Cognition, 103*, 460–472.

Shultz, T. R., & Vogel, A. (2004). A connectionist model of the development of transitivity. In *Proceedings of the twenty-sixth annual conference of the cognitive science society* (pp. 1243–1248). Mahwah, NJ: Erlbaum.

Sirois, S., & Shultz, T. R. (1998). Neural network modeling of developmental effects in discrimination shifts. *Journal of Experimental Child Psychology, 71*, 235–274.

CART

►Decision Tree

Cascor

►Cascade-Correlation

Case

►Instance

Case-Based Learning

►Instance-Based Learning

Case-Based Reasoning

Susan Craw
The Robert Gordon University, Scotland, UK

Synonyms

CBR; Experience-based reasoning; Lessons-learned systems; Memory-based learning

Definition

Case-based reasoning solves problems by retrieving similar, previously solved problems and reusing their solutions. Experiences are memorized as cases in a case base. Each experience is learned as a problem or situation together with its corresponding solution or action. The experience need not record *how* the solution was reached, simply that the solution was used for the problem. The case base acts as a memory, and remembering is achieved using similarity-based retrieval and reuse of the retrieved solutions. Newly solved problems may be retained in the case base and so the memory is able to grow as problem-solving occurs.

Motivation and Background

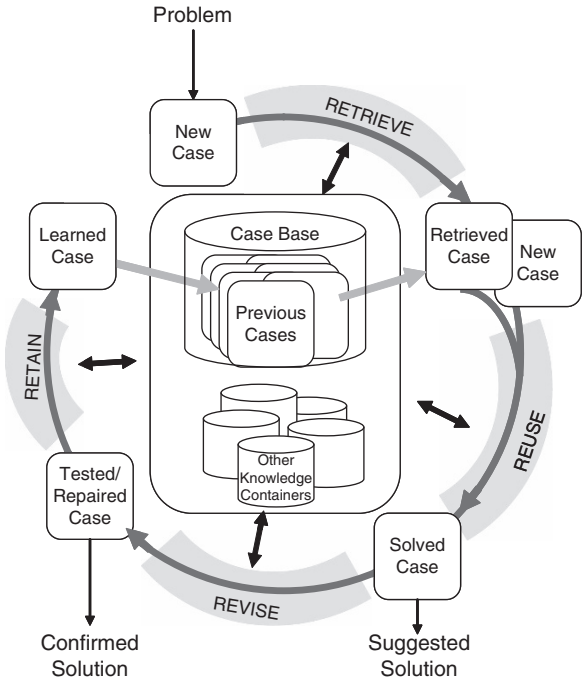
Case-based reasoning (CBR) is inspired by memory-based human problem-solving in which instances of earlier problem-solving are remembered and applied to solve new problems. For example, in Case Law, the decisions in trials are based on legal precedents from previous trials. In this way specific experiences are memorized, and remembered and reused when appropriate (Leake, 1996). This contrasts with rule-based or theory-based problem-solving in which knowledge of *how* to solve a problem is applied. A doctor diagnosing a patient's symptoms may apply knowledge about how diseases manifest themselves, or she may remember a previous patient who demonstrated similar symptoms. Schank's ►dynamic memory model was highly influential in early CBR systems (Kolodner, 1993; Riesbeck & Schank, 1989). Its emphasis on the use of specific experiences to underpin problem-solving and enable learning is replicated in CBR.

The fundamental assumption of CBR is that *Similar problems have similar solutions*. For example, a patient with similar symptoms will have the same diagnosis, the price of a house with similar accommodation and location will be similar, the design for a kitchen with a similar shape and size can be reused, a journey plan for a nearby destination is similar to the earlier trip. A related assumption is that the world is a regular place, and what holds true today will probably hold true tomorrow. A further assumption relevant to memory is that situations repeat, because if they do not, there is no point in remembering them!

CBR is an example of ►Lazy Learning because there is no learned model to apply to solve new problems. Instead, the generalization needed to solve unseen problems happens when a new problem is presented and the similarity-based retrieval identifies relevant previous experiences. The lack of a learned model and the reliance on stored experiences mean that CBR is particularly relevant in domains which are ill-defined, not well understood, or where no underlying theory is available.

Structure of the Learning System

Figure 1 shows the structure of a CBR system (Aamodt & Plaza, 1994). A case base of Previous Cases is the primary knowledge source in a CBR system, with additional knowledge being used to identify similar cases in the RETRIEVE stage, and to REUSE and REVISE the Retrieved Case. A CBR system learns as it solves new problems when a Learned Case is created from the New Case and its Confirmed Solution, and RETAINed as a new case in the case base.



Case-Based Reasoning. Figure 1. CBR system (adapted from Aamodt and Plaza (1994))

Knowledge Containers

Case knowledge is the primary source of knowledge in a CBR system. However, case knowledge is only one of four knowledge containers identified by Richter and Aamodt (2005).

- Vocabulary: The representation language used to describe the cases captures the concepts involved in the problem-solving.
- Similarity Knowledge: The similarity measure defines how the distances between cases are computed so that the nearest neighbors are identified for retrieval.
- Adaptation Knowledge: Reusing solutions from retrieved cases may require some adaptation to enable them to fit the new problem.
- Case Base: The stored cases capture the previous problem-solving experiences.

The content of each knowledge container is not fixed and knowledge in one container can compensate for a lack of knowledge in another. It is easy to see that a more sophisticated knowledge representation could be less demanding on the content of the case base. Similarly, vocabulary can make similarity assessment during retrieval easier, or a more complete case base could reduce the demands on adaptation during reuse. Further knowledge containers (e.g., maintenance) are proposed by others.

Cases may be represented as simple feature vectors containing nominal or numeric values. A case capturing a whisky tasting experience might contain features such as sweetness, color, nose, palate, and the ▶classification as the distillery where it was made.

Sweetness	Peatiness	Color	Nose	Palate	Distillery
6	5	amber	full	medium-dry	Dalmore

More structured representations can use frame-based or object-oriented cases. The choice of representation depends on the complexity of the experiences being remembered and is influenced by how similarity should be determined. Hierarchical case representations allow cases to be remembered at different levels of abstraction, and retrieval and reuse may occur at these different levels (Bergmann & Wilke, 1996).

In ▶classification, the case base can be considered to contain exemplars of problem-solving. Aha et al.'s (1991) family of instance-based learning algorithms IB1, IB2 and IB3 apply increasingly informed selection methods to determine whether a classification experience should become part of the case base. IB1 simply remembers all experiences, IB2 stores an experience only if it would be wrongly solved by the existing stored experiences, and IB3 keeps a score for the reuse of each experience and discards those whose classification success is poor. This notion of exemplar confirms a CBR case base as a source of knowledge; it contains only those experiences that are believed to be useful for problem-solving. A similar view is taken for non-classification domains; for example, the case base contains useful designs that can be used for re-design, and plans for re-planning.

One of the advantages of CBR is that a case base is composed of independent cases that each captures some local problem-solving knowledge that has been experienced. Therefore, the "knowledge acquisition bottleneck" of many rule-based and model-based systems is reduced for CBR. However, the Other Knowledge Containers provide additional knowledge acquisition demands that may lessen the advantage of CBR for some domains.

CBR Cycle

Aamodt and Plaza (1994) propose a four-stage CBR cycle for problem-solving and learning (Fig. 1). It is commonly referred to as the "Four REs" or "R^4" cycle to recognize the following stages.

- RETRIEVE: The cases that are most similar to the New Case defined by the description of the new problem are identified and retrieved from the case base. The RETRIEVE stage uses the similarity knowledge container in addition to the case base.
- REUSE: The solutions in the Retrieved (most similar) Cases are reused to build a Suggested Solution to create the Solved Case from the New Case. The REUSE stage may use the adaptation knowledge container to refine the retrieved solutions.
- REVISE: The Suggested Solution in the Solved Case is evaluated for correctness and is repaired if necessary to provide the Confirmed Solution in the Tested/Repaired Case. The REVISE stage may be achieved manually or may use adaptation knowledge, but it should be noted that a revision to a Suggested Solution is likely to be a less demanding task than solving the problem from scratch.
- RETAIN: The Repaired Case may be retained in the case base as a newly Learned Case if it is likely to be useful for future problem-solving. Thus the primary knowledge source for CBR may be added to during problem-solving and is an evolving, self-adaptive collection of problem-solving experiences.

Retrieval

CBR retrieval compares the problem part of the new case with each of the cases in the case base to establish the distance between the new case and the stored cases. The cases closest to the new case are retrieved for reuse. Retrieval is a major focus of López de Mántaras et al.'s (2005) review of research contributions related to the CBR cycle.

Similarity- and distance-based neighborhoods are commonly used interchangeably when discussing CBR

retrieval. Similarity and distance are inverses: the similarity is highest when the distance is close to 0, and the similarity is 0 when the distance is large. Several functions may be applied to define a suitable relationship between a distance d and a similarity s, including the following simple versions:

$$\text{Linear: } s = 1 - d \text{ (for normalized } d\text{)}$$
$$\text{Inverse: } s = \frac{1}{d} \text{ (for } d \neq 0\text{)}.$$

It is common to establish the distance between each pair of feature values and then to use a distance metric, often Euclidean or ►Manhattan distance (see also ►Similarity Measure), to calculate the distance between the feature vectors for the New and Retrieved Cases. The distance between two numeric feature values v and w for a feature F is normally taken to be the distance between the normalized values:

$$d(v, w) = \frac{|\, v - w \,|}{F_{max} - F_{min}}$$

where F_{max}/F_{min} are the maximum/minimum values of the feature F.

For nominal values v and w the simplest approach is to apply a binary distance function:

$$d(v, w) = \begin{cases} 0 & \text{if } v = w \\ 1 & \text{otherwise} \end{cases}$$

For ordered nominal values a more fine-grained distance may be appropriate. The distance between the ith value v_i and the jth value v_j in the ordered values $v_1, v_2, \ldots, v_n$ may use the separation in the ordering to define the distance:

$$d(v_i, v_j) = \frac{|\, i - j \,|}{n - 1}.$$

Extending this to arbitrary nominal values, a distance matrix D may define the distance between each pair of nominal values by assigning the distance $d(v_i, v_j)$ to d_{ij}.

Returning to the whisky-tasting example, suppose Sweetness and Peatiness score values 0–10, Color takes ordered values {pale, straw, gold, honey, amber}, Palate uses binary distance, and Nose is defined by the following distance matrix.

Nose Distance Matrix				
Distances	peat	fresh	soft	full
peat	0	0.3	1	0.5
fresh	0.3	0	0.5	0.7
soft	1	0.5	0	0.3
full	0.5	0.7	0.3	0

Dalmore whisky above can be compared with Laphroaig and The Macallan as follows:

Sweetness	Peatiness	Color	Nose	Palate	Distillery
2	10	amber	peat	medium-dry	Laphroaig
7	4	gold	full	big-body	The Macallan

The Manhattan distances are:

$$d(\text{Dalmore,Laphroaig}) = 0.4 + 0.5 + 0 + 0.5 + 0 = 1.4;$$

$$d(\text{Dalmore,The Macallan}) = 0.1 + 0.1 + 0.5 + 0 + 1 = 1.7.$$

Taking all the whisky features with equal importance, Dalmore is more similar to Laphroaig than to The Macallan.

In situations where the relative importance of features should be taken into account, a weighted version of the distance function should be used; for example, the weighted Manhattan distance between two normalized vectors $\mathbf{x} = (x_1, x_2, \ldots x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots y_n)$ with weight w_i for the ith feature is

$$d(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{n} w_i \,|\, x_i - y_i \,|}{\sum_{i=1}^{n} w_i}$$

In the example above if Peatiness has weight 4 and the other features have weight 1 then the weighted Manhattan distances are:

$$d(\text{Dalmore,Laphroaig}) = (0.4 + 4 \times 0.5 + 0 + 0.5 + 0)/8 = 0.36;$$
$$d(\text{Dalmore,The Macallan}) = (0.1 + 4 \times 0.1 + 0.5 + 0 + 1)/8 = 0.25.$$

Therefore, emphasizing the distinctive Peatiness feature, Dalmore is more similar to The Macallan than to Laphroaig.

The similarity knowledge container contains knowledge to calculate similarities. For simple feature vectors a weighted sum of distances is often sufficient, and the weights are similarity knowledge. However, even our whisky tasting domain had additional similarity knowledge containing the distance matrix for the Nose feature. Structured cases require methods to calculate the similarity of two cases from the similarities of components. CBR may use very knowledge-intensive methods to decide similarity for the retrieval stage. Ease of reuse or revision may even be incorporated as part of the assessment of similarity. Similarity knowledge may also define how ►missing values are handled: the feature may be ignored, the similarity may be maximally pessimistic, or a default or average value may be used to calculate the distance.

A CBR case base may be indexed to avoid similarity matching being applied to all the cases in the case base. One approach uses kd trees to partition the case base according to hyper-planes. ►Decision Tree algorithms may be used to build the kd tree by using the cases as training data, partitioning the cases according to the chosen decision nodes, and storing the cases in the appropriate leaf nodes. Retrieval first traverses the decision tree to select the cases in a leaf node, and similarity matching is applied to only this partition. Case Retrieval Nets are designed to speed up retrieval by applying spreading activation to select relevant cases. In Case Retrieval Nets the feature value nodes are linked via similarity to each other and to cases. Indexes can speed up retrieval but they also pre-select cases according to some criteria that may differ from similarity.

Reuse and Revision

Reuse may be as simple as copying the solution from the Retrieved Case. If k nearest neighbors are retrieved then a vote of the classes predicted in the retrieved cases may be used for ►classification, or the average of retrieved values for ►regression. A weighted vote or weighted average of the retrieved solutions can take account of the nearness of the retrieved cases in the calculation. For more complex solutions, such as designs or plans, the amalgamation of the solutions from the Retrieved Cases may be more knowledge intensive.

If the New Case and the Retrieved Case are different in a significant way then it may be that the solution from the Retrieved Case should be adapted before being proposed as a Suggested Solution. Adaptation is designed to recognize significant differences between the New and Retrieved Cases and to take account of these by adapting the solution in the Retrieved Case.

In classification domains, it is likely that all classes are represented in the case base. However, different problem features may alter the classification and so adaptation may correct for a lack of cases. In constructive problem-solving like design and planning, however, it is unlikely that all solutions (designs, plans, etc.) will be represented in the case base. Therefore, a retrieved case suggests an initial design or plan, and adaptation alters it to reflect novel feature values.

There are three main types of adaptation that may be used as part of the reuse step, to refine the solution in the Retrieved Case to match better the new problem, or as part of the revise stage to repair the Suggested Solution in the Solved Case.

- Substitution: Replace parts of the retrieved solution. In Hammond's (1990) CHEF system to plan Szechuan recipes, the substitution of ingredients enables the requirements of the new menu to be achieved. For example, the beef and broccoli in a retrieved recipe is substituted with chicken and snowpeas.
- Transformation: Add, change, or remove parts of the retrieved solution. CHEF adds a skinning step to the retrieved recipe that is needed for chicken but not for beef.
- Generative Adaptation: Replay the method used to derive the retrieved solution. Thus the retrieved solution is not adapted but a new solution is generated from reusing the retrieved method for the new circumstances. This approach is similar to reasoning by analogy.

CHEF also had a clear REVISE stage where the Suggested Solution recipe was tested in simulation and any faults were identified, explained, and repaired. In one recipe a strawberry soufflé was too liquid. CHEF has a set of Thematic Organization Packets (TOPs) that are templates for repairs for different types of explained failures. (TOPs continue the experience template theme of ►dynamic memory model MOPs.) One repair for the soufflé is to drain the strawberry pulp and this

transformation adaptation is one REVISE operation that could be applied.

The adaptation knowledge container is an important source of knowledge for some CBR systems, particularly for design and planning, where refining an initial design or plan is expected. Acquiring adaptation knowledge can be onerous. The CHEF examples above indicate that the knowledge must store refinements to the solutions initially proposed from the retrieved cases. Learning adaptation knowledge from the implicit refinement information captured in the case base has been effective for substitution adaptation in component-based design (Craw, Wiratunga, & Rowe, 2006).

Retention and Maintenance

Retention of new cases during problem-solving is an important advantage of CBR systems. However, it is not always advantageous to retain all new cases. The ►Utility Problem – *that the computational benefit from additional knowledge must not outweigh the cost of applying it* – in CBR refers to cases and the added cost of retrieval. The case base must be kept "lean and mean," and so new cases are not retained automatically, and cases that are no longer useful are removed. New cases should be retained if they add to the competence of the CBR system by providing problem-solving capability in an area of the problem space that is currently sparse. Conversely, existing cases should be reviewed for the role they play and forgetting cases is an important maintenance task. Existing cases may contain outdated experiences and so should be removed, or they may be superseded by new cases.

Case base maintenance manages the contents of the case base to achieve high competence. Competence depends on the domain and may involve

- quality of solution;
- user confidence in solution; or
- efficiency of solution prediction (e.g., speed-up learning).

Case base maintenance systems commonly assume that the case base contains a representative sample of the problem-solving experiences. They exploit this by using a leave-one-out approach where repeatedly for each case in the case base, the one extracted case is used as a new case to be solved, and the remaining cases become the case base. This enables the problem-solving competence of the cases in the case base to be estimated using the extracted cases as representative new cases to be solved. Smyth & McKenna's (2001) competence model uses this approach to identify competence groups of cases with similar problem-solving ability. This model is used to underpin maintenance algorithms to prioritize cases for deletion and to identify areas where new cases might be added. There are several trade-offs to be managed by case base maintenance algorithms: larger case bases contain more experiences but take longer for retrieval; smaller case bases are likely to lack some key problem-solving ability; cases whose solution is markedly different from their nearest neighbors may be noisy or may be an important outlier.

CBR Tools

myCBR (mycbr-project.net) and jCOLIBRI (www.sourceforge.net) are open source CBR tools. Both provide state-of-the-art CBR functionality, and jColibri also incorporates a range of facilities for textual CBR. Several commercial CBR tools are available including Empolis: Information Access Suite (www.attensity.com), Kaidara's Advisor (www.servigistics.com), and ISoft's ReCall (www.alice-soft.com).

Applications

Several successful deployed applications of CBR are described in Cheetham and Watson (2005), including Lockheed's CLAVIER for designing layouts for autoclave ovens, Compaq's SMART help-desk system, Boeing's CASSIOPÉE for trouble-shooting aircraft engines and General Electric's FormTool for plastic color matching (Cheetham, 2005). The development of the INRECA methodology for engineering CBR systems was based on a range of industrial applications (Bergmann et al., 2003).

The wide range of CBR applications is demonstrated by the following list of application types.

- *Classification* – Medical diagnosis systems use patient records as a source of reusable experiences. Examples include SHRINK for psychiatry, CASEY for cardiac disease, ICONS for antibiotic therapy for intensive care, and BOLERO for pneumonia. Other diagnostic systems include failure prediction of rails

for Dutch railways from ultrasonic NDT, and failure analysis of semiconductors at National Semiconductor. Classification systems include PROTOS for audiologic disorders.

- *Design* – Architectural design was a popular early domain: CYCLOPS, ARCHIE, FABEL, and CADsyn are all early case-based design systems. Other design applications include CADET and KRITIK for engineering design, JULIA for recipes, chemical formulation for tyre rubber and pharmaceutical products, Déjà Vu for plant control software.
- *Planning* – PRODIGY is a general purpose planner that uses analogical reasoning to adapt retrieved plans. Other planning applications include PARIS (Bergmann & Wilke, 1996) for manufacturing planning in mechanical engineering, HICAP for evacuation planning, planning for forest fire management, mission planning for US navy, and route planning for DaimlerChrysler cars.
- *Conversational CBR* – Conversational systems extract the problem specification from the user through an interactive case-based dialogue and suggest solutions that match the partial specification. Examples include help-desk support, CaseAdvisor and CBR Strategist for fault diagnosis, and Wasabi, Sermo and ShowMe product recommender systems.
- *Personalization* – Personalized compilations of news, stories, music tracks, TV listings reuse previous experiences of the individual or others who have similar tastes. Other forms of personalized systems using CBR include route and travel planning, SPAM filtering and email management, and ClixSmart Navigator for mobile devices.
- *Textual CBR* – Legal decision support systems were an important early application domain for textual CBR. Examples include HYPO (Ashley & Rissland, 1988), CATO, GREBE, and SMILE. Question answering was another fruitful text-based domain: FAQ-Finder and FA11Q. More recently, textual CBR is used for decision support systems based on textual reports; for example, SOPHIA.

Future Directions

The drivers for Ubiquitous Computing – wireless communication and small devices – also affect future developments in CBR. The local, independent knowledge of case bases make them ideal to collect experiences, and to deliver experience-based knowledge for reuse.

Textual CBR systems are becoming increasingly important for extracting and representing knowledge captured in textual documents. This is particularly influenced by the availability of electronic documents and the Web as an information source.

Cross References

►Explanation-Based Learning
►Instance-Based Learning
►Lazy Learning
►Nearest Neighbor
►Similarity Metrics

Recommended Reading

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications, 7*, 39–59. citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.1670.

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*(1), 37–66. doi:10.1023/A:1022689900470.

Ashley, K., & Rissland, E. L. (1988). A case-based approach to modeling legal expertise. *IEEE Expert, 3*(3), 70–77. doi:10.1109/64.21892.

Bergmann, R., Althoff, K.-D., Breen, S., Göker, M., Manago, M., & Traphöner, R. (Eds.). (2003). *Developing industrial case-based reasoning applications*. *LNCS* (Vol. 1612). Berlin/Heidelberg: Springer. doi:10.1007/b94998.

Bergmann, R., & Wilke, W. (1996). On the role of abstraction in case-based reasoning. In I. Smith & B. Faltings (Eds.), *Proceedings of the third European workshop on case-based reasoning*, Lausanne, Switzerland (pp. 28–43), *LNCS* (Vol. 1168). Berlin/Heidelberg: Springer.

Cheetham, W. (2005). Tenth anniversary of the plastics color formulation tool. *AI Magazine, 26*(3), 51–61. www.aaai.org/Papers/Magazine/Vol26/26-03/AIMag26-03-007.pdf.

Cheetham, W., & Watson, I. (2005). Fielded applications of case-based reasoning. *Knowledge Engineering Review, 20*(3), 321–323. doi:10.1017/S0269888906000580.

Craw, S., Wiratunga, N., & Rowe, R. C. (2006). Learning adaptation knowledge to improve case-based reasoning. *Artificial Intelligence, 170*(16–17), 1175–1192. doi: 10.1016/j.artint.2006.09.001.

Hammond, K. J. (1990). Explaining and repairing plans that fail. *Artificial Intelligence, 45*(1–2), 173–228.

Kolodner, J. L. (1993). *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann.

Leake, D. (1996). CBR in context: The present and future. In D. Leake (Ed.), *Case-based reasoning: Experiences, lessons, and future directions* (pp. 3–30). Menlo Park, CA: AAAI Press. citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.9114.

López de Mántaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M.T., Forbus, K., Aamodt, A., & Watson, I. (2005). Retrieval, reuse, revision,

and retention in case-based reasoning. *Knowledge Engineering Review, 20*(3), 215–240. doi: 10.1017/S0269888906000646.
Richter, M. M., & Aamodt, A. (2005). Case-based reasoning foundations. *Knowledge Engineering Review, 20*(3), 203–207. doi:10.1017/S0269888906000695.
Riesbeck, C. K., & Schank, R. C. (1989). *Inside case-based reasoning*. Hillsdale, NJ: Lawrence Erlbaum.
Smyth, B., & McKenna, E. (2001). Competence models and the maintenance problem. *Computational Intelligence, 17*(2), 235–249. doi:10.1111/0824-7935.00142.

Categorical Attribute

Synonyms

Qualitative attribute

Categorical attributes are attributes whose values can be placed into distinct categories. See ►Attribute and ►Measurement Scales.

Categorical Data Clustering

Periklis Andritsos[1], Panayiotis Tsaparas[2]
[1]Toronto, ON, Canada
[2]Mountain View, CA, USA

Synonyms

Clustering of nonnumerical data; Grouping

Definition

Data clustering is informally defined as the problem of partitioning a set of objects into groups, such that the objects in the same group are similar, while the objects in different groups are dissimilar. Categorical data clustering refers to the case where the data objects are defined over ►categorical attributes. A categorical attribute is an attribute whose domain is a set of discrete values that are not inherently comparable. That is, there is no single ordering or inherent distance function for the categorical values, and there is no mapping from categorical to numerical values that is semantically meaningful.

Motivation and Background

Clustering is a problem of great practical importance that has been the focus of substantial research in several domains for decades. As storage capacities grow, we have at hand larger amounts of data available for analysis and mining. Clustering plays an instrumental role in this process. This trend has created a surge of research activity in devising new clustering algorithms that can handle large amounts of data and produce results of high quality.

In data clustering, we want to partition objects into groups such that similar objects are grouped together while dissimilar objects are grouped separately. This objective assumes that there is some well-defined notion of similarity, or distance, between data objects, and a way to decide if a group of objects is a homogeneous cluster. Most of the clustering algorithms in the literature focus on data sets where objects are defined over numerical attributes. In such cases, the similarity (or dissimilarity) of objects and the quality of a cluster can be defined using well-studied measures that are derived from geometric analogies. These definitions rely on the semantics of the data values themselves. For example, the values \$100,000 and \$110,000 are more similar than \$100,000 and \$50,000, and intuitively more similar than \$10,000 and \$1. The existence of a distance measure allows us to define a quality measure for a clustering such as the mean square distance between each point and the representative of its cluster. Clustering then becomes the problem of grouping together points such that the quality measure is optimized.

However, there are many data sets where the data objects are defined over attributes, which are neither numerical nor inherently comparable in any way. We term such data sets *categorical*, since they represent values of certain categories. As a concrete example, consider the toy data set in Table 1 that stores information about movies. For the purpose of exposition, a movie is characterized by the attributes "director," "actor/actress," and "genre." In this setting, it is not immediately obvious what the distance, or similarity, is between the values "Coppola" and "Scorsese," or the movies "Vertigo" and "Harvey."

There are plenty of examples of categorical data: product data, where products are defined over attributes such as brand, model, or color; census data, where information about individuals includes attributes such as marital status, education, and occupation; ecological data where plants and animals can be described with attributes such as shape of petals or type of habitat.

Categorical Data Clustering. Table 1 An instance of a movie database

	Director	Actor	Genre
$\mathbf{t}_1$ (Godfather II)	Coppola	De Niro	Crime
$\mathbf{t}_2$ (Good fellas)	Scorsese	De Niro	Crime
$\mathbf{t}_3$ (Vertigo)	Hitchcock	Stewart	Thriller
$\mathbf{t}_4$ (N by NW)	Hitchcock	Grant	Thriller
$\mathbf{t}_5$ (Bishop's wife)	Koster	Grant	Comedy
$\mathbf{t}_6$ (Harvey)	Koster	Stewart	Comedy

There is a plethora of such data sets, and there is always a need for clustering and analyzing them.

The lack of an inherent distance or similarity measure between categorical data objects, makes categorical data clustering a challenging problem. The challenge lies in defining a quality measure for categorical data clustering that captures the human intuition of what it means for categorical data objects to be similar. In the next sections, we present an overview of the different efforts at addressing this problem and the resulting clustering algorithms.

Structure of the Learning System

Generic Data Clustering System

We first describe the outline for a generic data clustering system, not necessarily of categorical data. In the next section we focus on categorical data specific challenges.

Data clustering is not a one-step process. In one of the seminal texts on Cluster Analysis, Jain and Dubes divide the clustering process into the following stages (Jain & Dubes, 1988):

Data collection: Includes the careful extraction of relevant data objects from the underlying data sources. In our context, data objects are distinguished by their individual values for a set of attributes.

Initial screening: Refers to the massaging of data after its extraction from the source or sources. This stage is closely related to the process of data cleaning in databases (Jarke, Lenzerini, Vassiliou, & Vassiliadis, 1999).

Representation: Includes the proper preparation of the data in order to become suitable for the clustering algorithm. In this step, the similarity measure is chosen, and the characteristics and dimensionality of the data are examined.

Clustering tendency: Checks whether the data at hand has a natural tendency to cluster or not. This stage is often ignored, especially for large data sets.

Clustering strategy: Involves the careful choice of clustering algorithm and initial parameters, if any.

Validation: Validation is often based on manual examination and visual techniques. However, as the amount of data and its dimensionality grow, we may have no means to compare the results with preconceived ideas or other clusterings.

Interpretation: This stage includes the combination of clustering results with other analyses of the data (e.g., classification), in order to draw conclusions and suggest further analysis.

In this chapter, we are interested in problems relating to Representation and Clustering Strategy. These lie in the heart of the data clustering problem, and there has been considerable research effort in these areas within the Data Mining and Machine Learning communities. More specifically, we consider the following two subproblems.

Formal formulation of the clustering problem: In order to devise algorithms for clustering, we need to mathematically formulate the intuition captured in the informal definition of the clustering problem that similar objects should be grouped together and dissimilar objects should be grouped separately. The problem formulation typically requires at least one of the following:

- A measure of similarity or distance between two data objects.
- A measure of similarity or distance between a data object and a cluster of objects. This is often defined by defining a representative for a cluster as a (new) data object and comparing the data object with the representative.
- A measure of the quality of a cluster of data objects.

The result of the formulation step is to define a clustering optimization criterion that guides the grouping of the objects into clusters.

When the data is defined over numerical attributes, these measures are defined using geometric analogies. For example, if each object is a point in the Euclidean space, then the distance between two points can be

defined as the Euclidean distance, and the representative of a cluster as the mean Euclidean vector. The quality of a cluster can be defined with respect to the "variance" of the cluster, that is, the sum of squares of the distances between each object and the mean of the cluster. The optimization criterion then becomes to minimize the variance over all clusters of the clustering.

The clustering algorithm: Once we have a mathematical formulation of the clustering problem, we need an algorithm that will find the optimal clustering in an efficient manner. In most cases, finding the optimal solution is an NP-hard problem efficient heuristics or approximation algorithms are considered. There is a large literature on this subject that approaches the problem from different angles.

There exist a large number of different clustering techniques and algorithms. We now selectively describe some broad classes of clustering algorithms and problems. A thorough categorization of clustering techniques can be found in Han and Kamber (2001), where different clustering problems, paradigms, and techniques are discussed.

Hierarchical clustering algorithms: This is a popular clustering technique, since it is easy to implement, and it lends itself well to visualization and interpretation. Hierarchical algorithms create a hierarchical decomposition of the objects. They are either *agglomerative* (*bottom-up*) or *divisive* (*top-down*). *Agglomerative* algorithms start with each object being a separate cluster itself, and successively merge groups according to a distance measure. *Divisive* algorithms follow the opposite strategy. They start with one group of all objects and successively split groups into smaller ones, until each object falls into one cluster, or as desired. The hierarchical *dendrogram* produced is often in itself the output of the algorithm, since it can be used for visualizing the data. Most of the times, both approaches suffer from the fact that once a merge or a split is committed, it cannot be undone or refined.

Partitional clustering algorithms: ▶Partitional clustering algorithms define a clustering optimization criterion and then seek the partition that optimizes this criterion. Exhaustive search over all partitions is infeasible, since even for few data objects the number of possible partitions is huge. Partitional clustering algorithms often start with an initial, usually random, partition and proceed with its refinement by locally improving the optimization criterion. The majority of such algorithms could be considered as greedy-like algorithms. They suffer from the fact that they can easily get stuck to local optima.

Spectral clustering algorithms: Spectral algorithms view the data set to be clustered as a two dimensional matrix of data objects and attributes. The entries in the matrix may be the raw values or some normalized form of these values. The principal eigenvectors of the matrix have been shown to capture the main clusters in the data. There is a rich literature on different types of spectral algorithms.

Graph clustering: ▶Graph clustering defines a range of clustering problems, where the distinctive characteristic is that the input data is represented as a graph. The nodes of the graph are the data objects, and the (possibly weighted) edges capture the similarity or distance between the data objects. The data may come naturally in the form of a graph (e.g., a social network), or the graph may be derived in some way from the data (e.g., link two products if they appear together in a transaction). Some of the techniques we described above are directly applicable to graph data. We can also use techniques from graph theory for finding a good clustering.

Categorical Data Clustering System

In the clustering paradigm we outlined, a step of fundamental importance is to formally formulate the clustering problem, by defining a clustering optimization criterion. As we detailed above, for this step we need a measure of distance or similarity between the data objects, or a measure of cluster quality for a group of data objects. For categorical data there exists no inherent ordering or distance measure, and no natural geometric analogies we can explore, causing the clustering paradigm to break down. Research efforts on categorical data clustering have focused on addressing this problem by imposing distance measures on the categorical data and defining clustering quality criteria. We now outline some of these approaches.

Overlap-Based Similarity Measures A simple and intuitive method for comparing two categorical data objects is based on counting the overlap between the categorical values of the objects. The higher the overlap, the more similar the two objects are. This intuition leads to the

use of well-known measures such as the (*generalized*) *Hamming distance* (Jain & Dubes, 1988), which measures the number of attributes that take different values between two tuples, or the *Jaccard* similarity measure, which is defined as the intersection over the union of the values in the two tuples. In the example of Table 1, tuples $\mathbf{t}_1$ (Godfather II) and $\mathbf{t}_2$ (Good fellas) have Hamming distance 1 and Jaccard coefficient 1/2.

Two algorithms that make use of overlap-based measures are *k-modes* (Huang, 1998), and *RObust Clustering using linKs* (*ROCK*) (Guha, Rastogi, & Shim, 1999). The *k*-modes algorithm is a partitional algorithm inspired by the *k-means* algorithm, a well-known clustering algorithm for numerical data. The representative of a categorical data cluster is defined to be a data object where each attribute takes the *mode* emphasize the mode value of an attribute is the most frequent value for that attribute in the cluster.

The ROCK algorithm makes use of the Jaccard coefficient to define *links* between data objects. The data is then represented in the form of a graph, and the problem becomes essentially a graph clustering problem. Given two clusters of categorical data, ROCK measures the similarity of two clusters by comparing their *aggregate interconnectivity* against a user-specified model, thus avoiding the problem of defining a cluster representative.

Context-Based Similarity Measures One way to define relationships between categorical values is by comparing the *context* in which they appear. For two categorical attribute values we define the context as the values of the remaining attributes with which they co-occur in the data set. The more similar these two contexts are, the more similar the attribute values are. For example, in Table 1, Scorsese and Coppola are close since they appear in exactly the same context ("De Niro", "Crime"), while Scorsese and Hitchcock are far since their contexts are completely disjoint. Defining a distance between value contexts can be done using overlap similarity measures (Das & Mannila, 2000) or by using information-theoretic measures, i.e., comparing the distributions defined by the two contexts (Andritsos, Tsaparas, Miller, Kenneth, & Sevcik, 2004). Once we have the relationships between the values we can use standard clustering techniques for solving the clustering problem.

There are various algorithms that make use of the idea that similar values should appear in similar contexts in order to cluster categorical data. The *Clustering cAteCorigal daTa Using Summaries* (*CACTUS*) algorithm (Ganti, Gehrke, & Ramakrishnan, 1999) creates groups of attribute values based on the similarity of their context. It then uses a hierarchical greedy algorithm for grouping tuples and attributes.

In a slightly different fashion, *the STIRR algorithm (Sieving Through Iterated Relational Reinforcement)* [GKR98] uses the idea that similar tuples should contain co-occurring values and similar values should appear in tuples with high overlap. This idea is implemented via a dynamical system, inspired by Information Retrieval techniques (Kleinberg Jon, 1999). When the dynamical system is linear, the algorithm is similar to spectral clustering algorithms.

CLICKS (Zaki, Peters, Assent, & Seidl, 2005) is an algorithm that is similar to STIRR. Rather than a measure of similarity/distance, it uses a graph-theoretic approach to find *k* disjoint sets of vertices in a graph constructed for a particular data set. One special characteristic of this algorithm is that it discovers clusters in a subset of the underlying set of attributes.

Information-Theoretic Clustering Criteria The information content in a data set can be quantified through the well-studied notions of *entropy* and *mutual information* (Cover & Thomas, 1991). Entropy measures the uncertainty in predicting the values of the data when drawn from the data distribution. If we view each tuple, or cluster of tuples, as a distribution over the categorical values, then we can define the *conditional entropy* of the attribute values given a set of tuples, as the uncertainty of predicting the values in this set of tuples. If we have a single tuple, then the entropy is zero, since we can accurately predict the values. For tuple $\mathbf{t}_1$ we know the director, the actor, and the genre with full certainty. As we group tuples together the uncertainty (and entropy) increases. Grouping together tuples $\mathbf{t}_1$ and $\mathbf{t}_2$ creates uncertainty about the director attribute, while grouping $\mathbf{t}_1$ and $\mathbf{t}_3$ creates uncertainty about all attributes. Hence the latter grouping has higher entropy than the former. Information-theoretic criteria for clustering aim at generating clusters with low entropy, since this would imply that the clusters are homogeneous, and there is little

information loss as a result of the clustering. This formulation allows for defining the distance between sets of tuples, using entropy-based distance measures such as the *Jensen–Shannon* divergence (Cover & Thomas, 1991). The Jensen–Shannon divergence captures the informational distances in categorical data, in a similar way that Euclidean distance captures geometric distances inherent in numerical data.

Two algorithms that make use of this idea are *COOLCAT* (Barbarà, Couto, & Li, 2002) and *LIMBO* (*scaLable InforMation Bottleneck*) [ATMR04]. COOLCAT is a partitional algorithm that performs a local search for finding the partition with k clusters with the lowest entropy. LIMBO works by constructing a summary of the data set that preserves as much information about the data as possible and then produces a hierarchical clustering of the summary. It is a scalable algorithm that can be used in both static and streaming environments.

A related approach is adopted by the COBWEB algorithm (Fisher, 1987; Gluck & Corter, 1985), a divisive hierarchical algorithm that optimizes the *category utility* measure, which measures how well particular values can be predicted given the clustering as opposed to having them in the original data set unclustered.

Categorical Clustering as Clustering Aggregation A different approach to the categorical data clustering problem is to view it as a *clustering aggregation problem*. Given a collection of clusterings of the data objects, the clustering aggregation problem looks for the single clustering that agrees as much as possible with the input clusterings. The problem of clustering aggregation has been shown to be equivalent to categorical data clustering (Gionis, Mannila, & Tsaparas, 2007), where each categorical attribute defines a clustering of the data objects, grouping all the objects with the same value together. For example, in Table 1, the attribute "genre" defines three clusters: the Crime cluster, the Thriller cluster, and the Comedy cluster. Similarly, the attribute "actor" defines three clusters, and the attribute "director" defines four clusters.

Various definitions have been considered in the literature for the notion of agreement between the output clustering and the input clusterings. One definition looks at all pairs of objects, and defines a *disagreement* between two clusterings if one clustering places the two objects in the same cluster, while the other places them in different clusters; an *agreement* is defined otherwise. The clustering criterion is then to minimize the number of disagreements (or maximize the number of agreements). Other definitions are also possible, which make use of information-theoretic measures, or mappings between the clusters of the two clusterings. There is a variety of algorithms for finding the best aggregate cluster, many of which have also been studied theoretically.

Cross References

►Clustering
►Data Mining
►Graph clustering
►Instance-Based Learning
►Partitional clustering

Recommended Reading

Andritsos, P., Tsaparas, P., Miller, R. J., Kenneth, C., & Sevcik, K. C. (2004). LIMBO: Scalable clustering of categorical data. In *Proceedings of the 9th international conference on extending database technology (EDBT)* (pp. 123–146). Heraklion, Greece.

Barbarà, D., Couto, J., & Li, Y. (2002). COOLCAT: An entropy-based algorithm for categorical clustering. In *Proceedings of the 11th international conference on information and knowledge management (CIKM)* (pp. 582–589). McLean, VA.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory*. New York: Wiley.

Das, G., & Mannila, H. (2000). Context-based similarity measures for categorical databases. In *Proceedings of the 4th European conference on principles of data mining and knowledge discovery (PKDD)* (pp. 201–210). Lyon, France.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*, 139–172.

Ganti, V., Gehrke, J., & Ramakrishnan, R. (1999). CACTUS: Clustering categorical data using summaries. In *Proceedings of the 5th international conference on knowledge discovery and data mining (KDD)* (pp. 73–83). San Diego, CA.

Gionis, A., Mannila, H., & Tsaparas, P. (2007). Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, *1*(1), Article No 4.

Gluck, M., & Corter, J. (1985). Information, uncertainty, and the utility of categories. In *Proceedings of the 7th annual conference of the cognitive science society (COGSCI)* (pp. 283–287). Irvine, CA.

Guha, S., Rastogi, R., & Shim, K. (1999). ROCK: A robust clustering algorithm for categorical attributes. In *Proceedings of the 15th international conference on data engineering* (pp. 512–521). Sydney, Australia.

Han, J., & Kamber, M. (2001). *Data mining: Concepts and techniques*. San Francisco: Morgan Kaufmann.

Huang, Z. (1998). Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, *2*(3), 283–304.
Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Englewood Cliffs, NJ: Prentice-Hall.
Jarke, M., Lenzerini, M., Vassiliou, Y., & Vassiliadis, P. (1999). *Fundamentals of data warehouses*. Berlin: Springer.
Kleinberg, Jon (1999). Authoritative sources in a hyperlinked environment". *Journal of the ACM 46*(5): 604–632.
Zaki, M. J., Peters, M., Assent, I., & Seidl, T. (2005). CLICKS: An effective algorithm for mining subspace clusters in categorical datasets. In *Proceeding of the 11th international conference on knowledge discovery and data mining (KDD)* (pp. 736–742). Chicago, IL.

Categorization

►Classification
►Concept Learning

Category

►Class

Causal Discovery

►Learning Graphical Models

Causality

Ricardo Silva
University College London, London, UK

Definition

The main task in causal inference is predicting the outcome of an intervention. For example, a treatment assigned by a doctor that will change the patient's heart condition is an intervention. Predicting the change in the patient's condition is a causal inference task. In general, an intervention is an action taken by an external agent that changes the original values, or the probability distributions, of some of the variables in the system. Besides predicting outcomes of actions, causal inference is also concerned with explanation: identifying which were the causes of a particular event that happened in the past.

Motivation and Background

Many problems in machine learning are prediction problems. Given a feature vector $\mathbf{X}$, the task is to provide an estimate of some output vector $\mathbf{Y}$, or its conditional probability distribution $P(\mathbf{Y}|\mathbf{X})$. This typically assumes that the distribution of $\mathbf{Y}$ given $\mathbf{X}$ during learning is the same distribution at prediction time. There are many scenarios where this is not the case.

Epidemiology and several medical sciences provide counterexamples. Consider two seemingly straightforward learning problems. In the first example, one is given epidemiological data where smokers are clearly more propense than nonsmokers to develop lung cancer. Can I use this data to learn that smoking causes cancer? In the second example, consider a group of patients suffering from a type of artery disease. In this group, those that receive a bypass surgery are likely to survive longer than those that receive a particular set of drugs with no surgery.

There is no fundamental problem on using such datasets to predict the probability of a smoker developing lung cancer, or the life expectancy of someone who went through surgery. Yet, the data does not necessarily tell you if smoking is a cause of lung cancer, or that nationwide the government should promote surgery as the treatment of choice for that particular heart disease. What is going on?

There are reasons to be initially suspicious of such claims. This is well-known in statistics as the expression "association is not causation" (Wasserman, 2004, p. 253). The data-generating mechanism for our outcome $\mathbf{Y}$ ("developing lung cancer," "getting cured from artery disease") given the relevant inputs $\mathbf{X}$ ("smoking habit," "having a surgery") might change under an *intervention* for reasons such as follows.

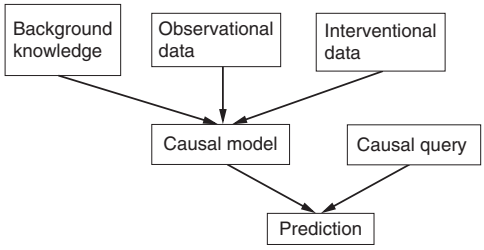
In the smoking example, the reality might be that there are several *hidden common causes* that are responsible for the observed association. A genetic factor, for instance: the possibility that there is a class of genotypes on which people are more likely to pick up smoking *and* develop lung cancer, without any direct causal connection between the two variables. In the artery disease

example, surgery might not be the best choice to be made by a doctor. It might have been the case that so far patients in better shape were more daring in choosing, by themselves, the surgery treatment. This *selection bias* will favor surgery over drug treatment, since from the outset the patients that are most likely to improve take that treatment.

When treatment is enforced by an *external agent* (the doctor), such selection bias disappears, and the resulting $P(\mathbf{Y}|\mathbf{X})$ will not be the same. One way of learning this relationship is through *randomized trials* (Rosenbaum, 2002). The simplest case consists on flipping a coin for each patient on the training set. Each face of the coin corresponds to a possible treatment, and assignment is done accordingly. Since assignment does not depend on any hidden common cause or selection bias, this provides a basis for learning causal effects. Machine learning and statistical techniques can be applied directly in this case (e.g., ►logistic regression). Data analysis performed with a randomized trial is sometimes called an *interventional study*.

The smoking case is more complicated: a direct intervention is not possible, since it is not acceptable to force someone to smoke or not to smoke. The inquiry asks only for a *hypothetical intervention*, that is, if someone is forced to smoke, will his or her chances of developing lung cancer increase? Such an intervention will not take place, but this still has obvious implications in public policy. This is the heart of the matter in issues such as deciding on raising tobacco taxes, or forbidding smoking in public places. However, data that measures this interventional data-generation mechanism will never be available for ethical reasons. The question has to be addressed through an *observational study*, that is, a study for causal predictions without interventional data.

Observational studies arise not only under the impossibility of performing interventions, but also in the case where performing interventions is too expensive or time consuming. In this case, observational studies, or a combination of observational and interventional studies, can provide extra information to guide an experimental analysis (Cooper & Yoo, 1999; Eaton & Murphy, 2007; Eberhardt, Glymour, & Scheines, 2005; Sachs, Prez, Pe'er, Lauffenburger, & Nolan, 2005). The use of observational data, or the combination of several interventional datasets, is where the greatest contributions of machine learning to causal inference rest.



Structure of the Learning System

Structure of Causal Inference

In order to use observational data, a causal inference system needs a way of linking the state of the world under an intervention to the *natural* state of the world. The natural state is defined as the one to which no external intervention is applied. In the most general formulation, this link between the natural state and the manipulated world is defined for interventions in any subset of variables in the system.

A common language for expressing the relationship between the different states of the world is a *causal graph*, as explained in more detail in the next section. A causal model is composed of the graph and a probability distribution that factorizes according to the graph, as in a standard ►graphical model. The only difference between a standard graphical model and a causal graphical model is that in the latter extra assumptions are made. The graphical model can be seen as a way of encoding such assumptions.

The combination of assumptions, observational, and interventional data generates such a causal graphical model. In the related problem of reinforcement learning, the agent has to maximize a specific utility function and typically has full control on which interventions (actions) can be performed. Here we will focus on the unsupervised problem of learning a causal model for a fixed input of observational and interventional data.

Because only some (or no) interventional data might be available, the learning system might not be able to answer some causal queries. That is, the system will not provide an answer for some prediction tasks.

Languages and Assumptions for Causal Inference Directed acyclic graphs (DAGs) are a popular language in machine learning to encode qualitative statements about causal relationships. A DAG is composed of a set of vertices and a set of directed edges. The notions

of parents, children, ancestors, and descendants are the usual ones found in graphical modeling literature.

In terms of causal statements, a directed edge $A \rightarrow B$ states that A is a *direct* cause of B: that is, different interventions on A will result in different distributions for B, even if we intervene on all other variables. The assumption that A is a cause of B is not used in noncausal ▶graphical models.

A causal DAG G satisfies the *causal Markov condition* if and only if a vertex is independent of all of its nondescendants given its direct causes (parents). In Fig. 1(a), A is independent of D, E, and F given its parents, B and C. It may or may not be independent of G given B and C.

The causal Markov condition implies several other conditional independence statements. For instance, in Fig. 1(a) we have that H is independent of F given A. Yet, this is not a statement about the parents of any vertex. Pearl's d-separation criterion (Pearl, 2000) is a sound and complete way of reading off independencies, out of a DAG, which are entailed by the causal Markov condition. We assume that the joint probability distribution over the vertice variables is *Markov* with respect to the graph, that is, any independence statement that is encoded by the graph should imply the corresponding independence in the distribution.

Representing Interventions

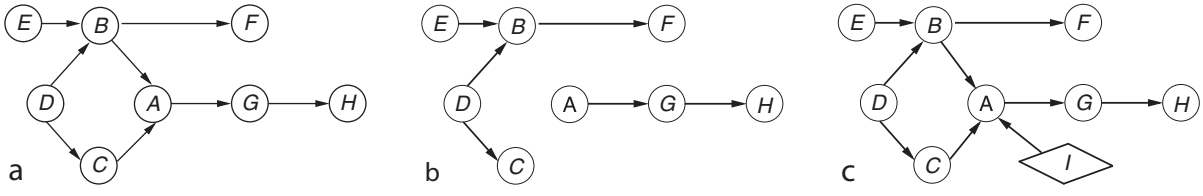
The local modularity given by the causal Markov condition leads to a natural notion of intervention. Random variable V, represented by a particular vertex in the graph, is following a *local mechanism*: its direct causes determine the distribution of V before its direct effects are generated. The role of an intervention is to *override* the natural local mechanism. An external agent substitutes the natural $P(V|Parents(V))$ by a new distribution $P_{Man}(V|Parents(V))$ while keeping the rest of the model unchanged ("Man" here stands for a particular manipulation). The notion of intervening by changing a single local mechanism is sometimes known as an *ideal intervention*. Other general types of interventions can be defined (Eaton & Murphy, 2007), but the most common frameworks for calculating causal effects rely on this notion.

A common type of intervention is the point mass intervention, which happens when V is set to some constant v. This can be represented graphically by "wiping out" all edges into V. Figure 1(b) represents the resulting graph in (a) under a point manipulation of A. Notice that A is now d-separated from its direct causes under this regime. It is also probabilistically independent, since A is now constant. This allows for a graphical machinery that can read off independencies out of a *manipulated* graph (i.e., the one with removed edges). It is the idea of representing the natural state of the world with a single causal graph, and allowing for modifications in this graph according to the intervention of choice, that links the different regimes obtained under different interventions.

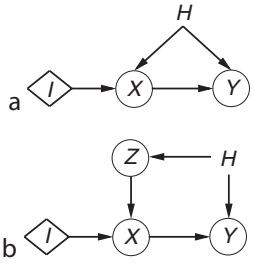
For the general case where a particular variable V is set to a new distribution, a *manipulation node* is added as an extra parent of V: this represents that an external agent is acting over that particular variable (Dawid, 2002; Pearl, 2000; Spirtes, Glymour, & Scheines, 2000), as illustrated in Fig. 1(c). $P(V|Parents(V))$ under intervention I is some new distribution $P_{Man}(V|Parents(V), I)$.

Calculating Distributions under Interventions

The notion of independence is a key aspect of probabilistic graphical models, where it allows for efficient computation of marginal probabilities. In causal graphical models, it also fulfills another important role: independencies indicate that the effect of some interventions can be estimated using observational data.



Causality. Figure 1. (a) A causal DAG. (b) Structure of the causal graph under some intervention that sets the value of *A* to a constant. (c) Structure of the causal graph under some intervention that changes the distribution of *A*



Causality. Figure 2. (a) *X* and *Y* have a hidden common cause *H*. (b) *Y* is dependent on the intervention node *I* given *X*, but conditioning on *Z* and marginalizing it out will allow us to eliminate the "back-door" path that links *X* and *Y* through the hidden common cause *H*

We will illustrate this concept with a simple example. One of the key difficulties in calculating a causal effect is *unmeasured confounding*, that is, hidden common causes. Consider Fig. 2(a), where X is a direct cause of Y, H is a hidden common cause of both and I is an intervention vertex. Without extra assumptions, there is no way of estimating the effect of X on Y using a training set that is sampled from the observed marginal $P(X, Y)$. This is more easily seen in the case where the model is multivariate Gaussian with zero mean. In this case, each variable is a linear combination of its parents with standard Gaussian additive noise

$$\begin{aligned} X &= aH + \epsilon_X \\ Y &= bX + cH + \epsilon_Y \end{aligned}$$

where H is also a standard normal random variable. The manipulated distribution $P_{Man}(Y|X, I)$, where I is a point intervention setting $X = x$, is a Gaussian distribution with mean $b \cdot x$. Value x is given by construction, but one needs to learn the unknown value b.

One can verify that the covariance of X and Y in the natural state is given by $a + bc$. Observational data, that is, data sampled from $P(X, Y)$, can be used to estimate the covariance of X and Y in the natural state, but from that it is not possible to infer the value of b: there are too many degrees of freedom.

However, there *are* several cases where the probability of **Y** given some intervention on **X** can be estimated with observational data and a given causal graph. Consider the graph in Fig. 2(b). The problem again is to learn the distribution of Y given X under regime I, that is, $P(Y|X, I)$. It can be read from the graph that I and Y are not independent given X, which means $P(Y|X, I) \neq P(Y|X)$. How can someone then estimate $P(Y|X, I)$ if no data for this process has been collected? The answer lies on *reducing the "causal query" to a "probabilistic query"* where the dependence on I disappears (and, hence, the necessity of having data measured under the I intervention). This is done by relying on the assumptions encoded by the graph:

$$\begin{aligned} P(Y|X, I) &= \textstyle\sum_z P(Y|X, I, z)P(Z = z|X, I) \\ &\quad (Z \text{ is discrete in this example}) \\ &= \textstyle\sum_z P(Y|X, z)P(Z = z|X, I) \\ &\quad (Y \text{ and } I \text{ are independent given } Z) \\ &\propto \textstyle\sum_z P(Y|X, z)P(X|z, I)P(Z = z|I) \\ &\quad (\text{By Bayes' rule}) \\ &= \textstyle\sum_z P(Y|X, z)P(X|z, I)P(Z = z) \\ &\quad (Z \text{ and } I \text{ are marginally independent}) \end{aligned}$$

In the last line, we have $P(Y|X, Z)$ and $P(Z)$, which can be estimated with observational data, since no intervention variable I appears on the expression. $P(X|Z, I)$ is set by the external agent: its value is known by construction. This means that the causal distribution $P(Y|X, I)$ can be learned even in this case where X and Y share a hidden common cause H.

There are several notations for denoting an interventional distribution such as $P(Y|X, I)$. One of the earliest was that of Spirtes et al. (2000), who used the notation

$$P(Y|set\ X = x) \tag{1}$$

to represent the distribution under an intervention I that fixed the value of X to some constant x. Pearl (2000) uses the operator *do* with a similar meaning.

$$P(Y|do(X = x)) \tag{2}$$

Pearl's *do*-calculus is essentially a set of operations for reducing a probability distribution that is a function of some intervention to a probability distribution that does not refer to any intervention. All reductions are conditioned on the independencies encoded in a given causal graph. This is in the same spirit of the example presented above.

The advantage of such notations is that, for point interventions, they lead to simple yet effective transformations (or to a report that no transformation is possible). Spirtes et al. (2000) and Pearl (2000) provide a detailed account of such prediction tools. By making a clear distinction between $P(Y|X)$ (X under the natural state) and $P(Y|do(X))$ (X under some intervention), much of the confusion that conflates causal and noncausal predictions disappears.

It is important to stress that methods such as the *do*-calculus are nonparametric, in the sense that they rely on conditional independence constraints only. More powerful reductions are possible if one is willing to provide extra information, such as assuming linearity of causal effects. For such cases, other parametric constraints can be exploited (Pearl, 2000; Spirtes et al., 2000).

Learning Causal Structure

In all of the previous section, we assumed that a causal graph was available. Since background knowledge is often limited, learning such graph structures becomes an important task. However, this cannot be accomplished without extra assumptions. To see why, consider again the example in Fig. 2(a): if $a + bc = 0$, it follows that the X and Y are independent in the natural state. However, Y is *not* causally independent of X (if $b \neq 0$): $P(Y|do(X = x_1))$ and $P(Y|do(X = x_2))$ will be two different Gaussians with means $b \cdot x_1$ and $b \cdot x_2$, respectively.

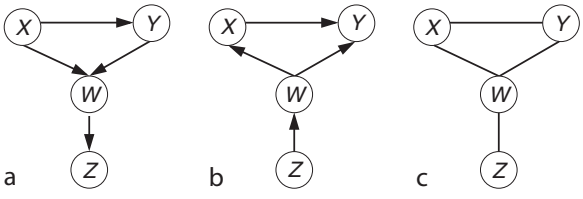
This example demonstrates that an independence constraint that is testable by observational data does not warrant causal independence, at least based on the causal Markov condition only. However, an independence constraint that arises from particular identities such as $a + bc = 0$ is not *stable*, in the sense that it does not follow from the qualitative causal relations entailed by the Markov condition: a change in any of the parameter values will destroy such a constraint.

The artificiality of unstable independencies motivates an extra assumption: the *faithfulness* condition (Spirtes et al., 2000), also known as the *stability* condition (Pearl, 2000). We say that a distribution P is faithful to a causal graph G if P is Markov with respect to G, *and* if each conditional independence in P corresponds to some d-separation in G. That is, on top of the causal Markov condition we assume that all independencies in P are entailed by the causal graph G.

The faithfulness condition allows us to reconstruct classes of causal graphs from observational data. In the simplest case, observing that X and Y are independent entails that there is no causal connection between X and Y. Consequently, $P(Y|do(X)) = P(Y|X) = P(Y)$. No interventional data was necessary to arrive at this conclusion, given the faithfulness condition.

In general, the solution is undetermined: more than one causal graph will be compatible with a set of observable independence constraints. Consider a simple example, where data is generated by a causal model with a causal graph given as in Fig. 3(a). This graph entails some independencies: for instance, that X and Z are independent given W, or that X and Y are not independent given any subset of $\{W, Z\}$. However, several other graphs entail the same conditional independencies. The graph in Fig. 3(b) is one example. The learning task is then discovering an *equivalence class* of graphs, not necessarily a particular graph. This is in contrast with the problem of learning the structure of noncausal graphical models: the fact that there are other structures compatible with the data is not important in this case, since we will not use such graphical models to predict the effect of some hypothetical intervention. An equivalence class might not be enough information to reduce a desired causal query to a probabilistic query, but it might require much less prior knowledge than specifying a full causal graph.

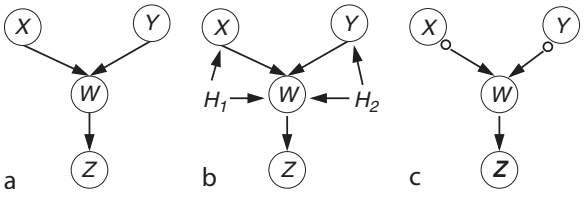
Assume for now that no hidden common causes exist in this domain. In particular, the graphical object in Fig. 3(c) is a representation of the equivalence class of graphs that are compatible with the independencies encoded in Fig. 3(a) (Pearl, 2000; Spirtes et al., 2000). All members of the equivalence class will have the same



Causality. Figure 3. (a) A particular causal graph which entails a few independence constraints, such as *X* and *Z* being independent given *W*. (b) A different causal graph that entails exactly the same independence constraints as (a). (c) A representation for all graphs that entail the same conditional independencies as (a) and (b)

skeleton of this representation, that is, the same adjacencies. An undirected edge indicates that there are two members in the equivalence class where directionality of this particular edge goes in opposite directions. Some different directions are illustrated in Fig. 3(b). One can verify from the properties of d-separation that if an expert or an experiment indicates that $X - W$ should be directed as $X \rightarrow W$, then the edge $W - Z$ is *compelled* to be directed as $W \rightarrow Z$: the direction $W \leftarrow Z$ is incompatible with the simultaneous findings that X and Z are independent given W, and that X causes W.

More can be discovered if more independence constraints exist. In Fig. 4(a), X is not a cause of Y. If we assume no hidden common causes exist in this domain, then no other causal graph is compatible with the independence constraints of Fig. 4(a): the equivalence class is this graph only. However, the assumption of no hidden common causes is strong and undesirable. For instance, the graph in Fig. 4(b), where H_1 and H_2 are hidden, is in the same equivalence class of (a). Yet, the graph in Fig. 4(a) indicates that $P(W|do(X)) = P(W|X)$, which can be arbitrarily different from the real $P(W|do(X))$ if Fig. 4(b) is the real graph. Some equivalence class representations, such as the Partial Ancestral Graph representation (Spirtes et al., 2000), are robust to hidden common causes: in Fig. 4(c), an edge that has a circle as endpoint indicates that is not known if there is a causal path into both, for example, X and W (which would be the case for a hidden common cause of X and W). The arrow into W does indicate that W cannot be a cause of X. A fully directed edge such as $W \rightarrow Z$ indicates total information: W is a cause of Z, Z is not a cause of W, and W and Z have no hidden common causes.



Causality. Figure 4. (a) A particular causal graph with no other member on its equivalence class (assuming there are no hidden common causes). (b) Graph under the presence of two hidden common causes H_1 and H_2. (c) A representation for all graphs that entail the same conditional independencies as (a), without assuming the nonexistence of hidden common causes

Given equivalence class representations and background knowledge, different types of algorithms explore independence constraints to learn an equivalence class. It is typically assumed that the true graph is acyclic. The basic structure is to evaluate how well a set of conditional independence hypotheses are supported by the data. Depending on which constraints are judged to hold in the population, we keep, delete, or orient edges accordingly. Some algorithms, such as the PC algorithm (Spirtes et al., 2000), test a single independence hypothesis at a time, and assemble the individual outcomes in the end into an equivalence class representation. Other algorithms such as the GES algorithm (Chickering, 2002; Meek, 1997) start from a prior distribution for graphs and parameters, and proceed to compare the marginal likelihood of members of different equivalence classes (which can be seen as a Bayesian joint test of independence hypotheses). In the end, this reduces to a search for the maximum a posteriori equivalence class estimator. Both PC and GES have consistency properties: in the limit of infinite data, they return the right equivalence class under the faithfulness assumption. However, both PC and GES, and most causal discovery algorithms, assume that there are no hidden common causes in the domain. The fast causal inference (FCI) algorithm of Spirtes et al. (2000) is able to generate equivalence class representations as in Fig. 4(c). As in the PC algorithm, this is done by testing a single independence hypothesis at a time, and therefore is not very robust given small samples. A GES-like algorithm with the consistency properties of FCI is not currently known. An algorithm that allows for cyclic networks is discussed by Richardson (1996). More details of the fundamentals of structure learning algorithms are given by Scheines (1997).

Our examples relied on conditional independence constraints. In this case, the equivalence class is known as the *Markov equivalence class*. Markov equivalence classes are "nonparametric," in the sense that they do not refer to any particular probability family. In practice, this advantage is limited by our ability on evaluating independence hypotheses within flexible probability families. Another shortcoming of Markov equivalence classes is that they might be poorly informative if few independence constraints exist in the population. This

will happen, for instance, if a single hidden variable is a common cause of all observed variables. If one is willing to incorporate further assumptions, such as linearity of causal relationships, parametric constraints can be used to define other types of equivalence classes that are more discriminative than the Markov equivalence class. Silva, Scheines, Glymour, & Spirtes (2006) describe how some rank constraints in the covariance matrix of the observed variables can be used to learn the structure of linear models, even if no independence constraints are observable.

Evaluating the success of a structure learning algorithm is difficult, since ultimately it depends on interventional data. A promising area of application is molecular biology, where the large number of variables makes the use of graphical models a promising venue for decomposing complex biological systems, and for combining multiple sources of observational and interventional data. Sachs et al. (2005) describe a potential application, with further analysis discussed by Eaton and Murphy (2007). Other applications are discussed in the volume edited by Glymour and Cooper (1999).

Confidence Intervals Several causal learning algorithms such as the PC and FCI algorithms (Spirtes et al., 2000) are consistent, in the sense that they can recover the correct equivalence class given the faithfulness assumption and an infinite amount of data. Although point estimates of causal effects are important, it is also important to provide confidence intervals. Bayesian confidence intervals are readily available by having priors over parameters and graphs. ►Markov chain Monte Carlo algorithms, however, might be problematic due to the high-dimensional and discrete graph space. A practical algorithm that relies on a prior over *orderings* of variables (such that for a given ordering, a graph is not allowed to have vertex X as an ancestor of Y if Y antecedes X in the ordering) is given by Friedman and Koller (2003).

Such methods do not necessarily guarantee good frequentist properties. As a matter of fact, it has been shown that no such method can exist given the faithfulness assumption only (Robins, Scheines, Spirtes, & Wasserman, 2003). An intuitive explanation is as follows: consider the model such as the one in Fig. 2(a). For any given sample size, there is at least one model such that the associations due to the paths $X \leftarrow H \rightarrow Y$ and $X \rightarrow Y$ nearly cancel each other (faithfulness is still preserved), making the covariance of X and Y statistically undistinguishable from zero. In order to achieve uniform consistency, causal inference algorithms will need assumptions stronger than faithfulness. Zhang and Spirtes (2003) provide some directions.

Other Languages and Tasks in Causal Learning

A closely related language for representing causal models is the *potential outcomes* framework popularized by Rubin (2004). In this case, random variables for a same variable Y are defined for each possible state of the intervened variable X. Notice that, by definition, only one of the possible Y outcomes can be observed for any specific data point. This model is popular in statistics literature as a type of missing data model. The relation between potential outcomes and graphical models is discussed by Pearl (2000).

A case where potential outcomes become more clearly motivated is in *causal explanation*. In this setup, the model is asked for the probability that a particular event in time was the cause of a particular outcome. This is often cast as a *counterfactual question*: had A been false, would B still have happened? Questions in History and law are of this type: the legal responsibility of an airplane manufacturer in an accident depends on technical malfunction being an *actual cause* of the accident. Ultimately, such issues of causal explanation, actual causation and other counterfactual answers, are untestable. Although machine learning can be a useful tool to derive the consequences of assumptions combined with data about other events of the same type, in general the answers will not be robust to changes in the assumptions, and the proper assumptions ultimately cannot be selected with the available data. Some advances in generating explanations with causal models are described by Halpern and Pearl (2005).

Cross References

►Graphical Models
►Learning Graphical Models

Recommended Reading

Chickering, D. (2002). Optimal structure identification with greedy search. *Journal of Machine Learning Research, 3*, 507–554.
Cooper, G., & Yoo, C. (1999). Causal discovery from a mixture of experimental and observational data. In *Uncertainty in Artificial Intelligence (UAI)*.

Dawid, A. P. (2002). Influence diagrams for causal modelling and inference. *International Statistical Review, 70*, 161–189.
Eaton, D., & Murphy, K. (2007). Exact Bayesian structure learning from uncertain interventions. In *Artificial Intelligence and Statistics (AISTATS)*.
Eberhardt, F., Glymour, C., & Scheines, R. (2005). On the number of experiments sufficient and in the worst case necessary to identify all causal relations among N variables. In *Uncertainty in Artificial Intelligence (UAI)* (pp. 178–184).
Friedman, N., & Koller, D. (2003). Being Bayesian about network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning, 50*, 95–126.
Glymour, C., & Cooper, G. (1999). *Computation, causation and discovery*. Cambridge, MA: MIT Press.
Halpern, J., & Pearl, J. (2005). Causes and explanations: A structural-model approach. Part II: Explanations. *British Journal for the Philosophy of Science, 56*, 889–911.
Meek, C. (1997). *Graphical models: Selecting causal and statistical models*. PhD thesis, Carnegie Mellon University.
Pearl, J. (2000). *Causality: Models, reasoning and inference*. Cambridge: Cambridge University Press.
Richardson, T. (1996). A discovery algorithm for directed cyclic graphs. In *Proceedings of 12th conference on Uncertainty in Artificial Intelligence*.
Robins, J., Scheines, R., Spirtes, P., & Wasserman, L. (2003). Uniform consistency in causal inference. *Biometrika, 90*, 491–515.
Rosenbaum, P. (2002). *Observational studies*. New York: Springer.
Rubin, D. (2004). Direct and indirect causal effects via potential outcomes. *Scandinavian Journal of Statistics, 31*, 161–170.
Sachs, K., Prez, O., Pe'er, D., Lauffenburger, D., & Nolan, G. (2005). Causal protein-signaling networks derived from multiparameter single-cell data. *Science, 308*, 523–529.
Scheines, R. (1997). An introduction to causal inference. In V. McKim & S. Turner (Eds.), *Causality in Crisis?* (pp. 185–200).
Silva, R., Scheines, R., Glymour, C., & Spirtes, P. (2006). Learning the structure of linear latent variable models. *Journal of Machine Learning Research, 7*, 191–246.
Spirtes, P., Glymour, C., & Scheines, R. (2000). *Causation, prediction and search*. Cambridge, MA: Cambridge University Press.
Wasserman, L. (2004). *All of statistics*. New York: Springer.
Zhang, J., & Spirtes, P. (2003). Strong faithfulness and uniform consistency in causal inference. In *Uncertainty in Artificial Intelligence*.

CBR

►Case-Based Reasoning

CC

►Cascade-Correlation

Certainty Equivalence Principle

►Internal Model Control

Characteristic

►Attribute

City Block Distance

►Manhattan Distance

Class

Chris Drummond
National Research Council of Canada

Synonyms

Category; Class; Collection; Kind; Set; Sort; Type

Definition

A class is a collection of things that might reasonably be grouped together. Classes that we commonly encounter have simple names so, as humans, we can easily refer to them. The class of dogs, for example, allows me to say "my dog ate my newspaper" without having to describe a particular dog, or indeed, a particular newspaper. In machine learning, the name of the class is called the class label. Exactly what it means to belong to a class, or category, is a complex philosophical question but often we think of a class in terms of the common properties of its members. We think particularly of those properties which seperate them from other things which are in many ways similar, e.g., cats mieow and dogs bow-wow. We would be unlikely to form a class from a random collection of things, as they would share no common properties. Knowing something belonged to such a collection would be of no particular benefit. Although

many every day classes will have simple names, we may construct them however we like, e.g., "The things I like to eat for breakfast on a Saturday morning." As there is no simple name for such a collection, in machine learning we would typically refer to it as the positive class, and all occurences of it are positive examples; the negative class would be everything else.

Motivation and Background

The idea of a class is important in learning. If we discover something belongs to a class, we suddenly know quite a lot about it even if we have not encountered that particular example before. In machine learning, our use of the term accords closely with the mathematical definition of a class, as a collection of sets unambiguously defined by a property that all its members share. It also accords with the idea of equivalence classes, which group similar things. Sets have an intension, the description of what it means to be a member, and an extension, things that belong to the set, useful properties of a class in machine learning. Class is also a term used extensively in knowledge bases to denote an important relationship between groups, of sub-class and super class. Learning is often viewed as a way of solving the knowledge acquisition bottleneck (Buchanan et al., 1983) in knowledge bases and the use of the term class in machine learning highlights this connection.

Recommended Reading

Buchanan, B., Barstow, D., Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., et al. (1983) Constructing an expert system. In F. Hayes-Roth, D.A. Waterman, & D.B. Lenat (Eds.), *Building expert systems* (pp. 127–167). Reading, MA: Addison-Wesley.

Class Imbalance Problem

Charles X. Ling, Victor S. Sheng
The University of Western Ontario
Canada

Definition

Data are said to suffer the *Class Imbalance Problem* when the class distributions are highly imbalanced. In this context, many ►classification learning algorithms have low ►predictive accuracy for the infrequent class. ►Cost-sensitive learning is a common approach to solve this problem.

Motivation and Background

Class imbalanced datasets occur in many real-world applications where the class distributions of data are highly imbalanced. For the two-class case, without loss of generality, one assumes that the minority or rare class is the positive class, and the majority class is the negative class. Often the minority class is very infrequent, such as 1% of the dataset. If one applies most traditional (cost-insensitive) classifiers on the dataset, they are likely to predict everything as negative (the majority class). This was often regarded as a problem in learning from highly imbalanced datasets.

However, Provost (2000) describes two fundamental assumptions that are often made by traditional cost-insensitive classifiers. The first is that the goal of the classifiers is to maximize the accuracy (or minimize the error rate); the second is that the class distribution of the training and test datasets is the same. Under these two assumptions, predicting everything as negative for a highly imbalanced dataset *is often the right thing to do*. Drummond and Holte (2000) show that it is usually very difficult to outperform this simple classifier in this situation.

Thus, the imbalanced class problem becomes meaningful only if one or both of the two assumptions above are not true; that is, if the cost of different types of error (false positive and false negative in the binary classification) is not the same, or if the class distribution in the test data is different from that of the training data. The first case can be dealt with effectively using methods in cost-sensitive meta-learning (see ►Cost-sensitive learning).

In the case when the misclassification cost is not equal, it is usually more expensive to misclassify a minority (positive) example into the majority (negative) class, than a majority example into the minority class (otherwise it is more plausible to predict everything as negative). That is, $FNcost > FPcost$. Thus, given the values of *FNcost* and *FPcost*, a variety of cost-sensitive meta-learning methods can be, and have been, used to solve the class imbalance problem (Japkowicz & Stephen, 2002; Ling & Li, 1998). If the values of

FNcost and *FPcost* are not unknown explicitly, *FNcost* and *FPcost* can be assigned to be proportional to the number of positive and negative training cases (Japkowicz & Stephen, 2002).

In case the class distributions of training and test datasets are different (e.g., if the training data is highly imbalanced but the test data is more balanced), an obvious approach is to sample the training data such that its class distribution is the same as the test data. This can be achieved by oversampling (creating multiple copies of examples of) the minority class and/or undersampling (selecting a subset of) the majority class (Provost, 2000).

Note that sometimes the number of examples of the minority class is too small for classifiers to learn adequately. This is the problem of insufficient (small) training data and different from that of imbalanced datasets.

Recommended Reading

Drummond, C., & Holte, R. (2000). Exploiting the cost (in)sensitivity of decision tree splitting criteria. In *Proceedings of the seventeenth international conference on machine learning* (pp. 239–246).

Drummond, C., & Holte, R. (2005). Severe class imbalance: Why better algorithms aren't the answer. In *Proceedings of the sixteenth European conference of machine learning, LNAI* (Vol. 3720, pp. 539–546).

Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5), 429–450.

Ling, C. X., & Li, C. (1998). Data mining for direct marketing – Specific problems and solutions. In *Proceedings of fourth international conference on Knowledge Discovery and Data Mining (KDD-98)* (pp. 73–79).

Provost, F. (2000). Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI'2000 workshop on imbalanced data*.

Classification

Chris Drummond
National Research Council of Canada

Synonyms

Categorization; Generalization; Identification; Induction; Recognition

Definition

In common usage, the word classification means to put things into categories, group them together in some useful way. If we are screening for a disease, we would group people into those with the disease and those without. We, as humans, usually do this because things in a group, called a ▶class in machine learning, share common characteristics. If we know the class of something, we know a lot about it. In machine learning, the term classification is most commonly associated with a particular type of learning where examples of one or more ▶classes, labeled with the name of the class, are given to the learning algorithm. The algorithm produces a classifier which maps the properties of these examples, normally expressed as ▶attribute-value pairs, to the class labels. A new example whose class is unknown is classified when it is given a class label by the classifier based on its properties. In machine learning, we use the word classification because we call the grouping of things a class. We should note, however, that other fields use different terms. In philosophy and statistics, the term categorization is more commonly used. In many areas, in fact, classification often refers to what is called ▶clustering in machines learning.

Motivation and Background

Classification is a common, and important, human activity. Knowing something's class allows us to predict many of its properties and so act appropriately. Telling other people its class allows them to do the same, making for efficient communication. This emphasizes two commonly held views of the objectives of learning. First, it is a means of ▶generalization, to predict accurately the values for previously unseen examples. Second, it is a means of compression, to make transmission or communication more efficient. Classification is certainly not a new idea and has been studied for some considerable time. From the days of the early Greek philosophers such as Socrates, we had the idea of categorization. There are essential properties of things that make them what they are. It embodies the idea that there are natural kinds, ways of grouping things, that are inherent in the world. A major goal of learning, therefore, is recognizing natural kinds, establishing the necessary and sufficient conditions for belonging to a category. This "classical" view of categorization, most

often attributed to Aristotle, is now strongly disputed. The main competitor is prototype theory; things are categorized by their similarity to a prototypical example (Lakoff, 1987), either real or imagined. There is also much debate in psychology (Ashby & Maddox, 2005), where many argue that there is no single method of categorization used by humans.

As much of the inspiration for machine learning originated in how humans learn, it is unsurprising that our algorithms reflect these distinctions. ▶Nearest neighbor algorithms would seem to have much in common with prototype theory. These have been part of pattern recognition for some time (Cover & Hart, 1967) and have become popular in machine learning, more recently, as ▶instance-based learners (Aha, Kiber, & Albert, 1991). In machine learning, we measure the distance to one or more members of a concept rather a specially constructed prototype. So, this type of learning is perhaps more a case of the exemplar learning found in the psychological literature, where multiple examples represent a category. The closest we have to prototype learning occurs in clustering, a type of ▶unsupervised learning, rather than classification. For example, in ▶k-means clustering group membership is determined by closeness to a central value.

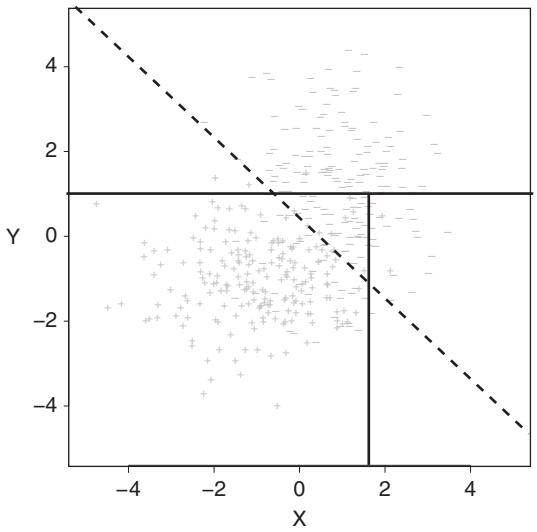
In the early days of machine learning, our algorithms (Mitchell, 1977; Winston, 1975) had much in common with the classical theory of categorization in philosophy and psychology. It was assumed that the data were consistent, there were no examples with the same attribute values but belonging to different classes. It was quickly realized that, even if the properties where necessary and sufficient to capture the class, there was often noise in the attribute and perhaps the class values. So, complete consistency was seldom attainable in practice. New ▶classification algorithms were designed, which could tolerate some noise, such as ▶decision trees (Breiman, Friedman, Olshen, & Stone, 1984; Quinlan, 1986, 1993) and rule-based learners (see ▶Rule Learning) (Clark & Niblett, 1989; Holte, 1993; Michalski, 1983).

Structure of the Learning System

Whether one uses instance-based learning, rule-based learning, decision trees, or indeed any other classification algorithm, the end result is the division of the input space into regions belonging to a single class. The input space is defined by the Cartesian product of the attributes, all possible combinations of possible values.

As a simple example, Fig. 1 shows two classes + and −, each a random sample of a normal distribution. The attributes are X and Y of real type. The values for each attribute range from $\pm\infty$. The figure shows a couple of alternative ways that the space may be divided into regions. The bold dark lines, construct regions using lines that are parallel to the axes. New examples that have Y less than 1 and X less than 1.5 with be classified as +, all others classified as −. Decision trees and rules form this type of boundary. A ▶linear discriminant function, such as the bold dashed line, would divide the space into half-spaces, with new examples below the line being classified as + and those above as −. Instance-based learning will also divide the space into regions but the boundary is implicit. Classification occurs by choosing the class of the majority of the nearest neighbors to a new example. To make the boundary explicit, we could mark the regions where an example would be classified as + and those classified as −. We would end up with regions bounded by polygons.

What differs among the algorithms is the shape of the regions, and how and when they are chosen. Sometimes the regions are implicit as in lazy learners (see ▶Lazy Learning) (Aha, 1997), where the boundaries are not decided until a new example is being classified.



Classification. Figure 1. Dividing the input space

Sometimes the regions are determined by decision theory as in generative classifiers (see ►Generative Learners) (Rubinstein & Hastie, 1997), which model the full joint distribution of the classes. For all classifiers though, the input space is effectively partitioned into regions representing a single class.

Applications

One of the reasons that classification is an important part of machine learning is that it has proved to be a very useful technique for solving practical problems. Classification has been used to help scientists in the exploration, and comprehension, of their particular domains of interest. It has also been used to help solve significant industrial problems. Over the years a number of authors have stressed the importance of applications to machine learning and listed many successful examples (Brachman, Khabaza, Kloesgen, Piatetsky-Shapiro, & Simoudis, 1996; Langley & Simon, 1995; Michie, 1982). There have also been workshops on applications (Aha & Riddle, 1995; Engels, Evans, Herrmann, & Verdenius, 1997; Kodratoff, 1994) at major machine learning conferences and a special issue of Machine Learning (Kohavi & Provost, 1998), one of the main journals in the field. There are now conferences that are highly focused on applications. Collocated with major artificial intelligence conferences is the Innovative Applications of Artificial Intelligence conference. Since 1989, this conference has highlighted practical applications of machine learning, including classification (Schorr & Rappaport, 1989). In addition, there are now at least two major knowledge discovery and ►data mining conferences (Fayyad & Uthurusamy, 1995; Komorowski & Zytkow, 1997) with a strong focus on applications.

Future Directions

In machine learning, there are already a large number of different classification algorithms, yet new ones still appear. It seems unlikely that there is an end in sight. The "no free lunch theory" (Wolpert & Macready, 1997) indicates that there will never be a single best algorithm, better than all others in terms of predictive power. However, apart from their predictive performance, each classifier has its own attractive properties which are important to different groups of people. So, new algorithms are still of value. Further, even if we are solely concerned about performance, it may be useful to have many different algorithms, all with their own biases (see ►Inductive Bias). They may be combined together to form an ensemble classifier (Caruana, Niculescu-Mizil, Crew, & Ksikes, 2004), which outperforms single classifiers of one type (see ►Ensemble Learning).

Limitations

Classification has been critical part of machine research for some time. There is a concern that the emphasis on classification, and more generally on ►supervised learning, is too strong. Certainly much of human learning does not use, or require, labels supplied by an expert. Arguably, unsupervised learning should play a more central role in machine learning research. Although classification does require a label, it does necessarily need an expert to provide labeled examples. Many successful applications rely on finding some, easily identifiable, property which stands in for the class.

Recommended Reading

Aha, D. W. (1997). Editorial. *Artificial Intelligence Review, 11*(1–5), 1–6.

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*(1), 37–66.

Aha, D. W., & Riddle, P. J. (Eds.). (1995). Workshop on applying machine learning in practice. In *Proceedings of the 12th international conference on machine learning*.

Ashby, F. G., & Maddox, W. T. (2005). Human category learning. *Annual Review of Psychology, 56*, 149–178.

Bishop, C. M. (2007). *Pattern recognition and machine learning*. New York: Springer.

Brachman, R. J., Khabaza, T., Kloesgen, W., Piatetsky-Shapiro, G., & Simoudis, E. (1996). Mining business databases. *Communications of the ACM, 39*(11), 42–48.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth.

Caruana, R., Niculescu-Mizil, A., Crew, G., & Ksikes, A. (2004). Ensemble selection from libraries of models. In *Proceedings of the 21st international conference on machine learning* (pp. 137–144).

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*, 261–284.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory, 13*, 21–27.

Dietterich, T., & Shavlik, J. (Eds.). *Readings in machine learning*. San Mateo, CA: Morgan Kaufmann.

Engels, R., Evans, B., Herrmann, J., & Verdenius, F. (Eds.). (1997). Workshop on machine learning applications in the real world;

methodological aspects and implications. In *Proceedings of the 14th international conference on machine learning*.
Fayyad, U. M., & Uthurusamy, R. (Eds.). (1995). *Proceedings of the first international conference on knowledge discovery and data mining*.
Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning, 11*(1), 63–91.
Kodratoff, Y. (Ed.). (1994). *Proceedings of MLNet workshop on industrial application of machine learning*.
Kodratoff, Y., & Michalski, R. S. (1990). *Machine learning: An artificial intelligence approach, (Vol. 3)*. San Mateo, CA: Morgan Kaufmann.
Kohavi, R., & Provost, F. (1998). Glossary of terms. Editorial for the special issue on applications of machine learning and the knowledge discovery process. *Machine Learning, 30*(2/3).
Komorowski, H. J., & Zytkow, J. M. (Eds.). (1997). *Proceedings of the first European conference on principles of data mining and knowledge discovery*.
Lakoff, G. (1987). *Women, fire and dangerous things*. Chicago, IL: University of Chicago Press.
Langley, P., & Simon, H. A. (1995). Applications of machine learning and rule induction. *Communications of the ACM, 38*(11), 54–64.
Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. S. Michalski, T. J. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 83–134). Palo Alto, CA: TIOGA Publishing.
Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.). (1983). *Machine learning: An artificial intelligence approach*. Palo Alto, CA: Tioga Publishing Company.
Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.). (1986). *Machine learning: An artificial intelligence approach, (Vol. 2)*. San Mateo, CA: Morgan Kaufmann.
Michie, D. (1982). *Machine intelligence and related topics*. New York: Gordon and Breach Science Publishers.
Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the fifth international joint conferences on artificial intelligence* (pp. 305–310).
Mitchell, T. M. (1997). *Machine learning*. Boston, MA: McGraw-Hill.
Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.
Quinlan, J. R. (1993). *C4.5 programs for machine learning*. San Mateo, CA: Morgan Kaufmann.
Rubinstein, Y. D., & Hastie, T. (1997). Discriminative vs informative learning. In *Proceedings of the third international conference on knowledge discovery and data mining* (pp. 49–53).
Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach*. Upper Saddle River, NJ: Prentice-Hall.
Schorr, H., & Rappaport, A. (Eds.). (1989). *Proceedings of the first conference on innovative applications of artificial intelligence*.
Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision* (pp. 157–209). New York: McGraw-Hill.
Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. San Fransisco: Morgan Kaufmann.
Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.

Classification Algorithms

There is a very large number of classification algorithms, including ▶decision trees, ▶instance-based learners, ▶support vector machines, ▶rule-based learners, ▶neural networks, ▶Bayesian networks. There also ways of combining them into ▶ensemble classifiers such as ▶boosting, ▶bagging, ▶stacking, and ▶forests of trees.

To delve deeper into classifiers and their role in machine learning, a number of books are recommended covering machine learning (Bishop, 2007; Mitchell, 1997; Witten & Frank, 2005) and artificial intelligence (Russell & Norvig, 2003) in general. Seminal papers on classifiers can be found in collections of papers on machine learning (Dietterich & Shavlik, 1990; Kodratoff & Michalski, 1990; Michalski, Carbonell, & Mitchell, 1983, 1986).

Recommended Reading

Bishop, C. M. (2007). *Pattern recognition and machine learning*. New York: Springer.
Dietterich, T., & Shavlik, J. (Eds.). *Readings in machine learning*. San Mateo, CA: Morgan Kaufmann.
Kodratoff, Y., & Michalski, R. S. (1990). *Machine learning: An artificial intelligence approach, (Vol. 3)*. San Mateo, CA: Morgan Kaufmann.
Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.). (1983). *Machine learning: An artificial intelligence approach*. Palo Alto, CA: Tioga Publishing Company.
Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.). (1986). *Machine learning: An artificial intelligence approach, (Vol. 2)*. San Mateo, CA: Morgan Kaufmann.
Mitchell, T. M. (1997). *Machine learning*. Boston, MA: McGraw-Hill.
Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach*. Upper Saddle River, NJ: Prentice-Hall.
Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. San Fransisco: Morgan Kaufmann.

Classification Learning

▶Concept Learning

Classification Tree

▶Decision Tree

Classifier Systems

Pier Luca Lanzi
Politecnico di Milano, Milano, Italy

Synonyms

Genetics-based machine learning; Learning classifier systems

Definition

Classifier systems are rule-based systems that combine ►temporal difference learning or ►supervised learning with a genetic algorithm to solve classification and ►reinforcement learning problems. Classifier systems come in two flavors: Michigan classifier systems, which are designed for online learning, but can also tackle offline problems; and Pittsburgh classifier systems, which can only be applied to offline learning.

In Michigan classifier systems (Holland, 1976), learning is viewed as an online adaptation process to an unknown environment that represents the problem and provides feedback in terms of a numerical reward. Michigan classifier systems maintain a single candidate solution consisting of a set of rules, or a population of classifiers. Michigan systems apply (1) temporal difference learning to distribute the incoming reward to the classifiers that are accountable for it; and (2) a genetic algorithm to select, recombine, and mutate individual classifiers so as to improve their contribution to the current solution.

In contrast, in Pittsburgh classifier systems (Smith, 1980), learning is viewed as an offline optimization process in which a genetic algorithm alone is applied to search for the best solution to a given problem. In addition, Pittsburgh classifier systems maintain not one, but a set of candidate solutions. While in the Michigan classifier system each individual classifier represents a part of the overall solution, in the Pittsburgh system each individual is a complete candidate solution (itself consisting of a set of classifiers). The fitness of each Pittsburgh individual is computed offline by testing it on a representative sample of problem instances. The individuals compete among themselves through selection, while crossover and mutation recombine solutions to search for better solutions.

Motivation and Background

Machine learning is usually viewed as a search process in which a solution space is explored until an appropriate solution to the target problem is found (Mitchell, 1982) (see ►Learning as Search). Machine learning methods are characterized by the way they represent solutions (e.g., using ►decision trees, rules), by the way they evaluate solutions (e.g., classification accuracy, information gain) and by the way they explore the solution space (e.g., using a ►general-to-specific strategy or a ►specific-to-general strategy).

Classifier systems are methods of *genetics-based* machine learning introduced by Holland, the father of ►genetic algorithms. They made their first appearance in Holland (1976) where the first diagram of a classifier system, labeled "cognitive system," was shown. Subsequently, they were described in detail in the paper "Cognitive Systems based on Adaptive Algorithms" (Holland and Reitman, 1978). Classifier systems are characterized by a rule-based representation of solutions and a genetics-based exploration of the solution space. While other ►rule learning methods, such as CN2 (Clark & Niblett, 1989) and FOIL (Quinlan & Cameron-Jones, 1995), generate one rule at a time following a sequential covering strategy (see ►Covering Algorithm), classifier systems work on one or more solutions at once, and they explore the solution space by applying the principles of natural selection and genetics.

In classifier systems (Holland, 1976; Holland and Reitman, 1978; Wilson, 1995), machine learning is modeled as an online adaptation process to an unknown *environment*, which provides feedback in terms of a numerical reward. A classifier system perceives the environment through its detectors and, based on its sensations, it selects an action to be performed in the environment through its effectors. Depending on the efficacy of its actions, the environment may eventually reward the system. A classifier system learns by trying to maximize the amount of reward it receives from the environment. To pursue such a goal, it maintains a set (a *population*) of condition-action-prediction rules, called *classifiers*, which represents the current solution. Each classifier's condition identifies some part of the problem domain; the classifier's action represents a decision on the subproblem identified by its condition; and the classifier's prediction, or strength, estimates the value of the action in terms of future

rewards on that subproblem. Two separate components, credit assignment and rule discovery, act on the population with different goals. ▶Credit assignment, implemented either by methods of temporal difference or supervised learning, exploits the incoming reward to estimate the action values in each subproblem so as to identify the best classifiers in the population. At the same time, rule discovery, usually implemented by a genetic algorithm, selects, recombines, and mutates the classifiers in the population to improve the current solution.

Classifier systems were initially conceived as modeling tools. Given a real system with unknown underlying dynamics, for instance a financial market, a classifier system would be used to generate a behavior that matched the real system. The evolved rules would provide a plausible, human readable model of the unknown system – a way to look inside the box. Subsequently, with the developments in the area of machine learning and the rise of reinforcement learning, classifier systems have been more and more often studied and presented as alternatives to other machine learning methods. Wilson's XCS (1995), the most successful classifier system to date, has proven to be both a valid alternative to other reinforcement learning approaches and an effective approach to classification and data mining (Bull, 2004; Bull & Kovacs, 2005; Lanzi, Stolzmann, & Wilson, 2000).

Kenneth de Jong and his students (de Jong, 1988; Smith, 1980, 1983) took a different perspective on genetics-based machine learning and modeled learning as an *optimization* process rather than an *adaptation* process as done in Holland (1976). In this case, the solution space is explored by applying a genetic algorithm to a population of individuals each representing a *complete* candidate solution – that is, a set of rules (or a production system, de Jong, 1988; Smith, 1980). At each cycle, a critic is applied to each individual (to each set of rules) to obtain a performance measure that is then used by the genetic algorithm to guide the exploration of the solution space. The individuals in the population compete among themselves through selection, while crossover and mutation recombine solutions to search for better ones.

The approaches of Holland (Holland, 1976; Holland and Reitman, 1978) and de Jong (de Jong, 1988; Smith, 1980, 1983) have been extended and improved in several ways (see Lanzi et al. (2000) for a review). The models of classifier systems that are inspired by the work of Holland (1976) at the University of Michigan are usually called Michigan classifier systems; the ones that are inspired by Smith (1980, 1983) and de Jong (1988) at the University of Pittsburgh are usually termed Pittsburgh classifier systems – or briefly, Pitt classifier systems.

Pittsburgh classifier systems separate the evaluation of candidate solutions, performed by an external critic, from the genetic search. As they evaluate candidate solutions as a whole, Pittsburgh classifier systems can easily identify and emphasize sequentially cooperating classifiers, which is particularly helpful in problems involving partial observability. In contrast, in Michigan classifier systems the credit assignment is focused, due to identification of the actual classifiers that produce the reward, so learning is *much* faster but sequentially cooperating classifiers are more difficult to spot. As Pittsburgh classifier systems apply the genetic algorithm to a set of solutions, they only work offline, whereas Michigan classifier systems work online, although they can also tackle offline problems. Finally, the design of Pittsburgh classifier systems involves decisions as to how an entire solution should be represented and how solutions should be recombined – a task which can be daunting. In contrast, the design of Michigan classifier systems involves simpler decisions about how a rule should be represented and how two rules should be recombined. Accordingly, while the representation of solutions and its related issues play a key role in Pittsburgh models, Michigan models easily work with several types of representations (Lanzi, 2001; Lanzi & Perrucci, 1999; Mellor, 2005).

Structure of the Learning System

Michigan and Pittsburgh classifier systems were both inspired by the work of Holland on the broadcast language (Holland, 1975). However, their structures reflect two different ways to model machine learning: as an adaptation process in the case of Michigan classifier systems; and as an optimization problem, in the case of Pittsburgh classifier systems. Thus, the two models, originating from the same idea (Holland's broadcast language), have radically different structures.

Michigan Classifier Systems

Holland's classifier systems define a general paradigm for genetics-based machine learning. The description in Holland and Reitman (1978) provides a list of principles for online learning through adaptation. Over the years, such principles have guided researchers who developed several models of Michigan classifier systems (Butz, 2002; Wilson, 1994, 1995, 2002) and applied them to a large variety of domains (Bull, 2004; Lanzi & Riolo, 2003; Lanzi et al., 2000). These models extended and improved Holland's original ideas, but kept all the ingredients of the original recipe: a population of classifiers, which represents the current system knowledge; a performance component, which is responsible for the short-term behavior of the system; a credit assignment (or reinforcement) component, which distributes the incoming reward among the classifiers; and a rule discovery component, which applies a genetic algorithm to the classifiers to improve the current knowledge.

Knowledge Representation

In Michigan classifier systems, knowledge is represented by a population of classifiers. Each classifier is usually defined by four main parameters: the *condition*, which identifies some part of the problem domain; the *action*, which represents a decision on the subproblem identified by its condition; the *prediction* or strength, which estimates the amount of reward that the system will receive if its action is performed; and finally, the *fitness*, which estimates how good the classifier is in terms of problem solution.

The knowledge representation of Michigan classifier systems is extremely flexible. Each one of the four classifier components can be tailored to fit the need of a particular application, without modifying the main structure of the system. In problems involving binary inputs, classifier conditions can be simply represented using strings defined over the alphabet $\{0, 1, \#\}$, as done in Holland and Reitman (1978), Goldberg (1989), and Wilson (1995). In problems involving real inputs, conditions can be represented as disjunctions of intervals, similar to the ones produced by other rule learning methods (Clark & Niblett, 1989) Conditions can also be represented as general-purpose symbolic expressions (Lanzi, 2001; Lanzi & Perrucci, 1999) or first-order logic expressions (Mellor, 2005). Classifier actions are typically encoded by a set of symbols (either binary strings or simple labels), but continuous real-valued actions are also available (Wilson, 2007). Classifier prediction (or strength) is usually encoded by a parameter (Goldberg, 1989; Holland & Reitman, 1978; Wilson, 1995). However, classifier prediction can also be computed using a parameterized function (Wilson, 2002), which results in solutions represented as an ensemble of local approximators – similar to the ones produced in generalized reinforcement learning (Sutton & Barto, 1998).

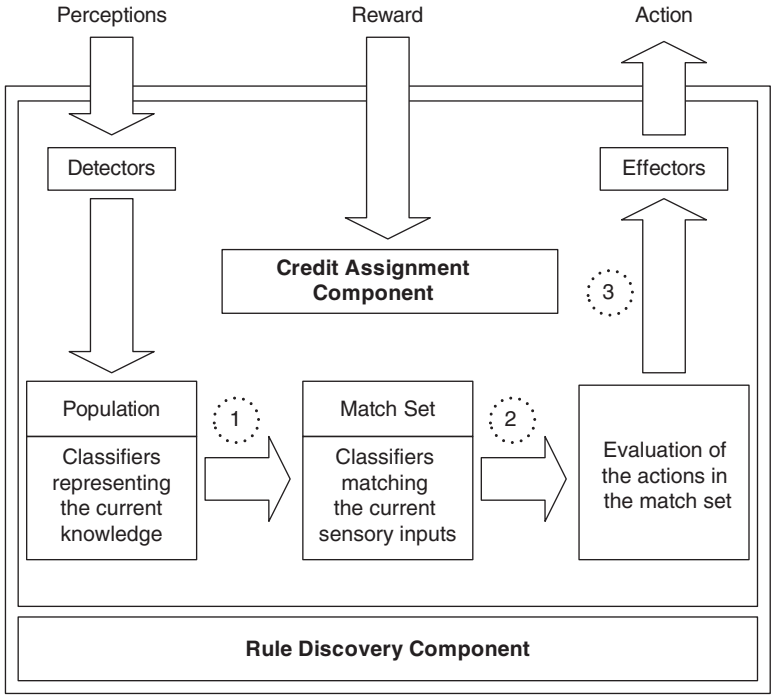
Performance Component

A simplified structure of Michigan classifier systems is shown in Fig. 1. We refer the reader to Goldberg (1989) and Holland and Reitman (1978) for a detailed description of the original model and to Butz (2002) and Wilson (1994, 1995, 2001) for descriptions of recent classifier system models.

A classifier system learns through trial and error interactions with an unknown environment. The system and the environment interact continually. At each time step, the classifier system perceives the environment through its detectors; it builds a *match set* containing all the classifiers in the population whose condition matches the current sensory input. The match set typically contains classifiers that advocate contrasting actions; accordingly, the classifier system evaluates each action in the match set, and selects an action to be performed balancing exploration and exploitation. The selected action is sent to the effectors to be executed in the environment; depending on the effect that the action has in the environment, the system receives a scalar reward.

Credit Assignment

The *credit assignment component* (also called reinforcement component, Wilson 1995) distributes the incoming reward to the classifiers that are accountable for it. In Holland and Reitman (1978), credit assignment is implemented by Holland's bucket brigade algorithm (Holland, 1986), which was partially inspired by the credit allocation mechanism used by Samuel in his



Classifier Systems. Figure 1. Simplified structure of a Michigan classifier system. The system perceives the environment through its detectors and (1) it builds the match set containing the classifiers in the population that match the current sensory inputs; then (2) all the actions in the match set are evaluated, and (3) an action is selected to be performed in the environment through the effectors

pioneering work on learning checkers-playing programs (Samuel, 1959).

In the early years, classifier systems and the bucket brigade algorithm were confined to the evolutionary computation community. The rise of reinforcement learning increased the connection between classifier systems and temporal difference learning (Sutton, 1988; Sutton & Barto, 1998): in particular, Sutton (1988) showed that the bucket brigade algorithm is a kind of temporal difference learning, and similar connections were also made in Watkins (1989) and Dorigo and Bersini (1994). Later, the connection between classifier systems and reinforcement learning became tighter with the introduction of Wilson's XCS (1995), in which credit assignment is implemented by a modification of Watkins Q-learning (Watkins, 1989). As a consequence, in recent years, classifier systems are often presented as methods of reinforcement learning with genetics-based generalization (Bull & Kovacs, 2005).

Rule Discovery Component

The *rule discovery component* is usually implemented by a genetic algorithm that selects classifiers in the population with probability proportional to their fitness; it copies the selected classifiers and applies genetic operators (usually crossover and mutation) to the offspring classifiers; the new classifiers are inserted in the population, while other classifiers are deleted to keep the population size constant.

Classifiers selection plays a central role in rule discovery. Classifier selection depends on the definition of classifier fitness and on the subset of classifiers considered during the selection process. In Holland and Reitman (1978), classifier fitness coincides with classifier prediction, while selection is applied to all the classifiers in the population. This approach results in a pressure toward classifiers predicting high returns, but typically tends to produce overly general solutions. To avoid such solutions, Wilson (1995) introduced the XCS classifier system in which accuracy-based fitness is

coupled with a niched genetic algorithm. This approach results in a pressure toward accurate maximally general classifiers, and has made XCS the most successful classifier system to date.

Pittsburgh Classifier Systems

The idea underlying the development of Pittsburgh classifier systems was to show that interesting behaviors could be evolved using a simpler model than the one proposed by Holland with Michigan classifier systems (Holland, 1976; Holland & Reitman, 1978).

In Pittsburgh classifier systems, each individual is a set of rules that encodes an entire candidate solution; each rule has a fixed length, but each rule set (each individual) usually contains a variable number of rules. The genetic operators, crossover and mutation, are tailored to the rule-based, variable-length representation. The individuals in the population compete among themselves, following the selection-recombination-mutation cycle that is typical of genetic algorithms (Goldberg, 1989; Holland, 1975). While in Michigan classifier systems individuals in the population (the single rules) cooperate, in Pittsburgh classifier systems there is no cooperation among individuals (the rule sets), so that the genetic algorithm operation is simpler for Pittsburgh models. However, as Pittsburgh classifier systems explore a much larger search space, they usually require more computational resources than Michigan classifier systems.

The pseudo-code of a Pittsburgh classifier system is shown in Fig. 2. At first, the individuals in the population are randomly initialized (line 2). At time t, the individuals are evaluated by an external critic, which returns a performance measure that the genetic algorithm exploits to compute the fitness of individuals (lines 3 and 10). Following this, selection (line 6), recombination, and mutation (line 7) are applied to the individuals in the population – as done in a typical genetic algorithm. The process stops when a termination criterion is met (line 4), usually when an appropriate solution is found.

The design of Pittsburgh classifier systems follows the typical steps of genetic algorithm design, which means deciding how a rule set should be represented, what genetic operators should be applied, and how the fitness of a set of rules should be calculated. In addition, Pittsburgh classifier systems need to address the *bloat* phenomenon (Tackett, 1994) that arises with any variable-sized representation, like the rule sets evolved by Pittsburgh classifier systems. Bloat can be defined as the growth of individuals without an actual fitness improvement. In Pittsburgh classifier systems, bloat increases the size of candidate solutions by adding useless rules to individuals, and it is typically limited by introducing a parsimony pressure that discourages large rule sets (Bassett & de Jong, 2000). Alternatively, Pittsburgh classifier systems can be combined with multi-objective optimization, so as to separate the maximization of the rule set performance and the minimization of the rule set size.

Examples of Pittsburgh classifier systems include SAMUEL (Grefenstette, Ramsey, & Schultz, 1990), the Genetic Algorithm Batch-Incremental Concept Learner (GABIL) (de Jong & Spears, 1991), GIL (Janikow, 1993), GALE (Llorà, 2002), and GAssist (Bacardit, 2004).

```
1.   t := 0
2.   Initialize the population P(t)
3.   Evaluate the rules sets in P(t)
4.   While the termination condition is not satisfied
5.   Begin
6.      Select the rule sets in P(t) and generate Ps(t)
7.      Recombine and mutate the rule sets in Ps(t)
8.      P(t+1) := Ps(t)
9.      t := t+1
10.     Evaluate the rules sets in P(t)
11.  End
```

Classifier Systems. Figure 2. Pseudo-code of a Pittsburgh classifier system

Applications

Classifier systems have been applied to a large variety of domains, including computational economics (e.g., Arthur, Holland, LeBaron, Palmer, & Talyer, 1996), autonomous robotics (e.g., Dorigo & Colombetti, 1998), classification (e.g., Barry, Holmes, & Llora, 2004), fighter aircraft maneuvering (Bull, 2004; Smith, Dike, Mehra, Ravichandran, & El-Fallah, 2000), and many others. Reviews of classifier system applications are available in Lanzi et al. (2000), Lanzi and Riolo (2003), and Bull (2004).

Programs and Data

The major sources of information about classifier systems are the LCSWeb maintained by Alwyn Barry, which can be reached through, and www.learning-classifier-systems.org_maintained by Xavier Llorà.

Several implementations of classifier systems are freely available online. The first standard implementation of Holland's classifier system in Pascal was described in Goldberg (1989), and it is available at http://www.illigal.org/; a C version of the same implementation, developed by Robert E. Smith, is available at http://www.etsimo.uniovi.es/ftp/pub/EC/CFS/src/. Another implementation of an extension of Holland's classifier system in C by Rick L. Riolo is available at http://www.cscs.umich.edu/Software/Contents.html. Implementations of Wilson's XCS (1995) are distributed by Alwyn Barry at the LCSWeb, by Martin V. Butz (at www.illigal.org), and by Pier Luca Lanzi (at xcslib.sf.net). Among the implementations of Pittsburgh classifier systems, the Samuel system is available from Alan C. Schultz at http://www.nrl.navy.mil/; Xavier Llorà distributes GALE (Genetic and Artificial Life Environment) a fine-grained parallel genetic algorithm for data mining at www.illigal.org/xllora.

Cross References

►Credit Assignment
►Genetic Algorithms
►Reinforcement Learning
►Rule Learning

Recommended Reading

Arthur, B. W., Holland, J. H., LeBaron, B., Palmer, R., & Talyer, P. (1996). *Asset pricing under endogenous expectations in an artificial stock market*. Technical Report, Santa Fe Institute.

Bacardit i Peñarroya, J. (2004). *Pittsburgh genetic-based machine learning in the data mining era: Representations, generalization, and run-time*. PhD thesis, Computer Science Department, Enginyeria i Arquitectura La Salle Universitat Ramon Llull, Barcelona.

Barry, A. M., Holmes, J., & Llora, X. (2004). Data mining using learning classifier systems. In L. Bull (Ed.), *Applications of learning classifier systems, studies in fuzziness and soft computing* (Vol. 150, pp. 15–67). Pagg: Springer.

Bassett, J. K., & de Jong, K. A. (2000). Evolving behaviors for cooperating agents. In *Proceedings of the twelfth international symposium on methodologies for intelligent systems*, LNAI (Vol. 1932). Berlin: Springer.

Booker, L. B. (1989). Triggered rule discovery in classifier systems. In J. D. Schaffer (Ed.), *Proceedings of the 3rd international conference on genetic algorithms (ICGA89)*. San Francisco: Morgan Kaufmann.

Bull, L. (Ed.). (2004). *Applications of learning classifier systems, studies in fuzziness and soft computing* (Vol. 150). Berlin: Springer, ISBN 978-3-540-21109-9.

Bull, L., & Kovacs, T. (Eds.). (2005). *Foundations of learning classifier systems, studies in fuzziness and soft computing* (Vol. 183). Berlin: Springer, ISBN 978-3-540-25073-9.

Butz, M. V. (2002). *Anticipatory learning classifier systems*. Genetic algorithms and evolutionary computation. Boston, MA: Kluwer Academic Publishers.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3(4), 261–283.

de Jong, K. (1988). Learning with genetic algorithms: An overview. *Machine Learning*, 3(2–3), 121–138.

de Jong, K. A., & Spears, W. M. (1991). Learning concept classification rules using genetic algorithms. In *Proceedings of the international joint conference on artificial intelligence* (pp. 651–656). San Francisco: Morgan Kaufmann.

Dorigo, M., & Bersini, H. (1994). A comparison of Q-learning and classifier systems. In D. Cliff, P. Husbands, J.-A. Meyer, & S. W. Wilson (Eds.), *From animals to animats 3: Proceedings of the third international conference on simulation of adaptive behavior* (pp. 248–255). Cambridge, MA: MIT Press.

Dorigo, M., & Colombetti, M. (1998). *Robot shaping: An experiment in behavior engineering*. Cambridge, MA: MIT Press/Bradford Books.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Grefenstette, J. J., Ramsey, C. L., & Schultz, A. (1990) Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5(4), 355–381.

Holland, J. (1986) Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning, an artificial intelligence approach* (Vol. II, Chap. 20) (pp. 593–623). San Francisco: Morgan Kaufmann.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press (Reprinted by the MIT Press in 1992).

Holland, J. H. (1976). Adaptation. *Progress in Theoretical Biology*, *4*, 263–293.

Holland, J. H., & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In D. A. Waterman & F. Hayes-Roth (Eds.), *Pattern-directed inference systems*. New York: Academic Press.

(Reprinted from Evolutionary computation. The fossil record. D. B. Fogel (Ed.), IEEE Press (1998)).
Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, *13*(2–3), 189–228.
Lanzi, P. L. (2001). Mining interesting knowledge from data with the XCS classifier system. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, et al. (Eds.), *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)* (pp. 958–965). San Francisco: Morgan Kaufmann.
Lanzi, P. L. (2005). Learning classifier systems: A reinforcement learning perspective. In L. Bull & T. Kovacs (Eds.), *Foundations of learning classifier systems, studies in fuzziness and soft computing* (pp. 267–284). Berlin: Springer.
Lanzi, P. L., & Perrucci, A. (1999). Extending the representation of classifier conditions part II: From messy coding to S-expressions. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, & R. E. Smith (Eds.), *Proceedings of the genetic and evolutionary computation conference (GECCO 99)* (pp. 345–352). Orlando, FL: Morgan Kaufmann.
Lanzi, P. L., & Riolo, R. L. (2003). Recent trends in learning classifier systems research. In A. Ghosh & S. Tsutsui (Eds.), *Advances in evolutionary computing: Theory and applications* (pp. 955–988). Berlin: Springer.
Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.). (2000). *Learning classifier systems: From foundations to applications*. *Lecture notes in computer science* (Vol. 1813). Berlin: Springer.
Llorá, X. (2002). *Genetics-based machine learning using fine-grained parallelism for data mining*. PhD thesis, Enginyeria i Arquitectura La Salle, Ramon Llull University, Barcelona.
Mellor, D. (2005). A first order logic classifier system. In H. Beyer (Ed.), *Proceedings of the 2005 conference on genetic and evolutionary computation (GECCO '05)*, (pp. 1819–1826). New York: ACM Press.
Quinlan, J. R., & Cameron-Jones, R. M. (1995). Induction of logic programs: FOIL and related systems. *New Generation Computing*, *13*(3&4), 287–312.
Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought*. New York: McGraw-Hill.
Smith, R. E., Dike, B. A., Niehra, R. K., Ravichandran, B., & El-Fallah, A. (2000). Classifier systems in combat: Two-sided learning of maneuvers for advanced fighter aircraft. *Computer Methods in Applied Mechanics and Engineering*, *186*(2–4), 421–437.
Smith, S. F. (1980) *A learning system based on genetic adaptive algorithms*. Doctoral dissertation, Department of Computer Science, University of Pittsburgh.
Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the eighth international joint conference on artificial intelligence* (pp. 421–425). Los Altos, CA: Morgan Kaufmann.
Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.
Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.
Tackett, W. A. (1994). *Recombination, selection, and the genetic construction of computer programs*. Unpublished doctoral dissertation, University of Southern California.
Watkins, C. (1989). *Learning from delayed rewards*. PhD thesis, King's College.
Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, *3*(2), 149–175.
Wilson, S. W. (2002). Classifiers that approximate functions. *Natural Computing*, *1*(2–3), 211–234.
Wilson, S. W. (2007). "Three architectures for continuous action" learning classifier systems. International workshops, IWLCS 2003–2005, revised selected papers. In T. Kovacs, X. Llorà, K. Takadama, P. L. Lanzi, W. Stolzmann, & S. W. Wilson (Eds.), *Lecture notes in artificial intelligence 4399* Vol. (pp. 239–257). Berlin: Springer.

Clause

A *clause* is a logical rule in a ▶logic program. Formally, a clause is a disjunction of (possibly negated) literals, such as

$$grandfather(x, y) \vee \neg father(x, z) \vee \neg parent(z, y).$$

In the logic programming language ▶Prolog this clause is written as

```
grandfather(X,Y) :- father(X,Z),
                    parent(Z,Y).
```

The part to the left of :– ("if") is the *head* of the clause, and the right part is its *body*. Informally, the clause asserts the truth of the head given the truth of the body. A clause with exactly one literal in the head is called a *Horn clause* or *definite clause*; logic programs mostly consist of definite clauses. A clause without a body is also called a *fact*; a clause without a head is also called a *denial*, or a *query* in a proof by refutation. The clause without head or body is called the *empty clause*: it signifies inconsistency or falsehood and is denoted □. Given a set of clauses, the *resolution* inference rule can be used to deduce logical consequences and answer queries (see ▶First-Order Logic).

In machine learning, clauses can be used to express classification rules for structured individuals. For example, the following definite clause classifies a molecular compound as carcinogenic if it contains a hydrogen atom with charge above a certain threshold.

```
carcinogenic(M) :- atom(M,A1),
                   element(A1,h),
                   charge(A1,C1),
                   geq(C1,0.168).
```

Cross References

►First-Order Logic
►Inductive Logic Programming
►Learning from Structured Data
►Logic Program
►Prolog

Clause Learning

In ►speedup learning, clause learning is a ►deductive learning technique used for the purpose of ►intelligent backtracking in satisfiability solvers. The approach analyzes failures at backtracking points and derives clauses that must be satisfied by the solution. The clauses are added to the set of clauses from the original satisfiability problem and serve to prune new search nodes that violate them.

Click-Through Rate (CTR)

CTR measures the success of a ranking of search results, or advertisement placing. Given the number of *impressions*, the number of times a web result or ad has been displayed, and the number of *clicks*, the number of users who clicked on the result/advertisement, CTR is the number of clicks divided by the number of impressions.

Clonal Selection

The clonal selection theory (CST) is the theory used to explain the basic response of the adaptive immune system to an antigenic stimulus. It establishes the idea that only those cells capable of recognizing an antigenic stimulus will proliferate, thus being selected against those that do not. Clonal selection operates on both T-cells and B-cells. When antibodies on a B-cell bind with an antigen, the B-cell becomes activated and begins to proliferate. New B-cell clones are produced that are an exact copy of the parent B-cell, but then they undergo somatic hypermutation and produce antibodies that are specific to the invading antigen. The B-cells, in addition to proliferating or differentiating into *plasma cells*, can differentiate into long-lived B *memory cells*. Plasma cells produce large amounts of *antibody* which will attach themselves to the antigen and act as a type of *tag* for T-cells to pick up on and remove from the system. This whole process is known as *affinity maturation*. This process forms the basis of many artificial immune system algorithms such as AIRS and aiNET.

Closest Point

►Nearest Neighbor

Cluster Editing

The Cluster Editing problem is almost equivalent to Correlation Clustering on complete instances. The idea is to obtain a graph that consists only of cliques. Although Cluster Deletion requires us to delete the smallest number of edges to obtain such a graph, in Cluster Editing we are permitted to add as well as remove edges. The final variant is Cluster Completion in which edges can only be added: each of these problems can be restricted to building a specified number of cliques.

Cluster Ensembles

Cluster ensembles are an unsupervised ►ensemble learning method. The principle is to create multiple different clusterings of a dataset, possibly using different algorithms, then aggregate the opinions of the different clusterings into an ensemble result. The final ensemble clustering should be in theory more reliable than the individual clusterings.

Cluster Optimization

►Evolutionary Clustering

Clustering

Clustering is a type of ►unsupervised learning in which the goal is to partition a set of ►examples into groups called clusters. Intuitively, the examples within a cluster are more similar to each other than to examples from other clusters. In order to measure the similarity between examples, clustering algorithms use various distortion or ►distance measures. There are two major types clustering approaches: generative and discriminative. The former assumes a parametric form of the data and tries to find the model parameters that maximize the probability that the data was generated by the chosen model. The latter represents graph-theoretic approaches that compute a similarity matrix defined over the input data.

Cross References

►Categorical Data Clustering
►Cluster Editing
►Cluster Ensembles
►Clustering from Data Streams
►Constrained Clustering
►Consensus Clustering
►Correlation Clustering
►Cross-Language Document Clustering
►Density-Based Clustering
►Dirichlet Process
►Document Clustering
►Evolutionary Clustering
►Graph Clustering
►*k*-Means Clustering
►*k*-Mediods Clustering
►Model-Based Clustering
►Partitional Clustering
►Projective Clustering
►Sublinear Clustering

Clustering Aggregation

►Consensus Clustering

Clustering Ensembles

►Consensus Clustering

Clustering from Data Streams

João Gama
University of Porto, Porto, Portugal

Definition

►*Clustering* is the process of grouping objects into different groups, such that the common properties of data in each subset is high, and between different subsets is low. The data stream clustering problem is defined as *to maintain a consistent good clustering of the sequence observed so far, using a small amount of memory and time*. The issues are imposed by the continuous arriving data points, and the need to analyze them in real time. These characteristics require incremental clustering, maintaining cluster structures that evolve over time. Moreover, the data stream may evolve over time and new clusters might appear, others disappear reflecting the dynamics of the stream.

Main Techniques

Major clustering approaches in data stream cluster analysis include:

- *Partitioning* algorithms: construct a partition of a set of objects into k clusters, that minimize some objective function (e.g., the sum of squares distances to the centroid representative). Examples include k-means (Farnstrom, Lewis, & Elkan, 2000), and k-medoids (Guha, Meyerson, Mishra, Motwani, & O'Callaghan, 2003)
- *Microclustering* algorithms: divide the clustering process into two phases, where the first phase is online and summarizes the data stream in local models (microclusters) and the second phase generates a global cluster model from the microclusters. Examples of these algorithms include BIRCH (Zhang, Ramakrishnan, & Livny, 1996) and CluStream (Aggarwal, Han, Wang, & Yu, 2003)

Basic Concepts

A powerful idea in clustering from data streams is the concept of *cluster feature*, *CF*. A cluster feature, or *microcluster*, is a compact representation of a set of points. A CF structure is a triple (N, LS, SS), used to store the sufficient statistics of a set of points:

- N is the number of data points
- LS is a vector, of the same dimension of data points, that store the linear sum of the N points
- SS is a vector, of the same dimension of data points, that store the square sum of the N points

The properties of cluster features are:

- *Incrementality*
  If a point x is added to the cluster, the sufficient statistics are updated as follows:

$$LS_A \leftarrow LS_A + x,$$

$$SS_A \leftarrow SS_A + x^2,$$

$$N_A \leftarrow N_A + 1.$$

- *Additivity*
  If A_1 and A_2 are disjoint sets, merging them is equal to the sum of their parts. The additive property allows us to merge subclusters incrementally.

$$LS_C \leftarrow LS_A + LS_B,$$
$$SS_C \leftarrow SS_A + SS_B,$$
$$N_C \leftarrow N_A + N_B.$$

A CF entry has sufficient information to calculate the norms

$$L_1 = \sum_{i=1}^{n} |x_{a_i} - x_{b_i}|,$$
$$L_2 = \sqrt{\sum_{i=1}^{n} (x_{a_i} - x_{b_i})^2}$$

and basic measures to characterize a cluster.

- *Centroid*, defined as the gravity center of the cluster:

$$\vec{X}0 = \frac{LS}{N}.$$

- *Radius*, defined as the average distance from member points to the centroid:

$$R = \sqrt{\frac{\sum_1^N (\vec{x}_i - \vec{X}0)^2}{N}}.$$

Partitioning Clustering

k-means is the most widely used clustering algorithm. It constructs a partition of a set of objects into k clusters that minimize some objective function, usually a squared error function, which imply round-shape clusters. The input parameter k is fixed and must be given in advance that limits its real applicability to streaming and evolving data.

Farnstrom et al. (2000) proposed a *single pass k-means* algorithm. The main idea is to use a buffer where points of the dataset are kept compressed. The data stream is processed in blocks. All available space on the buffer is filled with points from the stream. Using these points, find k centers such that the sum of distances from data points to their closest center is minimized. Only the k centroids (representing the clustering results) are retained, with the corresponding k cluster features. In the following iterations, the buffer is initialized with the k-centroids, found in previous iteration, weighted by the k cluster features, and incoming data points from the stream. The *Very Fast k-means* (VFKM) algorithm (Domingos & Hulten, 2001) uses the Hoeffding bound to determine the number of examples needed in each step of a k-means algorithm. VFKM runs as a sequence of k-means runs, with increasing number of examples until the Hoeffding bound is satisfied.

Guha et al. (2003) present an analytical study on k-median clustering data streams. The proposed algorithm makes a single pass over the data stream and uses small space. It requires $O(nk)$ time and $O(n\epsilon)$ space where k is the number of centers, n is the number of points, and $\epsilon < 1$. They have proved that any k-median algorithm that achieves a constant factor approximation cannot achieve a better run time than $O(nk)$.

Micro Clustering

The idea of dividing the clustering process into two layers, where the first layer generates local models (microclusters) and the second layer generates global models from the local ones, is a powerful idea that has been used elsewhere.

The BIRCH system (Zhang et al., 1996) builds a hierarchical structure of data, the CF-tree, where each node contains a set of cluster features. These CF's contain the sufficient statistics describing a set of points in the data set, and all information of the cluster features below in

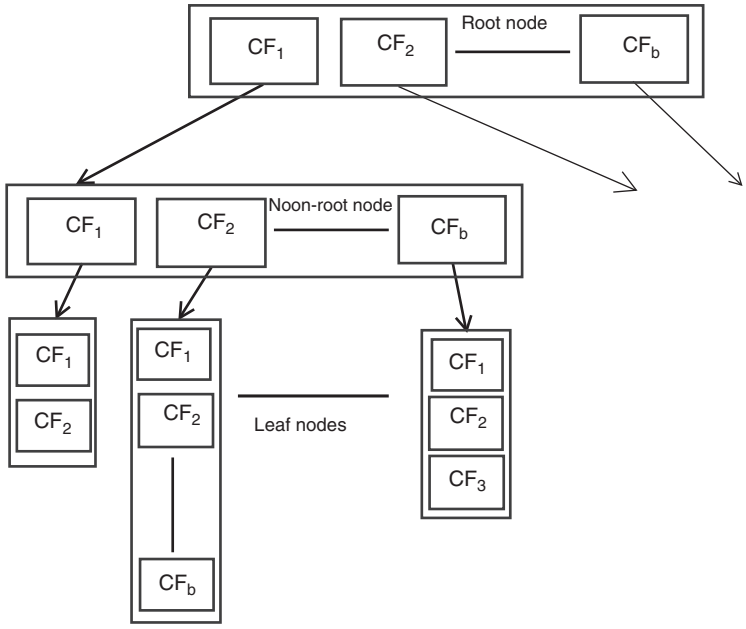
the tree. The system requires two user defined parameters: B the branch factor or the maximum number of entries in each non-leaf node; and T the maximum diameter (or radius) of any CF in a leaf node. The maximum diameter T defines the examples that can be *absorbed* by a CF. Increasing T, more examples can be absorbed by a micro-cluster and smaller CF-Trees are generated (Fig. 1).

When an example is available, it traverses down the current tree from the root it finds the appropriate leaf. At each non-leaf node, the example follow the *closest*-CF path, with respect to norms L_1 or L_2. If the closest-CF in the leaf cannot absorb the example, make a new CF entry. If there is no room for new leaf, split the parent node. A leaf node might be expanded due to the constraints imposed by B and T. The process consists of taking the two farthest CFs and creates two new leaf nodes. When traversing backup the CFs are updated.

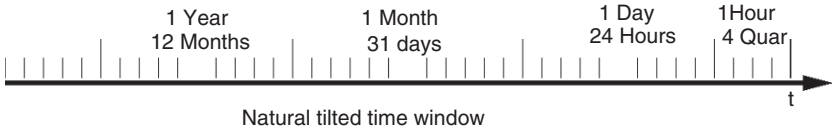
Monitoring the Evolution of the Cluster Structure

The *CluStream* Algorithm (Aggarwal et al., 2003) is an extension of the BIRCH system designed for data streams. Here, the CFs include temporal information: the time-stamp of an example is treated as a feature. CFs are initialized offline, using a standard k-means, with a large value for k. For each incoming data point, the distance to the centroids of existing CFs are computed. The data point is absorbed by an existing CF if the distance to the centroid falls within the *maximum boundary* of the CF. The *maximum boundary* is defined as a factor t of the *radius* deviation of the CF; otherwise, the data point starts a new micro-cluster.

CluStream can generate approximate clusters for any user defined time granularity. This is achieved by storing the CFT at regular time intervals, referred to as snapshots. Suppose the user wants to find clusters in the stream based on a history of length h, the off-line



Clustering from Data Streams. Figure 1. The clustering feature tree in BIRCH. B is the maximum number of CFs in a level of the tree



Clustering from Data Streams. Figure 2. The figure presents a *natural tilted time window*. The most recent data is stored with high-detail, older data is stored in a compressed way. The degree of detail decreases with time

component can analyze the snapshots stored at the snapshots t, the current time, and $(t - h)$ by using the addictive property of CFT. An important problem is when to store the snapshots of the current set of micro-clusters. For example, the natural time frame (Fig. 2) stores snapshots each quarter, four quarters are aggregated in hours, 24 h are aggregated in days, etc. The aggregation level is domain-dependent and explores the addictive property of CFT.

Tracking the Evolution of the Cluster Structure

Promising research lines are tracking change in clusters. Spiliopoulou, Ntoutsi, Theodoridis, and Schult (2006) present system MONIC, for detecting and tracking change in clusters. MONIC assumes that a cluster is an object in a geometric space. It encompasses changes that involve more than one cluster, allowing for insights on cluster change in the whole clustering. The transition tracking mechanism is based on the degree of overlapping between the two clusters. The concept of *overlap* between two clusters, X and Y, is defined as the normed number of common records weighted with the age of the records. Assume that cluster X was obtained at time t_1 and cluster Y at time t_2. The degree of overlapping between the two clusters is given by: $overlap(X, Y) = \sum_{a \in X \cap Y} age(a, t_2) / \sum_{x \in X} age(x, t_2)$. The degree of overlapping allows inferring properties of the underlying data stream. Cluster transition at a given time point is a change in a cluster discovered at an earlier timepoint. MONIC considers transitions as Internal and external transitions, that reflect the dynamics of the stream. Examples of cluster transitions include: the cluster survives, the cluster is absorbed; a cluster disappears; a new cluster emerges (Fig. 2).

Recommended Reading

Aggarwal, C., Han, J., Wang, J., & Yu, P. (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on very large data bases* (pp. 81–92). San Mateo, MA: Morgan Kaufmann.

Domingos, P., & Hulten, G. (2001). A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of international conference on machine learning* (pp. 106–113). San Mateo, MA: Morgan Kaufmann.

Farnstrom, F., Lewis, J., & Elkan, C. (2000). Scalability for clustering algorithms revisited. *SIGKDD Explorations, 2*(1), 51–57.

Guha, S., Meyerson, A., Mishra, N., Motwani, R., & O'Callaghan, L. (2003). Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering, 15*(3), 515–528.

Spiliopoulou, M., Ntoutsi, I., Theodoridis, Y., & Schult, R. (2006). Monic: Modeling and monitoring cluster transitions. In *Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 706–711). New York: ACM Press.

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). Birch: An efficient data clustering method for very large databases. In *Proceedings of ACM SIGMOD international conference on management of data* (pp. 103–114). New York: ACM Press.

Clustering of Nonnumerical Data

►Categorical Data Clustering

Clustering with Advice

►Correlation Clustering

Clustering with Constraints

►Correlation Clustering

Clustering with Qualitative Information

►Correlation Clustering

Clustering with Side Information

►Correlation Clustering

CN2

►Rule Learning

Co-Training

►Semi-Supervised Learning

Coevolution

►Coevolutionary Learning

Coevolutionary Computation

►Coevolutionary Learning

Coevolutionary Learning

R. Paul Wiegand
University of Central Florida, Orlando, FL, USA

Synonyms

Coevolution; Coevolutionary computation

Definition

Coevolutionary learning is a form of evolutionary learning (see ►Evolutionary Algorithms) in which the fitness evaluation is based on interactions between individuals. Since the evaluation of an individual is dependent on interactions with other evolving entities, changes in the set of entities used for evaluation can affect an individual's ranking in a population. In this sense, coevolutionary fitness is *subjective*, while fitness in traditional evolutionary learning systems typically uses an *objective* performance measure.

Motivation and Background

Ideally, coevolutionary learning systems focus on relevant areas of a search space by making adaptive changes between interacting, concurrently evolving parts. This can be particularly helpful when problem spaces are very large – infinite search spaces in particular. Additionally, coevolution is useful when applied to problems when no intrinsic objective measure exists. The interactive nature of evaluation makes them natural methods to consider for problems such as the search for game-playing strategies (Fogel, 2001). Finally, some coevolutionary systems appear natural for search spaces which contain certain kinds of complex structures (Potter, 1997; Stanley, 2004), since search on smaller components in a larger structure can be emphasized. In fact, there is reason to believe that coevolutionary systems may be well suited for uncovering complex structures within a problem (Bucci & Pollack, 2002).

Still, the dynamics of coevolutionary learning can be quite complex, and a number of pathologies often plague naïve users. Indeed, because of the subjective nature of coevolution, it can be easy to apply a particular coevolutionary learning system without a clear understanding of what kind of solution one expects a coevolutionary algorithm to produce. Recent theoretical analysis suggests that a clear concept of solution and a careful implementation of an evaluation process consistent with this concept can produce a coevolutionary system capable of addressing many problems (de Jong & Pollack, 2004; Ficici, 2004; Panait, 2006; Wiegand, 2003). Accordingly, a great deal of research in this area focuses on evaluation and progress measurement.

Structure of Learning System

Coevolutionary learning systems work in much the same way that an evolutionary learning system works: individuals encode some aspect of potential solutions to a problem, those representatives are altered during search using genetic-like operators such as mutation and crossover, and the search is directed by selecting better individuals as determined by some kind of fitness assessment. These heuristic methods gradually refine solutions by repeatedly cycling through such steps, using the ideas of heredity and survival of the fittest to produce new generations of individuals, with increased quality of solution. Just as in traditional evolutionary computation, there are many choices available to the engineer in designing such systems. The reader is referred to the chapters relating to evolutionary learning for more details.

However, there are some fundamental differences between traditional evolution and coevolution. In coevolution, measuring fitness requires evaluating the interaction between multiple individuals. Interacting individuals may reside in the same population or in different populations; the interactive nature of coevolution evokes notions of cooperation and competition in entirely new ways; the choices regarding how to best conduct evaluation of these interactions for the purposes of selection are particularly important; and there are unique coevolutionary issues surrounding representation. In addition, because of its interactive nature, the dynamics of coevolution can lead to some well-known pathological behaviors, and particularly careful attention to implementation choices to avoid such conditions is generally necessary.

Multiple Versus Single Population Approaches

Coevolution can typically be broadly classified as to whether interacting individuals reside in different populations or in the same population.

In the case of multipopulation coevolution, measuring fitness requires evaluating how individuals in one population interact with individuals in another. For example, individuals in each population may represent potential strategies for particular players of a game, they may represent roles in a larger ecosystem (e.g., predators and prey), or they may represent components that are fitted into a composite assembly with other component then applied to a problem. Though individuals in different populations interact for the purposes of evaluation, they are typically otherwise independent of one another in the coevolutionary search process.

In single population coevolution, an individual in the population is evaluated based on his or her interaction with other individuals in the same population. Such individuals may again represent potential strategies in a game, but evaluation may require them to trade off roles as to which player they represent in that game. Here, individuals interact not only for evaluation, but also implicitly compete with one another as resources used in the coevolutionary search process itself.

There is some controversy in the field as to whether this latter type qualifies as "coevolution." Evolutionary biologists often define coevolution exclusively in terms of multiple populations; however, in biological systems, fitness is always subjective, while the vast majority of computational approaches to evolutionary learning involve objective fitness assessment – and this subjective/objective fitness distinction creates a useful classification.

To be sure, there are fundamental differences between how single population and multipopulation learning systems behave (Ficici, 2004). Still, single population systems that employ subjective fitness assessment behave a lot more like multipopulation coevolutionary systems than like objective fitness based evolution. Moreover, historically, the field has used the term coevolution whenever fitness assessment is based on interactions between individuals, and a large amount of that research has involved systems with only one population.

Competition and Cooperation

The terms *cooperative* and *competitive* have been used to describe aspects of coevolution learning in at least three ways.

First and less commonly, these adjectives can describe qualitatively observed behaviors of potential solutions in coevolutionary systems, the results of some evolutionary process (e.g., "tit-for-tat" strategies, Axelrod, 1984).

Second, problems are sometimes considered to be inherently competitive or cooperative. Indeed, game theory provides some guidance for making such distinctions. However, since in many kinds of problems little may be known about the actual structure of the payoff functions involved, we may not actually be able to classify the problem as definitively competitive or cooperative.

The final and by far most common use of the term is to distinguish algorithms themselves. Cooperative algorithms are those in which interacting individuals succeed or fail together, while competitive algorithms are those in which individuals succeed at the expense of other individuals.

Because of the ambiguity of the terms, some researchers advocate abandoning them altogether, instead focusing distinguishing terminology on the form a potential solution takes. For example, using the term ▶compositional coevolution to describe an algorithm designed to return a solution composed of multiple individuals (e.g., a multiagent team) and using the term ▶test-based coevolution to describe an algorithm designed to return an individual who performs well against an adaptive set of tests (e.g., sorting network). This latter pair of terms is a slightly different, though probably more useful distinction than the cooperative and competitive terms.

Still, it is instructive to survey the algorithms based on how they have been historically classified.

Examples of competitive coevolutionary learning include simultaneously learning sorting networks and challenging data sets in a predator–prey type relationship (Hillis, 1991). Here, individuals in one population representing potential sorting networks are awarded a fitness score based on how well they sort opponent data sets from the other population. Individuals in the second population represent potential data sets whose fitness is based on how well they distinguish opponent sorting networks.

Competitive coevolution has also been applied to learning game-playing strategies (Fogel, 2001; Rosin & Belew, 1996). Additionally, competition has played a vital part in the attempts to coevolve complex agent

behaviors (Sims, 1994). Finally, competitive approaches have been applied to a variety of more traditional machine learning problems, for example, learning classifiers in one population and challenging subsets of exemplars in the other (Paredis, 1994).

Potter developed a relatively general framework for cooperative coevolutionary learning, applying it first to static function optimization and later to neural network learning (Potter, 1997). Here, each population contains individuals representing a portion of the network, and evolution of these components occurs almost independently, in tandem with one another, interacting only to be assembled into a complete network in order to obtain fitness. The decomposition of the network can be static and a priori, or dynamic in the sense that components may be added or removed during the learning process.

Moriarty et al. take a different, somewhat more adaptive approach to cooperative coevolution of neural networks (Moriarty & Miikkulainen, 1997). In this case, one population represents potential network *plans*, while a second is used to acquire node information. Plans are evaluated based on how well they solve a problem with their collaborating nodes, and the nodes receive a share of this fitness. Thus, a node is rewarded for participating more with successful plans, and thus receives fitness only indirectly.

Evaluation

Choices surrounding how interacting individuals in coevolutionary systems are evaluated for the purposes of selection are perhaps the most important choices facing an engineer employing these methods. Designing the evaluation method involves a variety of practical choices, as well as a broader eye to the ultimate purpose of the algorithm itself.

Practical concerns in evaluation include determining the number of individuals with whom to interact, how those individuals will be chosen for the interaction, and how the selection will operate on the results of multiple interactions (Wiegand, 2003). For example, one might determine the fitness of an individual by pairing him or her with all other individuals in the other populations (or the same population for single population approaches) and taking the average or maximum value of such evaluations as the fitness assessment. Alternatively, one may simply use the single best individual as determined by a previous generation of the algorithm, or a combination of those approaches. Random pairings between individuals is also common. This idea can be extended to use tournament evaluation where successful individuals from pairwise interactions are promoted and further paired, assigning fitness based on how far an individual progresses in the tournament. Many of these methods have been evaluated empirically on a variety of types of problems (Angeline & Pollack, 1993; Bull, 1997; Wiegand, 2003).

However, the designing of the evaluation method also speaks to the broader issue of how to best implement the desired ▶solution concept, (a criterion specifying which locations in the search space are solutions and which are not) (Ficici, 2004). The key to successful application of coevolutionary learning is to first elicit a clear and precise solution concept and then design an algorithm (an evaluation method in particular) that implements such a concept explicitly.

A successful coevolutionary learner capable of achieving reliable progress toward a particular solution concept often makes use of an archive of individuals and an update rule for that archive that insists the distance to a particular solution concept decrease with every change to the archive. For example, if one is interested in finding game strategies that satisfy Nash equilibrium constraints, one might consider comparing new individuals to an archive of potential individual strategies found so far that together represent a potential Nash mixed strategy (Ficici, 2004). Alternatively, if one is interested in maximizing the sum of an individual's outcomes over all tests, one may likewise employ an archive of discovered tests that candidate solutions are able to solve (de Jong, 2004).

It is useful to note that many coevolutionary learning problems are multiobjective in nature. That is, ▶underlying objectives may exist in such problems, each creating a different ranking for individuals depending on the set of tests being considered during evaluation (Bucci & Pollack, 2002). The set of all possible underlying objectives (were it known) is sufficient to determine the outcomes on all possible tests. A careful understanding of this can yield approaches that create

ideal and minimal evaluation sets for such problems (de Jong & Pollack, 2004).

By acknowledging the link between multiobjective optimization and coevolutionary learning, a variety of evaluation and selection methods based on notions of multiobjective optimization have been employed. For example, there are selection methods that use Pareto dominance between candidate solutions and their tests as their basis of comparison (Ficici, 2004). Additionally, such methods can be combined with archive-based approaches to ensure monotonicity of progress toward a Pareto dominance solution concept (de Jong & Pollack, 2004).

Representation

Perhaps the core representational question in coevolution is the role that an individual plays. In test-based coevolution, an individual typically represents a potential solution to the problem or a test for a potential solution, whereas in compositional coevolution individuals typically represent a candidate component for a composite or ensemble solution.

Even in test-based approaches, the true solution to the problem may be expressed as a population of individuals, rather than a single individual. The population may represent a mixed strategy while individuals represent potential pure strategies for a game. Engineers using such approaches should be clear of the form of the final solution produced by the algorithm, and that this form is consistent with the prescribed solution concept.

In compositional approaches, the key issues tend to surround about how the problem is decomposed. In some algorithms, this decomposition is performed a priori, having different populations represent explicit components of the problem (Potter, 1997). In other approaches, the decomposition is intended to be somewhat more dynamic (Moriarty & Miikkulainen, 1997; Potter, 1997). Still more recent approaches seek to harness the potential of compositional coevolutionary systems to search open-ended representational spaces by gradually *complexifying* the representational space during the search (Stanley, 2004).

In addition, a variety of coevolutionary systems have successfully dealt with some inherent pathologies by representing populations in spatial topologies, and restricting selection and interaction using geometric constraints defined by those topologies (Pagie, 1999). Typically, these systems involve overlaying multiple grids of individuals, applying selection within some neighborhood in a given grid, and evaluating interactions between individuals in different grids using a similar type of cross-population neighborhood. The benefits of these systems are in part due to their ability to naturally regulate loss of diversity and spread of interaction information by explicit control over the size and shape of these neighborhoods.

Pathologies and Remedies

Perhaps the most commonly cited pathology is the so-called *loss of gradient* problem, in which one population comes to severely dominate the others, thus creating a situation in which individuals cannot be distinguished from one another. The populations become disengaged and evolutionary progress may *stall* or *drift* (Watson & Pollack, 2001). Disengagement most commonly occurs when distinguishing individuals are lost in the evolutionary process (*forgetting*), and the solution to this problem typically involves somehow retaining potentially informative, though possibly inferior quality individuals (e.g., archives).

Intransitivities in the reward system can cause some coevolutionary systems to exhibit *cycling* dynamics (Watson & Pollack, 2001), where reciprocal changes force the system to orbit some part of a potential search space. The remedy to this problem often involves creating coevolutionary systems that change in response to traits in several other populations. Mechanisms introduced to produce such effects include *competitive fitness sharing* (Rosin & Belew, 1996).

Another challenging problem occurs when individuals in a coevolutionary systems *overspecialize* on one underlying objective at the expense of other necessary objectives (Watson & Pollack, 2001). In fact, overspecialization can be seen as a form of disengagement on some subset of underlying objectives, and likewise the repair to this problem often involves retaining individuals capable of making distinctions in as many underlying objectives as possible (Bucci & Pollack, 2003).

Finally, certain kinds of compositional coevolutionary learning algorithms can be prone to *relative overgeneralization*, a pathology in which components that perform reasonably well in a variety of composite solutions are favored over those that are part of an optimal solution (Wiegand, 2003). In this case, it is typically possible to bias the evaluation process toward optimal values by evaluating an individual in a variety of composite assemblies and assigning the best objective value found as the fitness (Panait, 2006).

In addition to pathological behaviors in coevolution, the subjective nature of these learning systems creates difficulty in measuring progress. Since fitness is subjective, it is impossible to determine whether these relative measures indicate progress or stagnation when the measurement values do not change much. Without engaging some kind of external or objective measure, it is difficult to understand what the system is really doing. Obviously, if an objective measure exists then it can be employed directly to measure progress (Watson & Pollack, 2001).

A variety of measurement methodologies have been employed when objective measurement is not possible. One method is to compare current individuals against all ancestral opponents (Cliff & Miller, 1995). Another predator/prey based method holds *master tournaments* between all the best predators and all the best prey found during the search (Nolfi & Floreano, 1998). A similar approach suggests maintaining the best individuals from each generation in each population in a *hall of fame* for comparison purposes (Rosin & Belew, 1996). Still other approaches seek to record the points during the coevolutionary search in which a new dominant individual was found (Stanley, 2004). A more recent approach advises looking at the *population differential*, examining all the information from ancestral generations rather than simply selecting a biased subset (Bader-Natal & Pollack, 2005). Conversely, an alternative idea is to consider how well the *dynamics* of the best individuals in different populations reflect the fundamental *best response* curves defined by the problem (Popovici, 2006).

With a clear solution concept, an appropriate evaluation mechanism implementing that concept, and practical progress measures in place, coevolution can be an effective and versatile machine learning tool.

Cross References

►Evolutionary Algorithms

Recommended Reading

Angeline, P., & Pollack, J. (1993). Competitive environments evolve better solutions for complex tasks. In S. Forest (Ed.), *Proceedings of the fifth international conference on genetic algorithms* (pp. 264–270). San Mateo, CA: Morgan Kaufmann.

Axelrod, R. (1984). *The evolution of cooperation*. New York: Basic Books.

Bader-Natal, A., & Pollack, J. (2005). Towards metrics and visualizations sensitive to Coevolutionary failures. In *AAAI technical report FS-05-03 coevolutionary and coadaptive systems*. AAAI Fall Symposium, Washington, DC.

Bucci, A., & Pollack, J. B. (2002). A mathematical framework for the study of coevolution. In R. Poli, et al. (Eds.), *Foundations of genetic algorithms VII* (pp. 221–235). San Francisco: Morgan Kaufmann.

Bucci, A., & Pollack, J. B. (2003). Focusing versus intransitivity geometrical aspects of coevolution. In E. Cantú-Paz, et al. (Eds.), *Proceedings of the 2003 genetic and evolutionary computation conference* (pp. 250–261). Berlin: Springer.

Bull, L. (1997). Evolutionary computing in multi-agent environments: Partners. In T. Bäck (Ed.), *Proceedings of the seventh international conference on genetic algorithms* (pp. 370–377). San Mateo, CA: Morgan Kaufmann.

Cliff, D., & Miller, G. F. (1995). Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *Proceedings of the third European conference on artificial life* (pp. 200–218). Berlin: Springer.

de Jong, E. (2004). The maxsolve algorithm for coevolution. In H. Beyer, et al. (Eds.), *Proceedings of the 2005 genetic and evolutionary computation conference* (pp. 483–489). New York, NY: ACM Press.

de Jong, E., & Pollack, J. (2004). Ideal evaluation from coevolution. *Evolutionary Computation, 12*, 159–192.

Ficici, S. G. (2004). *Solution concepts in coevolutionary algorithms*. PhD thesis, Brandeis University, Boston, MA.

Fogel, D. (2001). *Blondie24: Playing at the edge of artificial intelligence*. San Francisco: Morgan Kaufmann.

Hillis, D. (1991). Co-evolving parasites improve simulated evolution as an optimization procedure. *Artificial life II, SFI studies in the sciences of complexity* (Vol. 10, pp. 313–324).

Moriarty, D., & Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation, 5*, 373–399.

Nolfi, S., & Floreano, D. (1998). Co-evolving predator and prey robots: Do "arm races" arise in artificial evolution? *Artificial Life, 4*, 311–335.

Pagie, L. (1999). Information integration in evolutionary processes. PhD thesis, Universiteit Utrecht, the Netherlands.

Panait, L. (2006). *The analysis and design of concurrent learning algorithms for cooperative multiagent systems*. PhD thesis, George Mason University, Fairfax, VA.

Paredis, J. (1994). Steps towards co-evolutionary classification networks. In R. A. Brooks & P. Maes (Eds.), *Artificial life IV,*

proceedings of the fourth international workshop on the synthesis and simulation of living systems (pp. 359–365). Cambridge, MA: MIT Press.

Popovici, E. (2006). *An analysis of multi-population co-evolution*. PhD thesis, George Mason University, Fairfax, VA.

Potter, M. (1997). *The design and analysis of a computational model of cooperative co-evolution*. PhD thesis, George Mason University, Fairfax, VA.

Rosin, C., & Belew, R. (1996). New methods for competitive coevolution. *Evolutionary Computation, 5*, 1–29.

Sims, K. (1994). Evolving 3D morphology and behavior by competition. In R. A. Brooks & P. Maes (Eds.), *Artificial life IV, proceedings of the fourth international workshop on the synthesis and simulation of living systems* (pp. 28–39). Cambridge, MA: MIT Press.

Stanley, K. (2004). *Efficient evolution of neural networks through complexification*. PhD thesis, The University of Texas at Austin, Austin, TX.

Watson, R., & Pollack, J. (2001). Coevolutionary dynamics in a minimal substrate. In L. Spector, et al. (Eds.), *Proceedings from the 2001 genetic and evolutionary computation conference* (pp. 702–709). San Francisco: Morgan Kaufmann.

Wiegand, R. P. (2003). *An analysis of cooperative coevolutionary algorithms*. PhD thesis, George Mason University, Fairfax, VA.

Collaborative Filtering

Collaborative Filtering (CF) refers to a class of techniques used in that recommend items to users that other users with similar tastes have liked in the past. CF methods are commonly sub-divided into *neighborhood-based* and *model-based* approaches. In neighborhood-based approaches, a subset of users are chosen based on their similarity to the active user, and a weighted combination of their ratings is used to produce predictions for this user. In contrast, model-based approaches assume an underlying structure to users' rating behavior, and induce predictive models based on the past ratings of all users.

Collection

►Class

Collective Classification

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor
University of Maryland, MD, USA

Synonyms

Iterative classification; Link-based classification

Definition

Many real-world ►classification problems can be best described as a set of objects interconnected via links to form a network structure. The links in the network denote relationships among the instances such that the class labels of the instances are often correlated. Thus, knowledge of the correct label for one instance improves our knowledge about the correct assignments to the other instances it connects to. The goal of collective classification is to *jointly* determine the correct label assignments of all the objects in the network.

Motivation and Background

Traditionally, a major focus of machine learning is to solve classification problems: given a corpus of documents, classify each according to its topic label; given a collection of e-mails, determine which are spam; given a sentence, determine the part-of-speech tag for each word; given a hand-written document, determine the characters, etc. However, much of the work in machine learning makes an *independent and identically distributed* (IID) assumption, and focuses on predicting the class label of each instance in isolation. In many cases, however, the class labels whose values need to be determined can benefit if we know the correct assignments to related class labels. For example, it is easier to predict the topic of a webpage if we know the topics of the webpages that link to it, the chance of a particular word being a verb increases if we know that the previous word in the sentence is a noun, knowing the rest of the characters in a word can make it easier to identify an unknown character, etc. In the last decade, many researchers have proposed techniques that attempt to classify samples in a joint or collective manner instead of treating each sample in isolation, and reported significant gains in classification accuracy.

Theory/Solution

Collective classification is a combinatorial optimization problem, in which we are given a set of nodes, $\mathcal{V} = \{v_1, \ldots, v_n\}$, and a neighborhood function $\mathcal{N}$, where $\mathcal{N}_i \subseteq \mathcal{V} \backslash \{v_i\}$, which describes the underlying network structure. Each node in $\mathcal{V}$ is a random variable that can take a value from an appropriate domain, $\mathcal{L} = \{l_1, \ldots, l_q\}$. $\mathcal{V}$ is further divided into two sets of nodes: $\mathcal{X}$, the nodes for which we know the correct values (observed variables) and, $\mathcal{Y}$, the nodes whose values need to be determined. Our task is to label the nodes $y_i \in \mathcal{Y}$ with one of a small number of predefined labels in $\mathcal{L}$.

Even though it is only in the last decade that collective classification has entered the collective conscience of machine learning researchers, the general idea can be traced further back (Besag, 1986). As a result, a number of approaches have been proposed. The various approaches to collective classification differ in the kinds of information they aim to exploit to arrive at the correct classification, and their mathematical underpinnings. We discuss each in turn.

Relational Classification

Traditional classification concentrates on using the observed attributes of the instance to be classified. Relational classification (Slattery & Craven, 1998) attempts to go a step further by classifying the instance using not only the instance's own attributes but also the instance's neighbors' attributes. For example, in a hypertext classification domain where we want to classify webpages, not only would we use the webpage's own words but we would also look at the webpages linking to this webpage using hyperlinks and their words to arrive at the correct class label. Results obtained using relational classification have been mixed. For example, even though there have been reports of classification accuracy gains using such techniques, in certain cases, these techniques can harm classification accuracy (Chakrabarti, Dom, & Indyk, 1998).

Iterative Collective Classification with Neighborhood Labels

A second approach to collective classification is to use the class labels assigned to the neighbor instead of using the neighbor's observed attributes. For example, going back to our hypertext classification example, instead of using the linking webpage's words we would, in this case, use its assigned labels to classify the current webpage. Chakrabarti et al. (1998) illustrated the use of this approach and reported impressive classification accuracy gains. Neville and Jensen (2000) further developed the approach, and referred to the approach as iterative classification, and studied the conditions under which it improved classification performance (Jensen, Neville, & Gallagher, 2004). Techniques for feature construction from the neighboring labels were developed and studied (Lu & Getoor, 2003), along with methods that make use of *only* the label information (Macskassy & Provost, 2007), as well as a variety of strategies for when to commit the class labels (McDowell, Gupta, & Aha, 2007).

Algorithm 1 depicts pseudo-code for a simple version of the Iterative Classification Algorithm (ICA). The basic premise behind ICA is extremely simple. Consider a node $Y_i \in \mathcal{Y}$ whose value we need to determine and suppose we know the values of all the other nodes in its neighborhood $\mathcal{N}_i$ (note that $\mathcal{N}_i$ can contain both observed and unobserved variables). Then, ICA assumes that we are given a local classifier f that takes the values of $\mathcal{N}_i$ as arguments and returns a label value for Y_i from the class label set $\mathcal{L}$. For local classifiers f that do not return a class label but a goodness/likelihood value given a set of attribute values and a label, we

Algorithm 1 Iterative classification algorithm

Iterative Classification Algorithm (ICA)

for each node $Y_i \in \mathcal{Y}$ **do** {bootstrapping}
  {compute label using only observed nodes in $\mathcal{N}_i$}
  compute $\vec{a}_i$ using only $\mathcal{X} \cap \mathcal{N}_i$
  $y_i \leftarrow f(\vec{a}_i)$
end for
repeat {iterative classification}
  generate ordering $\mathcal{O}$ over nodes in $\mathcal{Y}$
  for each node $Y_i \in \mathcal{O}$ **do**
    {compute new estimate of y_i}
    compute $\vec{a}_i$ using current assignments to $\mathcal{N}_i$
    $y_i \leftarrow f(\vec{a}_i)$
  end for
until all class labels have stabilized or a threshold number of iterations have elapsed

simply choose the label that corresponds to the maximum goodness/likelihood value; in other words, we replace f with $\text{argmax}_{l \in \mathcal{L}} f$. This makes the local classifier f extremely flexible and we can use anything ranging from a decision tree to a ►support vector machine (SVM). Unfortunately, it is rare in practice that we know all values in $\mathcal{N}_i$, which is why we need to repeat the process iteratively, in each iteration, labeling each Y_i using the current best estimates of $\mathcal{N}_i$ and the local classifier f, and continuing to do so until the assignments to the labels stabilize.

Most local classifiers are defined as functions whose argument consists of a fixed-length vector of attribute values. A common approach to circumvent such a situation is to use an aggregation operator such as count, mode, or prop, which measures the proportion of neighbors with a given label. In Algorithm 1, we use $\vec{a}_i$ to denote the vector encoding the values in $\mathcal{N}_i$ obtained after aggregation. Note that in the first ICA iteration, all labels y_i are undefined and to initialize them we simply apply the local classifier to the observed attributes in the neighborhood of Y_i, this is referred to as "bootstrapping" in Algorithm 1.

Researchers in collective classification (Macskassy & Provost, 2007; McDowell et al., 2007; Neville & Jensen, 2000) have extended the simple algorithm described above, and developed a version of Gibbs sampling that is easy to implement and faster than traditional Gibbs sampling approaches. The basic idea behind this algorithm is to assume, just like in the case of ICA, that we have access to a local classifier f that can sample for the best label estimate for Y_i given all the values for the nodes in $\mathcal{N}_i$. We keep doing this repeatedly for a fixed number of iterations (a period known as "burn-in"). After that, not only do we sample for labels for each $Y_i \in \mathcal{Y}$ but we also maintain count statistics as to how many times we sampled label l for node Y_i. After collecting a predefined number of such samples we output the best label assignment for node Y_i by choosing the label that was assigned the maximum number of times to Y_i while collecting samples.

One of the benefits of both variants of ICA is fairly simple to make use of any local classifier. Some of the classifiers used included the following: naïve Bayes (Chakrabarti et al., 1998; Neville & Jensen, 2000), ►logistic regression (Lu & Getoor, 2003), ►decision trees, (Jensen et al., 2004) and weighted-vote relational neighbor (Macskassy & Provost, 2007). There is some evidence to indicate that discriminately trained local classifiers such as logistic regression tend to produce higher accuracies than others; this is consistent with results in other areas.

Other aspects of ICA that have been the subject of investigation include the ordering strategy to determine in which order to visit the nodes to relabel in each ICA iteration. There is some evidence to suggest that ICA is fairly robust to a number of simple ordering strategies such as random ordering, visiting nodes in ascending order of diversity of its neighborhood class labels, and labeling nodes in descending order of label confidences (Getoor, 2005). However, there is also some evidence that certain modifications to the basic ICA procedure tend to produce improved classification accuracies. For example, both (Neville & Jensen, 2000) and (McDowell et al., 2007) propose a strategy where only a subset of the unobserved variables are utilized as inputs for feature construction. More specifically, in each iteration, they choose the top-k most confident predicted labels and use only those unobserved variables in the following iteration's predictions, thus ignoring the less confident predicted labels. In each subsequent iteration they increase the value of k so that in the last iteration all nodes are used for prediction. McDowell et al. report that such a "cautious" approach leads to improved accuracies.

Collective Classification with Graphical Models

In addition to the approaches described above, which essentially focus on local representations and propagation methods, another approach to collective classification is by first representing the problem with a high-level global ►graphical model and then using learning and inference techniques for the graphical modeling approach to arrive at the correct classifications. These proposals include the use of both directed ►graphical models (Getoor, Segal, Taskar, & Koller, 2001) and undirected graphical models (Lafferty, McCallum, & Pereira, 2001; Taskar, Abbeel, & Koller, 2002). See ►*statistical relational learning* and Getoor and Taskar (2007) for a survey of various graphical models that are suitable for collective classification. In general, these techniques can use both neighborhood labels and observed attributes

of neighbors. On the other hand, due to their generality, these techniques also tend to be less efficient than the iterative collective classification techniques.

One common way of defining such a global model uses a *pairwise Markov random field* (pairwise MRF) (Taskar et al., 2002). Let $G = (\mathcal{V}, E)$ denote a graph of random variables as before where $\mathcal{V}$ consists of two types of random variables, the unobserved variables, $\mathcal{Y}$, which need to be assigned domain values from label set $\mathcal{L}$, and observed variables $\mathcal{X}$ whose values we know (see ►Graphical Models). Let Ψ denote a set of *clique potentials*. Ψ contains three distinct types of functions:

- For each $Y_i \in \mathcal{Y}$, $\psi_i \in \Psi$ is a mapping $\psi_i : \mathcal{L} \rightarrow \Re_{\geq 0}$, where $\Re_{\geq 0}$ is the set of nonnegative real numbers.
- For each $(Y_i, X_j) \in E$, $\psi_{ij} \in \Psi$ is a mapping $\psi_{ij} : \mathcal{L} \rightarrow \Re_{\geq 0}$.
- For each $(Y_i, Y_j) \in E$, $\psi_{ij} \in \Psi$ is a mapping $\psi_{ij} : \mathcal{L} \times \mathcal{L} \rightarrow \Re_{\geq 0}$.

Let $\mathbf{x}$ denote the values assigned to all the observed variables in $\mathcal{V}$ and let x_i denote the value assigned to X_i. Similarly, let $\mathbf{y}$ denote any assignment to all the unobserved variables in $\mathcal{V}$ and let y_i denote a value assigned to Y_i. For brevity of notation we will denote by ϕ_i the clique potential obtained by computing $\phi_i(y_i) = \psi_i(y_i) \prod_{(Y_i, X_j) \in E} \psi_{ij}(y_i)$. We are now in a position to define a pairwise MRF.

Definition 1 *A pairwise Markov random field (MRF) is given by a pair* $\langle G, \Psi \rangle$ *where* G *is a graph and* Ψ *is a set of clique potentials with* ϕ_i *and* ψ_{ij} *as defined above. Given an assignment* $\mathbf{y}$ *to all the unobserved variables* $\mathcal{Y}$*, the pairwise MRF is associated with the probability distribution* $P(\mathbf{y}|\mathbf{x}) = \frac{1}{\mathcal{Z}(\mathbf{x})} \prod_{Y_i \in \mathcal{Y}} \phi_i(y_i) \prod_{(Y_i, Y_j) \in E} \psi_{ij}(y_i, y_j)$ *where* $\mathbf{x}$ *denotes the observed values of* $\mathcal{X}$ *and* $\mathcal{Z}(\mathbf{x}) = \sum_{\mathbf{y}'} \prod_{Y_i \in \mathcal{Y}} \phi_i(y_i') \prod_{(Y_i, Y_j) \in E} \psi_{ij}(y_i', y_j')$.

Given a pairwise MRF, it is conceptually simple to extract the best assignments to each unobserved variable in the network. For example, we may adopt the criterion that the best label value for Y_i is simply the one corresponding to the highest marginal probability obtained by summing over all other variables from the probability distribution associated with the pairwise MRF. Computationally, however, this is difficult to achieve since computing one marginal probability requires summing over an exponentially large number of terms, which is why we need approximate inference algorithms. Hence, approximate inference algorithms are typically employed, the two most common being *loopy belief propagation* (LBP) and *mean-field relaxation labeling*.

Applications

Due to its general applicability, collective classification has been applied to a number of real-world problems. Foremost in this list is document classification. Chakrabarti et al. (1998) was one of the first to apply collective classification to corpora of patents linked via hyperlinks and reported that considering attributes of neighboring documents actually hurts classification performance. Slattery and Craven (1998) also considered the problem of document classification by constructing features from neighboring documents using an ►*inductive logic programming* rule learner. Yang, Slattery, & Ghani (2002) conducted an in-depth investigation over multiple datasets commonly used for document classification experiments and identified different patterns. Other applications of collective classification include object labeling in images (Hummel & Zucker, 1983), analysis of spatial statistics (Besag, 1986), iterative decoding (Berrou, Glavieux, & Thitimajshima, 1993), part-of-speech tagging (Lafferty et al., 2001), classification of hypertext documents using hyperlinks (Taskar et al., 2002), link prediction (Getoor, Friedman, Koller, & Taskar, 2002; Taskar, Wong, Abbeel, & Koller, 2003), optical character recognition (Taskar, Guestrin, & Koller, 2003), entity resolution in sensor networks (Chen, Wainwright, Cetin, & Willsky, 2003), predicting disulphide bonds in protein molecules (Taskar, Chatalbashev, Koller, & Guestrin, 2005), segmentation of 3D scan data (Anguelov et al., 2005), and classification of e-mail speech acts (Carvalho & Cohen, 2005). Recently, there have also been attempts to extend collective classification techniques to the semi-supervised learning scenario (Lu & Getoor, 2003b; Macskassy, 2007; Xu, Wilkinson, Southey, & Schuurmans, 2006).

Cross References

►Decision Trees
►Inductive Logic Programming
►Learning From Structured Data

►Relational Learning
►Semi-Supervised Learning
►Statistical Relational Learning

Recommended Reading

Anguelov, D., Taskar, B., Chatalbashev, V., Koller, D., Gupta. D., Heitz, G., et al. (2005). Discriminative learning of Markov random fields for segmentation of 3d scan data. In *IEEE computer society conference on computer vision and pattern recognition*. IEEE Computer Society, Washington D.C.

Berrou, C., Glavieux, A., & Thitimajshima, P. (1993). Near Shannon limit error-correcting coding and decoding: Turbo codes. In *Proceedings of IEEE international communications conference*, Geneva, Switzerland, IEEE.

Besag, J. (1986). On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society, B-48*, 259–302.

Carvalho, V., & Cohen, W. W. (2005). On the collective classification of email speech acts. In *Special interest group on information retrieval*, Salvador, Brazil, ACM.

Chakrabarti, S., Dom, B., & Indyk, P. (1998). Enhanced hypertext categorization using hyperlinks. In *International conference on management of data*, Seattle, Washington New York: ACM.

Chen, L., Wainwright, M., Cetin, M., & Willsky, A. (2003). Multitargetmultisensor data association using the tree-reweighted max-product algorithm. In *SPIE Aerosense conference*. Orlando, Florida.

Getoor, L. (2005). Link-based classification. In *Advanced methods for knowledge discovery from complex data*. New York: Springer.

Getoor, L., & Taskar, B. (Eds.). (2007). *Introduction to statistical relational learning*. Cambridge, MA: MIT Press.

Getoor, L., Segal, E., Taskar, B., & Koller, D. (2001). Probabilistic models of text and link structure fro hypertext classification. In *Proceedings of the IJCAI workshop on text learning: Beyond supervision*, Seattle, WA.

Getoor, L., Friedman, N., Koller, D., & Taskar, B. (2002). Learning probabilistic models of link structure. *Journal of Machine Learning Research, 3*, 679–707.

Hummel, R., & Zucker, S. (1983). On the foundations of relaxation labeling processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 5*, 267–287.

Jensen, D., Neville, J., & Gallagher, B. (2004). Why collective inference improves relational classification. In *Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining*, Seattle, WA. ACM.

Lafferty, J. D., McCallum, A., & Pereira, F. C. N. (2001). conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the international conference on machine learning*, Washington DC. San Francisco, CA: Morgan Kaufmann.

Lu, Q., & Getoor, L. (2003a). Link based classification. In *Proceedings of the international conference on machine learning*. AAAI Press, Washington, D.C.

Lu, Q., & Getoor, L. (2003b). Link-based classification using labeled and unlabeled data. In *ICML workshop on the continuum from labeled to unlabeled data in machine learning and data mining*. Washington, D.C.

Macskassy, S., & Provost, F. (2007). Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research, 8*, 935–983.

Macskassy, S. A. (2007). Improving learning in networked data by combining explicit and mined links. In *Proceedings of the twenty-second conference on artificial intelligence*. AAAI Press, Vancouver, Canada.

McDowell, L. K., Gupta, K. M., & Aha, D. W. (2007). Cautious inference in collective classification. In *Proceedings of AAAI*. AAAI Press, Vancouver, Canada.

Neville, J., & Jensen, D. (2007). Relational dependency networks. *Journal of Machine Learning Research, 8*, 653–692.

Neville, J., & Jensen, D. (2000). Iterative classification in relation data. In *Workshop on statistical relational learning, AAAI*.

Slattery, S., & Craven, M. (1998). Combining statistical and relational methods for learning in hypertext domains. In *International conferences on inductive logic programming*. Springer-Verlag, London, UK.

Taskar, B., Abbeel, P., & Koller, D. (2002). Discriminative probabilistic models for relational data. In *Proceedings of the annual conference on uncertainty in artificial intelligence*. Morgan Kauffman, San Francisco, CA.

Taskar, B., Guestrin, C., & Koller, D. (2003a). Max-margin markov networks. In *Neural information processing systems*. MIT Press, Cambridge, MA.

Taskar, B., Wong, M. F., Abbeel, P., & Koller, D. (2003b). Link prediction in relational data. In *Natural information processing systems*. MIT Press, Cambridge, MA.

Taskar, B., Chatalbashev, V., Koller, D., & Guestrin, C. (2005). Learning structured prediction models: A large margin approach. In *Proceedings of the international conference on machine learning*. ACM, New York, NY.

Xu, L., Wilkinson, D., Southey, F., & Schuurmans, D. (2006). Discriminative unsupervised learning of structured predictors. In *Proceedings of the international conference on machine learning*. ACM, New York, NY.

Yang, Y., Slattery, S., & Ghani, R. (2002). A study of approaches to hypertext categorization. *Journal of Intelligent Information Systems. 18*(2–3), 219–241.

Commercial Email Filtering

►Text Mining for Spam Filtering

Committee Machines

►Ensemble Learning

Community Detection

►Group Detection

Comparable Corpus

A comparable corpus (pl. corpora) is a document collection composed of two or more disjoint subsets, each written in a different language, such that documents in each subset are on a same topic as the documents in the others. The prototypical example of a comparable corpora is a collection of newspaper article written in different languages and reporting about the same events: while they will not be, strictly speaking, the translation of one another, they will share most of the semantic content. Some methods for ►cross-language text mining rely, totally or partially, on the statistical properties of comparable corpora.

Competitive Coevolution

►Test-Based Coevolution

Competitive Learning

Competitive learning is an ►artificial neural network learning process where different neurons or processing elements compete on who is allowed to learn to represent the current input. In its purest form competitive learning is in the so-called winner-take-all networks where only the neuron that best represents the input is allowed to learn. Since all neurons learn to better represent the kinds of inputs they already are good at representing, they become specialized to represent different kinds of inputs. For vector-valued inputs and representations, the input becomes quantized to the unit having the closest representation (model), and the representations are adapted to minimize the representation error using stochastic gradient descent.

Competitive learning networks have been studied as models of how receptive fields and feature detectors, such as orientation-selective visual neurons, develop in neural networks. The same process is at work in online ►K-means clustering, and variants of it in ►Self-Organizing Maps (SOM) and the EM algorithm of mixture models.

Complex Adaptive System

►Complexity in Adaptive Systems

Complexity in Adaptive Systems

Jun He
Aberystwyth University, Wales, UK

Synonyms

Adaptive system; Complex adaptive system

Definition

An ►adaptive system, or complex adaptive system, is a special case of complex systems, which is able to adapt its behavior according to changes in its environment or in parts of the system itself. In this way, the system can improve its performance through a continuing interaction with its environment. The concept of ►complexity in an adaptive system is used to analyze the interactive relationship between the system and its environment, which can be classified into two types: ►internal complexity for model complexity, and ►external complexity for data complexity. The internal complexity is defined by the amount of input, information, or energy that the system receives from its environment. The external complexity refers to the complexity of how the system represents these inputs through its internal process.

Motivation and Background

Adaptive systems range from natural systems to artificial systems (Holland, 1992, 1995; Waldrop, 1992). Examples of natural systems include ant colonies, ecosystem, the brain, neural network and immune system, cell and developing embryo; examples of artificial systems include stock market, social system, manufacturing businesses, and human social group-based

endeavor in a cultural and social system such as political parties or communities. All these systems have a common feature: they can adapt to their environment.

An adaptive system is adaptive in that way it has the capacity to change its internal structure for adapting the environment. It is complex in the sense that it is interactive with its environment. The interaction between an adaptive system and its environment is dynamic and nonlinear. Complexity emerges from the interaction between the system and environment, the elements of the system, where the emergent macroscopic patterns are more complex than the sum of the these low-level (microscopic) elements encompassed in the system. Understanding the evolution and development of adaptive systems still faces many mathematical challenges (Levin, 2003).

The concepts of external and internal complexities are used to analyze the relation between an adaptive system and its environment. The description given below is based on Jürgen Jost's (2004) work, which introduced these two concepts and applied the theoretical framework to the construction of learning models, e.g., to design neural network architectures. In the following, the concepts are mainly applied to analyze the interaction between the system and its environment. The interaction among individual elements of the system is less discussed however, the concepts can be explored in that situation too.

Theory

Adaptive System Environment and Regularities

The environment of an adaptive system is more complex than the system itself and its changes cannot be completely predictable for the system. However, the changes of the environment are not purely random and noisy; there exist regularities in the environment. An adaptive system can recognize these regularities, and depending on these regularities the system will express them through its internal process in order to adapt to the environment.

The input that an adaptive system receives or extracts from its environment usually includes two parts: one is the part with regularities; and another is that appears random to the system. The part of regularities is useful and meaningful. An adaptive system will represent these regularities by internal processes. But the part of random input is useless, and even at the worst it will be detrimental for an adaptive system. However, it will depend on the adaptive system's internal model of the external environment for how to determine which part of input is meaningful and regular, and which part is random and devoid of meaning and structure.

An adaptive system will translate the external regularities into its internal ones, and only the regularities are useful to the system. The system tries to extract as many regularities as possible, and to represent these regularities as efficiently as possible in order to make optimal use of its capacity.

The notions of external complexity and internal complexity are used to investigate these two complementary aspects conceptually and quantitatively. In terms of these notions, an adaptive system aims to increase its external complexity and reduce its internal complexity.

The two processes operate on their own time scale but are intricately linked and mutually dependent on each other. For example, the internal complexity will be only reduced if the external complexity is fixed. Under fixed inputs received from the external environment, an adaptive system can represent these inputs systems more efficiently and optimize its internal structure. If the external complexity is increased, e.g., if additional new input is required to handle by the system, then it is necessary to increase its internal complexity.

The increase of internal complexity may occur through the creation of redundancy in the existing adaptive system, e.g., to duplicate some internal structures, and then enable the system to handle more external input. Once the input is fixed, the adaptive system then will represent the input as efficiently as possible and reduce the internal input. The decrease of internal complexity can be achieved through discarding some input as meaningless and irrelevant, e.g., leaving some regularities out for the purpose.

Since the inputs relevant to the systems are those which can be reflected in the internal model, the external complexity is not equivalent to the amount of raw data received from the environment. In fact, it is only relevant to the inputs which can be processed in the internal model, or observations in some adaptive systems. Thus the external complexity ultimately is decided by the internal model constructed by the system.

External and Internal Complexities

External complexity means data complexity, which is used to measure the amount of input received from the environment for the system to handle and process. Such a complexity can be measured by entropy in the term of information theory.

Internal complexity is model complexity, which is used to measure the complexity of a model for representing the input or information received by the system.

The aim of the adaptive system is to obtain an efficient model as simple as possible, with the capacity to handle as much input as possible. On one hand, the adaptive system will try to maximize its external complexity and then to adapt to its environment in a maximal way; on the other hand, to minimize its internal complexity and then to construct a model to process the input in a most efficient way.

These two aims sometimes seem conflicting, but such a conflict can be avoided when these two processes operate on different time scales. If given a model, the system will organize the input data and try to increase its ability to deal with the input from its environment, and then increase its external complexity. If given the input, conversely, it tries to simplify its model which represents that input and thus to decrease the internal complexity. The meaning of the input is relevant to the time scale under investigation. On a short time scale, for example, the input may consist of individual signals, but on a long time scale, it will be a sequence of signals which satisfies a probability distribution. A good internal model tries to express regularities in the input sequence, rather than several individual signals. And the decrease of internal complexity will happen on this time scale.

A formal definition of the internal and external complexities concepts is based on the concept of entropy from statistical mechanics and information theory. Given a model θ, the system can model data as with $X(\theta) = (X_1, \ldots, X_k)$, which is assumed to have an internal probability distribution $P(X(\theta))$ so that entropy can be computed. The external complexity is defined by

$$-\sum_{i=1}^{k} P(X_i(\theta)) \log_2 P(X_i(\theta)). \qquad (1)$$

An adaptive system tries to maximize the above external complexity.

The probability distribution $P(X(\theta))$ is for quantifying the information value of the data $X(\theta)$. The value of information can be described in other approaches, e.g., the length of the representation of the data in the internal code of the system (Rissanen, 1989). In this case, the optimal coding is a consequence of the minimization of internal complexity, and then the length of the representation of data $X_i(\theta)$ behaves like $\log_2 P(X(\theta))$ (Rissanen, 1989).

On a short time scale, for a given model θ, the system tries to increase the amount of meaningful input information $X(\theta)$. On a long time scale, when the input is given, e.g., when the system has gathered a set of inputs on a time scale with a stationary probability distribution of input patterns Ξ, then the model should be improved to handle the input as efficiently as possible and reduce the complexity of the model. This complexity, or internal complexity, is defined by

$$-\sum_{i=1}^{k} P(\Xi_i \mid \theta) \log_2 P(\Xi_i \mid \theta) - \log_2 P(\theta), \qquad (2)$$

with respect to the model θ.

If Rissanen's (1989) ►minimum description length principle is applied to the above formula, then the optimal model will satisfy the variation problem

$$\min_{\theta} \left(-\log_2 P(\Xi \mid \theta) - \log_2 P(\theta) \right). \qquad (3)$$

Here in the above minimization problem, there are two objectives to minimize. The first term is to measure how efficiently the model represents or encodes the data; and the second one is to measure how complicated the model is. In computer science, this latter term corresponds to the length of the program required to encode the model.

The concepts of external and internal complexities can be applied into a system divided into subsystems. In this case, some internal part of the original whole system will become external to a subsystem. Thus the internal input of a subsystem consists of original external input and also input from the rest of the system, i.e., other subsystems.

Application: Learning

The discussion of these two concepts, external and internal complexities, can be put into the background of learning. In statistical learning theory (Vapnik, 1998), the criterion for evaluating a learning process is the expected prediction error of future data by the model

based on training data set with partial and incomplete information. The task is to construct a probability distribution drawn from an a-priori specific class for representing the distribution underlying the input data received. Usually, if a higher error is produced by a model on the training data, then a higher error will be expected on the future data. The error will depend on two factors: one is the accuracy of the model on the training data set, another is the simplicity of the model itself. The description of the data set can be split into two parts, the regular part, which is useful in constructing the model; and the random part, which is a noise to the model.

The learning process fits very well into the theory framework of internal and external complexities. If the model is too complicated, it will bring the risk of over-fitting the training data. In this case, some spurious or putative regularity is incorporated into the model, which will not appear in the future data. The model should be constrained within some model class with bounded complexity. This complexity in this context of statistical learning theory is measured by the Vapnik-Chervonenkis dimension (see ▶VC Dimension) (Vapnik, 1998). Under the simplest form of statistical learning theory, the system aims at finding a representation with smallest error in a class with given complexity constraints; and then the model should minimize the expected error on future data and also over-fitting error.

The two concepts of over-fitting and leaving out regularities can be distinguished in the following sense. The former is caused by the noise in the data, i.e., the random part of the data, and this leads to putative regularities, which will not appear in the future data. The latter, leaving out regularities, means that the system can forgo some part of regularities in the data, or it is possible to make data compression. Thus, leaving out regularities can be used to simplify the model and reduce the internal complexity. However, a problem is still waiting for answer here, that is, what regularities in the data set are useful for data compression and also meaningful for future prediction; and what parts are random to the model.

The internal complexity is the model complexity. If the internal complexity is chosen too small, then the model does not have enough capacity to represent all the important features of the data set. If the internal complexity is too large, on the other hand, then the model does not represent the data efficiently. The internal complexity is preferably minimized under appropriate constraints on the adequacy of the representation of data. This is consistent with Rissanen's principle of Minimum Description Length (Rissanen, 1989) to represent a given data set in the most efficient way. Thus a good model is both to simplify the model itself and to represent the data efficiently.

The external complexity is the data complexity which should be large to represent the input accurately. This is related to Jaynes' principle of maximizing the ignorance (Jaynes, 1957), where a model for representing data should have the maximal possible entropy under the constraint that all regularities can be reproduced. In this way, putative regularities could be eliminated in the model. However, this principle should be applied with some conditions as argued by Gell-Mann and Lloyd (1996); it cannot eliminate the essential regularities in the data, and an overlying complex model should be avoided.

For some learning system, only a selection of data is gathered and observed by the system. Thus a middle term, observation, is added between model and data. The concept of observation refers to the extraction of value of some specific quantity from a given data or data pool. What a system can observe depends on its internal structure and its general model of the environment. The system does not have direct access to the raw data, but through constructing a model of the environment solely on the basis of the values of its observation.

For such kind of learning system, Jaynes' principle (Jaynes, 1957) is still applicable for increasing the external complexity. For the given observation made on a data set, the maximum entropy representation should be selected. However, this principle is still subject to the modification of Gell-Mann and Lloyd (1996) to a principle where the model should not lose the essential regularities observed in the data.

By contrast, the observations should be selected to reduce the internal complexity. Given a model, if the observation can be made on a given data set, then these observations should be selected so as to minimize the resulting entropy of the model, with the purpose of minimizing the uncertainty left about the data. Thus it leads to reduce the complexity.

In most of the cases, the environment is dynamic, i.e., the data set itself can be varied, then the external

complexity should be maximized again. Thus the observation should be chosen for maximal information gain extracted from the data to increase the external complexity. Jaynes' principle (Jaynes, 1957) can be applied as the same as in previous discussion. But on a longer time scale, when the inputs reach some stationary distribution, the model should be simplified to reduce its internal complexity.

Recommended Reading

Gell-Mann, M., & Lloyd, S. (1996). Information measures, effective complexity, and total information. *Complexity, 2*(1), 44–52.
Holland, J. (1992). *Adaptation in natural and artificial systems*. Cambridge, MA: MIT Press.
Holland, J. (1995). *Hidden order: How adaptation builds complexity*. Reading, MA: Addison-Wesley.
Jaynes, E. (1957). Information theory and statistical mechanics. *Physical Review, 106*(4), 620–630.
Jost, J. (2004). External and internal complexity of complex adaptive systems. *Theory in Biosciences, 123*(1), 69–88.
Levin, S. (2003). Complex adaptive systems: Exploring the known, the unknown and the unknowable. *Bulletin of the American Mathematical Society, 40*(1), 3–19.
Rissanen, J. (1989). *Stochastic complexity in statistical inquiry*. Singapore: World Scientific.
Vapnik, V. (1998). *Statistical learning theory*. New York: John Wiley & Sons.
Waldrop, M. (1992). *Complexity: The emerging science at the edge of order and chaos*. New York: Simon & Schuster.

Complexity of Inductive Inference

Sanjay Jain, Frank Stephan
National University of Singapore,
Singapore, Republic of Singapore

Definition

In ▶inductive inference, the complexity of learning can be measured in various ways: by the number of hypotheses issued in the worst case until the correct hypothesis is found; by the number of data items to be consumed or to be memorized in order to learn in the worst case; by the Turing degree of oracles needed to learn the class under a certain criterion; by the intrinsic complexity which is – like the Turing degrees in recursion theory – a way to measure the complexity of classes by using reducibilities between them.

Detail

We refer the reader to the article ▶Inductive Inference for basic definitions in inductive inference and the notations used below. Let $\mathbb{N}$ denote the set of natural numbers. Let $\varphi_0, \varphi_1, \ldots$ denote a fixed acceptable programming system (Rogers, 1967). Let $W_i = \text{domain}(\varphi_i)$.

Mind Changes and Anomalies

The first measure of complexity of learning can be considered as the number of mind changes needed before the learner converges to its final hypothesis in the **TxtEx** model of learning. The number of mind changes by a learner M on a text T can be counted as $\text{card}(\{m : ? \neq M(T[m]) \neq M(T[m+1])\})$. A learner M $\mathbf{TxtEx}_n$ learns a class $\mathcal{L}$ of languages iff M **TxtEx** learns $\mathcal{L}$ and for all $L \in \mathcal{L}$, for all texts T for L, M makes at most n mind changes on T. $\mathbf{TxtEx}_n$ is defined as the collection of language classes which can be $\mathbf{TxtEx}_n$ identified (see Case & Smith (1983) for details).

Consider the class of languages $\mathcal{L}_n = \{L : \text{card}(L) \leq n\}$. It can be shown that $\mathcal{L}_{n+1} \in \mathbf{TxtEx}_{n+1} - \mathbf{TxtEx}_n$.

Now consider anomalous learning. A class $\mathcal{C}$ is $\mathbf{TxtEx}^a_b$-learnable iff there is a learner, which makes at most b mind changes (where $b = *$ denotes that the number of mind changes is finite on each text for a language in the class, but not necessarily bounded by a constant) and whose final hypothesis is allowed to make up to a errors (where $a = *$ denotes finitely many errors). For these learning criteria, we get a two-dimensional hierarchy on what can be learnt. Let $\mathcal{C}_n = \{f : \varphi_{f(0)} =^n f\}$. For a total function f, let $L_f = \{\langle x, f(x)\rangle : x \in \mathbb{N}\}$, where $\langle \cdot, \cdot \rangle$ denotes a computable pairing function: a bijective mapping from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. Let $\mathcal{L}_{\mathcal{C}} = \{L_f : f \in \mathcal{C}\}$. Then, one can show that $\mathcal{L}_{\mathcal{C}_{n+1}} \in \mathbf{TxtEx}^{n+1}_0 - \mathbf{TxtEx}^n$. Similarly, if we consider the class $\mathcal{S}_n = \{f : \text{card}(\{m : f(m) \neq f(m+1)\}) \leq n\}$, then one can show that $\mathcal{L}_{\mathcal{S}_{n+1}} \in \mathbf{TxtEx}^0_{n+1} - \mathbf{TxtEx}^*_n$ (we refer the reader to Case and Smith (1983) for a proof of the above).

Data and Time Complexity

Wiehagen (1986) considered the complexity of number of data needed for learning. Regarding time complexity, one should note the result by Pitt (1989) that any **TxtEx**-learnable class of languages can be **TxtEx**-learnt by a learner that has time complexity (with respect to

the size of the input) bounded by a linear function. This result is achieved by a delaying trick, where the learner just repeats its old hypothesis unless it has enough time to compute its later hypothesis. This seriously effects what one can say about time complexity of learning. One proposal made by Daley and Smith (1986) is to consider the total time used by the learner until its sequence of hypotheses converges, resulting in a possibly more reasonable measure of time in the complexity of learning.

Iterative and Memory-Bounded Learning

Another measure of complexity of learning can be considered when one restricts how much past data a learner can remember. Wiehagen introduced the concept of *iterative* learning in which the learner cannot remember any past data. Its new hypothesis is based only on its previous conjecture and the new datum it receives. In other words, there exists a recursive function F such that $\mathrm{M}(T[n+1]) = F(\mathrm{M}(T[n]), T(n))$, for all texts T and for all n. Here, $\mathrm{M}(T[0])$ is some fixed value, say the symbol '?' which is used by the learner to denote the absence of a reasonable conjecture. It can be shown that being iterative restricts the learning capacity of learners. For example, let $L_e = \{2x : x \in \mathbb{N}\}$ and let $\mathcal{L} = \{L_e\} \cup \{\{S \cup \{2n+1\}\} : n \in \mathbb{N}, S \subseteq L_e, \text{and } \max(S) \leq n\}$; then $\mathcal{L}$ can be shown to be **TxtEx**-learnable but not iteratively learnable.

Memory-bounded learning (see Lange & Zeugmann, 1996) is an extension of memory-limited learning, where the learner is allowed to memorize upto some fixed number of elements seen in the past. Thus, M is an m-memory-bounded learner if there exists a function *mem* and two computable functions mF and F such that, for all texts T and all n:

- $mem(T[0]) = \emptyset$;
- $\mathrm{M}(T[n+1]) = F(\mathrm{M}(T[n]), mem(T[n]), T(n+1))$;
- $mem(T[n+1]) = mF(\mathrm{M}(T[n]), mem(T[n]), T(n+1))$;
- $mem(T[n+1]) - mem(T[n]) \subseteq \{T(n+1)\}$;
- $\mathrm{card}(mem(T[n])) \leq m$.

It can be shown that the criteria of inference based on **TxtEx**-learning by m-memory-bounded learners form a proper hierarchy.

Besides memorizing some past elements seen, another way to address this issue is by giving feedback to the learner (see Case, Jain, Lange, & Zeugmann, 1999) on whether some element has appeared in the past data. A feedback learner is an iterative learner, which is additionally allowed to query whether certain elements appeared in earlier data. An n-feedback learner is allowed to make n such queries at each stage (when it receives the new input datum). Thus, M is an m-feedback learner if there exist computable functions Q and a F such that, for all texts T and all n:

- $Q(\mathrm{M}(T[n]), T(n))$ is defined and is a set of m elements;
- If $Q(\mathrm{M}(T[n]), T(n)) = (x_1, x_2, \ldots, x_m)$ then $\mathrm{M}(T[n+1]) = F(\mathrm{M}(T[n]), T(n), y_1, y_2, \ldots, y_m)$, where $y_i = 1$ iff $x_i \in \mathrm{ctnt}(T[n])$.

Again, it can be shown that allowing more feedback gives greater learning power, and thus one can get a hierarchy based on the amount of feedback allowed.

Complexity of Final Hypothesis

Another possibility on complexity of learning is to consider the complexity or size of the final grammar output by the learner. Freivalds (1975) considered the case when the final program/grammar output by the learner is minimal: that is, there is no smaller index that accepts/generates the same language. He showed that this severely restricts the learning capacity of learners. Not only that, the learning capacity depends on the acceptable programming system chosen, unlike the case for most other criteria of learning such as **TxtEx** or **TxtBc**, which are independent of the acceptable programming system chosen. In particular, there are acceptable programming systems in which only classes containing finitely many infinite languages can be learnt using minimal final grammars (see Freivalds, 1975; Jain and Sharma, 1993). Chen (1982) considered a modification of such a paradigm where one considers convergence to nearly minimal grammars rather than minimal. That is, instead of requiring that the final grammars are minimal, one requires that they are within a recursive function h of minimal. Here h may depend on the class being learnt. Chen showed that this allows one to have the criteria of minimal learnability

to be independent of the acceptable programming system chosen. However, one can show that some simple classes are not minimally learnable. An example of such a class is the class $\mathcal{L}_{\mathcal{C}}$ which is derived from $\mathcal{C} = \{f : \forall^{\infty} \times [f(x) = 0]\}$, the class of all functions which are almost everywhere 0.

Intrinsic Complexity

Another way to consider complexity of learning is to consider relative complexity in a way similar to how one considers Turing reductions in computability theory. Such a notion is called intrinsic complexity of the class. This was first considered by Freivalds et al. (1995) for function learning. Jain and Sharma (1996) considered it for language learning, and the following discussion is from there.

An *enumeration operator* (see Rogers, 1967), Θ, is an algorithmic mapping from SEQ into SEQ such that the following two conditions are satisfied:

- for all $\sigma, \tau \in$ SEQ, if $\sigma \subseteq \tau$, then $\Theta(\sigma) \subseteq \Theta(\tau)$;
- for all texts T, $\lim_{n\to\infty} |\Theta(T[n])| = \infty$.

By extension, we think of Θ as also mapping texts to texts such that $\Theta(T) = \bigcup_n \Theta(T[n])$. Furthermore, we define $\Theta(L) = \{\text{ctnt}(\Theta(T)) : T \text{ is a text for } L\}$. Intuitively, $\Theta(L)$ denotes the set of languages to whose texts Θ maps texts of L. The reader should note the overloading of this notation because the type of the argument to Θ could be a sequence, a text or a language.

One says that a sequence of grammars $g_0, g_1, \ldots$ is an acceptable **TxtEx**-sequence for L if the sequence of grammars converges to a grammar for L.

$\mathcal{L}_1 \leq_{weak} \mathcal{L}_2$ iff there are two operators Θ and Ψ such that for all $L \in \mathcal{L}_1$, for all texts T for L, $\Theta(T)$ is a text for some $L' \in \mathcal{L}_2$ such that if $g_0, g_1, \ldots$ is an acceptable **TxtEx**-sequence for L' then $\Psi(g_0, g_1, \ldots)$ is an acceptable **TxtEx**-sequence for L.

Note that different texts for the same language L may be mapped by Θ to texts for different languages in $\mathcal{L}_2$ above. If we require that different texts for L are mapped to texts for the same language L' in $\mathcal{L}_2$, then we get a stronger notion of reduction called strong reduction: $\mathcal{L}_1 \leq_{strong} \mathcal{L}_2$ iff $\mathcal{L}_1 \leq_{weak} \mathcal{L}_2$ and for all $L \in \mathcal{L}_1$, $\Theta(L)$ contains only one language, where Θ is as in the definition for $\leq_{weak}$ reduction.

It can be shown that *FIN* is a complete class for **TxtEx**-identification with respect to $\leq_{weak}$ reduction (see Jain & Sharma, 1996). Interestingly it was shown that the class of pattern languages (Angluin, 1980), the class $SD = \{L : W_{\min(L)} = L\}$ and the class $COINIT = \{\{x : x \geq n\} : n \in \mathbb{N}\}$ are all equivalent under $\leq_{strong}$. Let *code* be a bijective mapping from non-negative rational numbers to natural numbers. Then, one can show that the class RINIT $= \{\{code(x) : 0 \leq x \leq r, x \text{ is a rational number}\} : 0 \leq r \leq 1, r \text{ is a rational number }\}$ is $\leq_{strong}$ complete for **TxtEx** (see Jain, Kinber, & Wiehagen, 2001).

Interestingly every finite directed acyclic graph can be embedded into the $\leq_{strong}$ degree structure (Jain & Sharma, 1997). On the other hand the degree structure is non-dense in the sense that there exist classes $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathcal{L}_1 <_{strong} \mathcal{L}_2$, but for any class $\mathcal{L}$ such that $\mathcal{L}_1 \leq_{strong} \mathcal{L} \leq_{strong} \mathcal{L}_2$, either $\mathcal{L}_1 \equiv_{strong} \mathcal{L}$ or $\mathcal{L} \equiv_{strong} \mathcal{L}_2$. Similar result holds for $\leq_{weak}$ reducibility (see Jain & Sharma, 1997).

Interesting connections between learning of elementary formal systems (Shinohara, 1994), intrinsic complexity and ordinal mind changes (Freivalds & Smith, 1993) were shown in (Jain & Sharma, 1997).

Learning Using Oracles

Another method to measure complexity of learning is to see how powerful an oracle (given to the learning machine) has to be to make a class learnable. It can be shown that an oracle A permits to explanatorily learn the class of all recursive functions iff A is high (Adleman & Blum, 1991). Furthermore, an oracle is trivial, that is, does not give additional learning power for explanatory learning of function classes iff the oracle has 1-generic Turing degree and is Turing reducible to the halting problem (Slaman & Solovay, 1991). The picture is a bit different in the general case of learning languages. For every oracle A there is an oracle B and a class, which is **TxtEx**-learnable using the oracle B but not using the oracle A (Jain & Sharma, 1993). Note that there are also classes of languages like Gold's class of all finite languages plus the set of natural numbers which are not **TxtEx**-learnable using any oracle. Furthermore, for oracles above the halting problem, **TxtEx**-learning and **TxtBc**-learning using these oracles coincide.

Recommended Reading

Adleman, L., & Blum, M. (1991). Inductive inference and unsolvability. *Journal of Symbolic Logic, 56*, 891–900.

Angluin, D. (1980). Finding patterns common to a set of strings. *Journal of Computer and System Sciences, 21*, 46–62.

Case, J., Jain, S., & Lange, S., & Zeugmann, T. (1999). Incremental concept learning for bounded data mining. *Information and Computation, 152*(1), 74–110.

Case, J., & Smith, C. H. (1983). Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science, 25*, 193–220.

Daley, R. P., & Smith, C. H. (1986). On the complexity of inductive inference. *Information and Control, 69*, 12–40.

Chen, K.-J. (1982). Tradeoffs in inductive inference of nearly minimal sized programs. *Information and Control, 52*, 68–86.

Freivalds, R. (1975). Minimal Gödel numbers and their identification in the limit. *Lecture Notes in Computer Science, 32*, 219–225.

Freivalds, R., Kinber, E., & Smith, C. H. (1995). On the intrinsic complexity of learning. *Information and Computation, 123*, 64–71.

Freivalds, R., & Smith, C. H. (1993). On the role of procrastination in machine learning. *Information and Computation, 107*(2), 237–271.

Jain, S., Kinber, E., & Wiehagen, R. (2001). Language learning from texts: Degrees of intrinsic complexity and their characterizations. *Journal of Computer and System Sciences, 63*, 305–354.

Jain, S., & Sharma, A. (1993). On the non-existence of maximal inference degrees for language identification. *Information Processing Letters, 47*, 81–88.

Jain, S., & Sharma, A. (1994). Program size restrictions in computational learning. *Theoretical Computer Science, 127*, 351–386.

Jain, S., & Sharma, A. (1996). The intrinsic complexity of language identification. *Journal of Computer and System Sciences, 52*, 393–402.

Jain, S., & Sharma, A. (1997). The structure of intrinsic complexity of learning. *Journal of Symbolic Logic, 62*, 1187–1201.

Jain, S., & Sharma, A. (1997). Elementary formal systems, intrinsic complexity and procrastination. *Information and Computation, 132*, 65–84.

Lange, S., & Zeugmann, T. (1996). Incremental learning from positive data. *Journal of Computer and System Sciences, 53*, 88–103.

Pitt, L. (1989). Inductive inference, DFAs, and computational complexity. *Analogical and inductive inference, second international workshop, AII 1989*, LNAI. (Vol. 397, pp. 18–44) Heidelberg: Springer.

Rogers, H. (1967). *Theory of recursive functions and effective computability*. New York: McGraw-Hill (Reprinted, MIT Press 1987).

Shinohara, T. (1994). Rich classes inferable from positive data: Length–bounded elementary formal systems. *Information and Computation, 108*, 175–186.

Slaman, T. A. & Solovay, R. (1991). When oracles do not help. *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, (pp. 379–383), Morgan Kaufmann.

Wiehagen, R. (1976). Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. *Journal of Information Processing and Cybernetics* (*EIK*), *12*, 93–99.

Wiehagen, R. (1986). On the complexity of effective program synthesis. In: K. Jantke (Ed.). Analogical and Inductive Inference. *Proceedings of the International Workshop*, Springer LNCS, (Vol. 265, pp. 209–219).

Compositional Coevolution

Synonyms

Cooperative coevolution

Definition

A coevolutionary system constructed to learn composite solutions in which individuals represent different candidate components and must be evaluated together with other individuals in order to form a complete solution. Though not precisely the same as *cooperative coevolution*, there is a significant overlap.

Cross References

►Coevolutionary Learning

Computational Complexity of Learning

Sanjay Jain, Frank Stephan

National University of Singapore, Singapore, Republic of Singapore

Definition

Measures of the complexity of learning have been developed for a number of purposes including ►Inductive Inference, ►PAC Learning, and ►Query-Based Learning. The complexity is usually measured by the largest possible usage of ressources that can occur during the learning of a member of a class. Depending on the context, one measures the complexity of learning either by a single number/ordinal for the whole class or by a function in a parameter n describing the complexity of the target to be learnt. The actual measure can be the number of mind changes, the number of queries submitted to a teacher, the number of wrong conjectures issued, the number of errors made or the number of examples processed until learning succeeds. In addition to this, one can equip the learner with an oracle and determine the complexity of the oracle needed to perform

the learning process. Alternatively, in complexity theory, instead of asking for an NP-complete oracle to learn a certain class, the result can also be turned into the form "this class is unlearnable unless RP = NP" or something similar. (Here RP is the class of decision problems solvable by a randomized polynomial time algorithm and NP is the class of decision problems solvable by a nondeterministic polynomial time algorithm and both algorithms never give "yes" answer for an instance of the problem with "no" answer.)

Detail

In ▶PAC Learning, one usually asks how many examples are needed to learn the concept, where the number of examples needed mainly depends on the Vapnik Chervonenkis dimension of the class to be learnt, the error permitted, and the confidence required. Furthermore, for certain classes of finite Vapnik Chervonenkis dimension, learnability can still fail when the learner is required to be computable in polynomial time; hence there is, besides the dimension, also a restriction stemming from the computational complexity of problems such as the complexity of finding concepts consistent with all data observed so far.

For ▶Query-Based Learning, one common criterion to be looked at is the number of queries made during the learning process. If a class contains 2^n different $\{0,1\}$-valued functions f and one is required to learn the class using membership-queries, that is, by asking queries of the form whether $f(x) = 0$ or $f(x) = 1$, then there is a function f on which the learner needs at least n queries until it knows which of the given functions f is; for some classes consisting of 2^n functions the number of queries needed can be much worse – as much as $2^n - 1$. A well-known result of Angluin is that one can learn the class of all regular languages with polynomially many equivalence and membership queries measured with respect to the number of states of the smallest deterministic finite automaton accepting the language to be learnt. Further research has been done dealing with which query algorithms can be implemented by a polynomial time learner and which need for polynomial time learning, in addition to the teacher informing on the target concept, also some oracle supplying information that cannot be computed in polynomial time. See the entry ▶Query-Based Learning for an overview of these results.

For ▶Inductive Inference, most complexity measures are measures applying to the overall class and not just a parameterized version. When learning the class of all sets with up to n elements, the learner might first issue the conjecture $\emptyset$ and then revise (up to n times) its hypothesis when a new datum is observed; such a measure is called the mind change complexity of learning. Mind change complexity has been generalized to measure the complexity by recursive ordinals or the notation of these. Furthermore, one can measure the long term memory of past data observed either by a certain number of examples remembered or by the number of bits stored on a tape describing the long-term memory of the learner. Besides these quantitative notions, a further frequently studied question is the following: Which oracles support the learning process in a way that some classes become learnable using the oracle, but are unlearnable without using any oracle? An example of such a type of result is that the class of all recursive functions can be learnt if and only if the learner has access to a high oracle, that is, an oracle that permits to compute a function which dominates (i.e., grows faster than) every recursive function. See the entry ▶Complexity of Inductive Inference for more information.

Computational Discovery of Quantitative Laws

▶Equation Discovery

Concept Drift

Claude Sammut[1], Michael Harries[2]
[1]The University of New South Wales, Sydney, Australia
[2]Advanced Products Group, Citrix Labs, North Ryde, NSW, Australia

Synonyms

Context-sensitive learning; Learning with hidden context

Definition

Concept drift occurs when the values of hidden variables change over time. That is, there is some unknown

context for ►concept learning and when that context changes, the learned concept may no longer be valid and must be updated or relearned.

Motivation and Background

Prediction in real-world domains is complicated by potentially unstable phenomena that are not known in advance to the learning system. For example, financial market behavior can change dramatically with changes in contract prices, interest rates, inflation rates, budget announcements, and political and world events. Thus, concept definitions that may have been learned in one context become invalid in a new context. This *concept drift* can be due to changes in context and is often directly reflected by one or more attributes. When changes in context are not reflected by any known attributes they can be said to be *hidden*. Hidden changes in context cause problems for any predictive approach that assumes concept stability.

Structure of the Learning System

Machine learning approaches can be broadly categorized as either ►batch learning or ►incremental learning. Batch systems learn off-line by examining a large collection of instances *en masse* and form a single concept. Incremental systems evolve and change a concept definition as new observations are processed (Schlimmer & Granger 1986a; Aha et al., 1991; Koltzer & Maloof, 2003).

The most common approach to learning in domains with hidden changes in context has been to use an incremental learning approach in which the importance of older items is progressively decayed. A popular implementation of this, originally presented in Kubet (1989), is to use a window of recent instances from which concept updates are derived. Other examples of this approach include Widmer and Kubat (1996), Kubat and Widmer (1995), Kilander and Jansson (1993), and Salganikoff (1993). Swift adaptation to changes in context can be achieved by dynamically varying the window size in response to changes in accuracy and concept complexity (Widmer & Kubat, 1996).

There are many domains in which the context can be expected not only to change but for earlier contexts to hold again at some time in the future. That is, contexts can repeat in domains such as financial prediction, dynamic control, and underrepresented data mining tasks. In these domains, prediction accuracy can be improved by storing knowledge about past contexts for reuse. FLORA3 (Widmer & Kubat, 1993) addresses domains in which contexts recur by storing and retrieving concepts that appear stable as the learner traverses the series of input data.

In many situations, there is no constraint to learn incrementally. For example, many organizations maintain large data bases of historical data that are suitable for data mining. These data bases may hold instances that belong to a number of contexts but do not have this context explicitly recorded. Many of these data bases may incorporate time as an essential attribute, for example, financial records and stock market price data. Interest in mining datasets of this nature suggests the need for systems that can learn global concepts and are sensitive to changing and hidden contexts. Systems such as FLORA3 also imply that an off-line recognition of stable concepts can be useful for ►on-line prediction.

An alternative to on-line learning for domains with hidden changes in context is to examine the data *en masse* in an attempt to directly identify concepts associated with stable, hidden contexts. Some potential benefits of such an approach are:

1. Context specific (known as local) concepts could be used as part of a multiple model on-line predictive system.
2. Local concepts could be verified by experts, or used to improve domain understanding.
3. A model of the hidden context could be induced using context characteristics such as context duration, order, and stability. The model could also use existing attributes and domain feedback if available.
4. Stable contexts identified could be used as target characteristics for selecting additional attributes from the outside world as part of an iterative data mining process.

Splice (Harries, Sammut, & Horn, 1998) is a ►meta-learning system that implements a context sensitive batch learning approach. Splice is designed to identify intervals with stable hidden context, and to induce and refine local concepts associated with hidden contexts.

Identifying Context Change

In many domains with hidden changes in context, time can be used to differentiate hidden contexts. Most

machine learning approaches to these domains do not explicitly represent time as they assume that current context can be captured by focusing on recent examples. The implication is that hidden context will be reflected in contiguous intervals of time. For example, an attempt to build a system to predict changes in the stock market could produce the following ►decision tree:

```
Year > 2002
        Year < 2005
                Attribute A = true :  Market Rising
                Attribute A = false :  Market Falling
        Year ≥ 2005
                Attribute B = true :  Market Rising
                Attribute B = false :  Market Falling
```

This tree contains embedded knowledge about two intervals of time: in one of which, 2002–2004, attribute A is predictive; in the other, 2005 onward, attribute B is predictive. As time (in this case, year) is a monotonically increasing attribute, future classification using this decision tree will only use attribute B. If this domain can be expected to have recurring hidden context, information about the prior interval of time could be valuable.

The decision tree in the example above contains information about changes in context. We define context as:

- Context is any attribute whose values are largely independent but tend to be stable over contiguous intervals of another attribute known as the environmental attribute.

The ability of decision trees to capture context is associated with the fact that decision tree algorithms use a form of context-sensitive feature selection (CSFS). A number of machine learning algorithms can be regarded as using CSFS including decision tree algorithms (Quinlan, 1993), ►rule induction algorithms (Clark & Niblett, 1989), and ►ILP systems (Quinlan, 1990). All of these systems produce concepts containing local information about context.

When contiguous intervals of time reflect a hidden attribute or context, we call time the environmental attribute. The environmental attribute is not restricted to time alone as it could be any ordinal attribute over which instances of a hidden context are liable to be contiguous. There is also no restriction, in principle, to one dimension. Some alternatives to time as environmental attributes are dimensions of space, and space–time combinations.

Given an environmental attribute, we can utilize a CSFS machine learning algorithm to gain information on likely hidden changes in context. The accuracy of the change points found will be dependent upon at least hidden context duration, the number of different contexts, the complexity of each local concept, and noise.

The CSFS identified context change points can be expected to contain errors of the following types:

1. ►Noise or serial correlation errors. These would take the form of additional incorrect change points.
2. Errors due to the repetition of tests on time in different parts of the concept. These would take the form of a group of values clustered around the actual point where the context changed.
3. Errors of omission, change points that are missed altogether.

The initial set of identified context changes can be refined by contextual ►clustering.

This process combines similar intervals of the dataset, where the similarity of two intervals is based upon the degree to which a partial model is accurate on both intervals.

Recent Advances

With the increasing amount of data being generated by organizations, recent work on concept drift has focused on mining from high volume ►data streams Hulten, Spencer, & Domingos, 2001; Wang, Fan, Yu, & Han, 2003; Koltzer & Maloof, 2003, Mierswa, Wurst, Klinkenberg, Scholz, & Euler, 2006; Chu & Zaniolo, 2004; Gaber, Zaslavsky, & Krishnaswamy, 2005. Methods such as Hulten *et al*' s, combine decision tree learning with incremental methods for efficient updates, thus avoiding relearning large decision trees. Koltzer and Maloof also use incremental methods combined in an ►ensemble.

Cross References

►Decision Trees
►Ensemble Methods
►Incremental Learning
►Inductive Logic Programming
►Lazy Learning

Recommended Reading

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*, 37–66.

Chu, F., & Zaniolo, C. (2004). Fast and light boosting for adaptive mining of data streams. In *Advances in knowledge discovery and data mining. Lecture notes in computer science* (Vol. 3056, pp. 282–292). Springer.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*, 261–283.

Clearwater, S., Cheng, T.-P., & Hirsh, H. (1989). Incremental batch learning. In *Proceedings of the sixth international workshop on machine learning* (pp. 366–370). Morgan Kaufmann.

Domingos, P. (1997). Context-sensitive feature selection for lazy learners. *Artificial Intelligence Review, 11*, 227–253. [Aha, D. (Ed.). Special issue on lazy learning.]

Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2005). Mining data streams: A review. *SIGMOD Rec., 34*(2), 18–26.

Harries, M., & Horn, K. (1996). Learning stable concepts in domains with hidden changes in context. In M. Kubat & G. Widmer (Eds.), *Learning in context-sensitive domains (workshop notes). 13th international conference on machine learning,* Bari, Italy.

Harries, M. B., Sammut, C., & Horn, K. (1998). Extracting hidden context. *Machine Learning, 32*(2), 101–126.

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 97–106). New York: ACM.

Kilander, F., & Jansson, C. G. (1993). COBBIT – A control procedure for COBWEB in the presence of concept drift. In P. B. Brazdil (Ed.), *European conference on machine learning* (pp. 244–261). Berlin: Springer.

Kolter, J. Z., & Maloof, M. A. (2003). Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Third IEEE international conference on data mining ICDM-2003* (pp. 123–130). IEEE CS Press.

Kubat, M. (1989). Floating approximation in time-varying knowledge bases. *Pattern Recognition Letters, 10*, 223–227.

Kubat, M. (1992). A machine learning based approach to load balancing in computer networks. *Cybernetics and Systems Journal*.

Kubat, M. (1996). Second tier for decision trees. In *Machine learning: Proceedings of the 13th international conference* (pp. 293–301). California: Morgan Kaufmann.

Kubat, M., & Widmer, G. (1995). Adapting to drift in continuous domains. In *Proceedings of the eighth European conference on machine learning* (pp. 307–310). Berlin: Springer.

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., & Euler, T. (2006). Yale: Rapid prototyping for complex data mining tasks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 935–940). New York: ACM.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann: San Mateo.

Salganicoff, M. (1993). Density adaptive learning and forgetting. In *Machine learning: Proceedings of the tenth international conference* (pp. 276–283). San Mateo: Morgan Kaufmann.

Schlimmer, J. C., & Granger, R. I., Jr. (1986a). Beyond incremental processing: Tracking concept drift. In *Proceedings AAAI-86* (pp. 502–507). Los Altos: Morgan Kaufmann.

Schlimmer, J., & Granger, R., Jr. (1986b). Incremental learning from noisy data. Machine Learning, *1*(3), 317–354.

Turney, P. D. (1993a). Exploiting context when learning to classify. In P. B. Brazdil (Ed.), European conference on machine learning (pp. 402–407). Berlin: Springer.

Turney, P. D. (1993b). Robust classification with context sensitive features. In *Paper presented at the industrial and engineering applicatións of artificial intelligence and expert systems*.

Turney, P., & Halasz, M. (1993). Contextual normalization applied to aircraft gas turbine engine diagnosis. *Journal of Applied Intelligence, 3*, 109–129.

Wang, H., Fan, W., Yu, P. S., & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 226–235). New York: ACM.

Widmer, G. (1996). Recognition and exploitation of contextual clues via incremental meta-learning. In L. Saitta (Ed.), *Machine learning: Proceedings of the 13th international workshop* (pp. 525–533). San Francisco: Morgan Kaufmann.

Widmer, G., & Kubat, M. (1993). Effective learning in dynamic environments by explicit concept tracking. In P. B. Brazdil (Ed.), *European conference on machine learning* (pp. 227–243). Berlin: Springer.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23, 69–101.

Concept Learning

Claude Sammut
The University of New South Wales, Sydney, NSW, Australia

Synonyms

Categorization; Classification learning

Definition

The term *concept learning* is originated in psychology, where it refers to the human ability to learn categories for object and to recognize new instances of those categories. In machine learning, concept is more formally

defined as "inferring a boolean-valued function from training examples of its inputs and outputs" (Mitchell, 1997).

Background

Bruner, Goodnow, and Austin (1956) published their book *A Study of Thinking*, which became a landmark in psychology and would later have a major impact on machine learning. The experiments reported by Bruner, Goodnow, and Austin were directed toward understanding a human's ability to categorize and how categories are learned.

> We begin with what seems a paradox. The world of experience of any normal man is composed of a tremendous array of discriminably different objects, events, people, impressions... But were we to utilize fully our capacity for registering the differences in things and to respond to each event encountered as unique, we would soon be overwhelmed by the complexity of our environment... The resolution of this seeming paradox... is achieved by man's capacity to categorize. To categorize is to render discriminably different things equivalent, to group objects and events and people around us into classes... The process of categorizing involves... an act of invention... If we have learned the class "house" as a concept, new exemplars can be readily recognised. The category becomes a tool for further use. The learning and utilization of categories represents one of the most elementary and general forms of cognition by which man adjusts to his environment.

The first question that they had to deal with was that of representation: what is a concept? They assumed that objects and events could be described by a set of attributes and were concerned with how inferences could be drawn from attributes to class membership. Categories were considered to be of three types: conjunctive, disjunctive, and relational.

> ...when one learns to categorize a subset of events in a certain way, one is doing more than simply learning to recognise instances encountered. One is also learning a rule that may be applied to new instances. The concept or category is basically, this "rule of grouping" and it is such rules that one constructs in forming and attaining concepts.

The notion of a rule as an abstract representation of a concept influenced research in machine learning. For example, ▶decision tree learning was used as a means of creating a cognitive model of concept learning (Hunt, Martin, & Stone, 1966). This model later inspired Quinlan's development of ID3 (Quinlan, 1983).

The learning experience may be in the form of examples from a trainer or the results of trial and error. In either case, the program must be able to represent its observations of the world, and it must also be able to represent hypotheses about the patterns it may find in those observations. Thus, we will often refer to the ▶observation language and the ▶hypothesis language. The observation language describes the inputs and outputs of the program and the hypothesis language describes the internal state of the learning program, which corresponds to its theory of the concepts or patterns that exist in the data.

The input to a learning program consists of descriptions of objects from the universe and, in the case of ▶supervised learning, an output value associated with the example. The universe can be an abstract one, such as the set of all natural numbers, or the universe may be a subset of the real world. No matter which method of representation we choose, descriptions of objects in the real world must ultimately rely on measurements of some properties of those objects. These may be physical properties such as size, weight, and color or they may be defined for objects, for example, the length of time a person has been employed for the purpose of approving a loan. The accuracy and reliability of a learned concept depends on the accuracy and reliability of the measurements.

A program is limited in the concepts that it can learn by the representational capabilities of both observation and hypothesis languages. For example, if an attribute/value list is used to represent examples for an induction program, the measurement of certain attributes and not others clearly places bounds on the kinds of patterns that the learner can find. The learner is said to be *biased* by its observation language (see ▶Language Bias). The hypothesis language also places constraints on what may and may not be learned. For

example, in the language of attributes and values, relationships between objects are difficult to represent. Whereas, a more expressive language, such as first-order logic, can easily be used to describe relationships.

Unfortunately, representational power comes at a price. Learning can be viewed as a search through the space of all sentences in a language for a sentence that best describes the data. The richer the language, the larger is the search space. When the search space is small, it is possible to use "brute force" search methods. If the search space is very large, additional knowledge is required to reduce the search.

Rules, Relations, and Background Knowledge

In the early 1960s, there was no discipline called "machine learning." Instead, learning was considered to be part of "pattern recognition," which had not yet split from AI. One of the main problems addressed at that time was how to represent patterns so that they could be recognized easily. Symbolic description languages were developed to be expressive and learnable.

Banerji (1960, 1962) first devised a language, which he called a "description list," which utilized an object's attributes to perform pattern recognition. Pennypacker, a masters student of Banerji at the Case Institute of Technology, implemented the recognition procedure and also used Bruner, Goodnow, and Austin's *Conservative Focussing Strategy* to learn conjunctive concepts (Pennypacker, 1963). Bruner, Goodnow, and Austin describe the strategy as follows:

> ... this strategy may be described as finding a positive instance to serve as a focus, then making a sequence of choices each of which alters but one attribute value [of the focus] and testing to see whether the change yields a positive or negative instance. Those attributes of the focus which, when changed, still yield positive instance are *not* part of the concept. Those attributes of the focus that yield negative instances when changed *are* features of the concept.

The strategy is only capable of learning *conjunctive concepts*, that is, the concept description can only consist of a simple conjunction of tests on attribute values. Recognizing the limitations of simple attribute/value representations, Banerji (1964) introduced the use of predicate logic as a description language. Thus, Banerji was one of the earliest advocates of what would, many years later, become *Inductive Logic Programming*.

In the 1970s, a series of algorithms emerged that developed concept learning further. Winston's ARCH program (Winston, 1970) was influential as one of the first widely known concept learning programs. Michalski (1973, 1983) devised the Aq family of learning algorithms that set some of the early benchmarks for learning programs. Early relational learning programs were developed by Hayes-Roth (1973), Hayes-Roth and McDermott (1977), and Vere (1975, 1977).

Banerji emphasized the importance of a description language that could "grow." That is, its descriptive power should increase as new concepts are learned. These concepts become background knowledge for future learning. A simple example from Banerji (1980) illustrates the use of background knowledge. There is a language for describing instances of a concept and another for describing concepts. Suppose we wish to represent the binary number, 10, by a left-recursive binary tree of digits "0" and "1":

$$[\text{head}: [\text{head}: 1; \text{tail}: \text{nil}]; \text{tail}: 0]$$

"head" and "tail" are the names of attributes. Their values follow the colon. The concepts of binary digit and binary number are defined as

$$x \in \mathit{digit} \equiv x = 0 \vee x = 1$$
$$x \in \mathit{num} \equiv (\mathit{tail}(x) \in \mathit{digit} \wedge \mathit{head}(x) = \mathit{nil}) \vee (\mathit{tail}(x) \in \mathit{digit} \wedge \mathit{head}(x) \in \mathit{num})$$

Thus, an object belongs to a particular class or concept if it satisfies the logical expression in the body of the description. Note that the concept above is *disjunctive*. Predicates in the expression may test the membership of an object in a previously learned concept and can express *relations* between objects. Cohen and Sammut (1982) devised a learning system based on Banerji's ideas of a growing concept description language and this was further extended by Sammut and Banerji (1986).

Concept Learning and Noise

One of the most severe drawbacks of early concept learning systems was that they assumed that data sets

were not noisy. That is, all attribute values and class labels in the training data are assumed to be correct. This is unrealistic in most real applications. Thus, concept learning systems began incorporating statistical measures to minimize the effects of noise and to estimate error rates (Breiman, Friedman, Olshen, & Stone, 1984; Cohen, 1995; Quinlan, 1986, 1993).

Learning to classify objects from training examples has gone on to become one of the central themes of machine learning research. As the robustness of classification systems has increased, they have found many applications, particularly in data mining but in a broad range of other areas.

Cross References

►Data Mining
►Decision Tree Learning
►Inductive Logic Programming
►Learning as Search
►Relational Learning
►Rule Learning

Recommended Reading

Banerji, R. B. (1960). An information processing program for object recognition. *General Systems*, *5*, 117–127.

Banerji, R. B. (1962). The description list of concepts. *Communications of the Association for Computing Machinery*, *5*(8), 426–432.

Banerji, R. B. (1964). A Language for the Description of Concepts. *General Systems, 9*, 135–141.

Banerji, R. B. (1980). *Artificial intelligence: A theoretical approach*. New York: North Holland.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth.

Bruner, J. S., Goodnow, J. J., & Austin, G. A. (1956). *A study of thinking*. New York: Wiley.

Cohen, B. L., & Sammut, C. A. (1982). Object recognition and concept learning with CONFUCIUS. *Pattern Recognition Journal*, *15*(4), 309–316.

Cohen, W. W. (1995). In fast effective rule induction. In *Proceedings of the twelfth international conference on machine learning*, Lake Tahoe, California. Menlo Park: Morgan Kaufmann.

Hayes-Roth, F. (1973). A structural approach to pattern learning and the acquisition of classificatory power. In *First international joint conference on pattern recognition* (pp. 343–355). Washington, D.C.

Hayes-Roth, F., & McDermott, J. (1977). Knowledge acquisition from structural descriptions. In *Fifth international joint conference on artificial intelligence* (pp. 356–362). Cambridge, MA.

Hunt, E. B., Marin, J., & Stone, P. J. (1966). *Experiments in induction*. New York: Academic.

Michalski, R. S. (1973). Discovering classification rules using variable valued logic system VL1. In *Third international joint conference on artificial intelligence* (pp. 162–172). Stanford, CA.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. Palo Alto: Tioga.

Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.

Pennypacker, J. C. (1963). An elementary information processor for object recognition. SRC No. 30-I-63-1. Case Institute of Technology.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. Palo Alto: Tioga.

Quinlan, J. R. (1986). The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos: Morgan Kaufmann.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Sammut, C. A., & Banerji, R. B. (1986). Learning concepts by asking questions. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2, pp. 167–192). Los Altos, CA: Morgan-Kaufmann.

Vere, S. (1975). Induction of concepts in the predicate calculus. In *Fourth international joint conference on artificial intelligence* (pp. 351–356). Tbilisi, Georgia, USSR.

Vere, S. A. (1977). Induction of relational productions in the presence of background information. In *Fifth international joint conference on artificial intelligence*. Cambridge, MA.

Winston, P. H. (1970). Learning structural descriptions from examples. Unpublished PhD Thesis, MIT Artificial Intelligence Laboratory.

Conditional Random Field

A *Conditional Random Field* is a form of ►Graphical Model for segmenting and ►classifying sequential data. It is the ►discriminative learning counterpart to the ►generative learning Markov Chain model.

Recommended Reading

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th international conference on machine learning* (pp. 282–289). San Francisco, Morgan Kaufmann.

Confirmation Theory

The branch of philosophy concerned with how (and indeed whether) evidence can confirm a hypothesis, even though typically it does not entail it. A distinction is sometimes drawn between *total confirmation*: how well confirmed a hypothesis is, given all available evidence and *weight-of-evidence*: the amount of extra confirmation added to the total confirmation of a hypothesis by a particular piece of evidence. Confirmation is often measured by the probability of a hypothesis conditional on evidence.

Confusion Matrix

Kai Ming Ting
Monash University, Australia

Definition

A confusion matrix summarizes the classification performance of a ►classifier with respect to some ►test data. It is a two-dimensional matrix, indexed in one dimension by the true class of an object and in the other by the class that the classifier assigns. Table 1 presents an example of confusion matrix for a three-class classification task, with the classes *A*, *B*, and *C*.

The first row of the matrix indicates that 13 objects belong to the class *A* and that 10 are correctly classified as belonging to *A*, two misclassified as belonging to *B*, and one as belonging to *C*.

A special case of the confusion matrix is often utilized with two classes, one designated the *positive* class and the other the *negative* class. In this context, the four cells of the matrix are designated as ►*true positives* (TP), ►*false positives* (FP), ►*true negatives* (TN), and ►*false negatives* (FN), as indicated in Table 2.

A number of measures of classification performance are defined in terms of these four classification outcomes.

►*Specificity* = ►*True negative rate* = TN/(TN + FP)

►*Sensitivity* = ►*True positive rate* = ►*Recall* = TP/(TP + FN)

Confusion Matrix. Table 1 An example of three-class confusion matrix

		Assigned Class		
		A	*B*	*C*
Actual Class	*A*	10	2	1
	B	0	6	1
	C	0	3	8

Confusion Matrix. Table 2 The outcomes of classification into positive and negative classes

		Assigned Class	
		Positive	Negative
Actual Class	Positive	TP	FN
	Negative	FP	TN

►*Positive predictive value* = ►*Precision* = TP/(TP + FP)

►*Negative predictive value* = TN/(TN + FN)

Conjunctive Normal Form

Bernhard Pfahringer
University of Waikato, Hamilton, New Zealand

Conjunctive normal form (CNF) is an important normal form for propositional logic. A logic formula is in conjunctive normal form if it is a single conjunction of disjunctions of (possibly negated) literals. No more nesting and no other negations are allowed. Examples are:

a
$\neg b$
$a \wedge b$
$(a \vee \neg b) \wedge (c \vee d)$
$\neg a \wedge (b \vee \neg c \vee d) \wedge (a \vee \neg d)$

Any arbitrary formula in propositional logic can be transformed into conjunctive normal form by application of the laws of distribution, De Morgan's laws, and by removing double negations. It is important to note that this process can lead to exponentially larger formulas which implies that the process in the worst case runs in exponential time. An example for this behavior is the following formula given in ►disjunctive normal form (DNF), which is linear in the number of propositional variables in this form. When transformed into conjunctive normal form (CNF), its size is exponentially larger.

DNF: $(a_0 \wedge a_1) \vee (a_2 \wedge a_3) \vee \ldots \vee (a_{2n} \wedge a_{2n+1})$

CNF: $(a_0 \vee a_2 \vee \ldots \vee a_{2n}) \wedge (a_1 \vee a_2 \vee \ldots \vee a_{2n}) \wedge \ldots \wedge (a_1 \vee a_3 \vee \ldots \vee a_{2n+1})$

Recommended Reading

Russell, S., & Norvig, P. (2002). *Artificial intelligence: A modern approach* (p. 215). Prentice Hall

Connection Strength

►Weight

Connections Between Inductive Inference and Machine Learning

John Case[1], Sanjay Jain[2]
[1]University of Delaware, Newark, USA
[2]National University of Singapore, Singapore, Republic of Singapore

Definition

Inductive inference is a theoretical framework to model learning in the limit. Here we will discuss some results in inductive inference, which have relevance to machine learning community.

The mathematical/theoretical area called ►Inductive Inference, is also known as *computability theoretic learning* and *learning in the limit* (Jain, Osherson, Royer, & Sharma, 1999; Odifreddi, 1999) typically *but, as will be seen below, not always* involves a situation depicted in (1) just below.

$$\text{Data } d_0, d_1, d_2, \ldots \xrightarrow{\text{In}} M \xrightarrow{\text{Out}} \text{Programs } e_0, e_1, e_2, \ldots . \quad (1)$$

Let $\mathbb{N}$ = the set of nonnegative integers. Strings, including program strings, computer reals, and other data structures, inside computers, are finite bit strings and, hence, can be coded into $\mathbb{N}$. Therefore, mathematically at least, it is without loss of mathematical generality that we sometimes use the data type $\mathbb{N}$ where standard practice would use a different type.

In (1), $d_0, d_1, d_2, \ldots$ can be, e.g., the successive values of a function $f : \mathbb{N} \to \mathbb{N}$ or the elements of a (formal) language $L \subseteq \mathbb{N}$ in some order; M is a machine; the e_i's are from some hypothesis space of programs; and, for M's *successful* learning, later e_i's exactly or approximately compute the f or L.

Such learning is *off-line*: in successful cases, one comes away with programs for past and future data. For the related problem of *online extrapolation* of next values for a function f, *suitable* e_i's may *be* the values of $f(i)$'s based on having seen *strictly* prior values of f.

Detail

We will discuss the off-line case until we say otherwise. It is typical in applied machine learning to present to a learner whatever data one has and to obtain *one* corresponding program hopefully for predicting these data and future data. In inductive inference the case where only one program is output is called *one-shot* learning. More typically, in inductive inference, one allows for *mind-changes*, i.e., for a succession of output programs, as one receives successively more input data, with the later programs hopefully eventually being useful for predictions. Typically, one does not get success on one's first conjecture/output program, but rather, one may achieve success eventually, or, as it is said, *in the limit* after some sequence of trial and error. It is helpful at this juncture to present a problem for which this latter approach makes more sense than the one-shot approach.

We will consider some different criteria of successful learning of f or L by M. For example, **Ex**-style criteria

will require that all but finitely many of the e_i's are *syntactically* the same *and* do a reasonable job of computing the f or L. **Bc**-style criteria are more relaxed, more powerful, but less useful (Bārzdiņš, 1974; Case & Lynes, 1982; Case & Smith, 1983): they do not require almost all e_i's be the same syntactically.

Here is a well-known regression technique from, e.g., (Hildebrand, 1956), for exactly "curve-fitting" polynomials. It is the method involving calculating *forward differences*. We express it as a learning machine M_0 and illustrate with its being fed an *example* data sequence generated by a cubic polynomial

$$x^3 - 2x^2 + 2x + 3. \tag{2}$$

See (Hildebrand, 1956), for how to recover the polynomials themselves.

M_0, fed a finite data sequence of natural numbers, first looks for iterated forward differences to become (apparently) constant, then outputs a rule/program, which uses the (apparent) constant to extrapolate the data sequence for any desired prediction. For example, were M_0 given the data sequence in the top row of Table 1, it would calculate 6 to be the apparent constant after *three* differencings, so M_0 then outputs the following informal rule/program.

- To generate the level 0 sequence, at level 0, start with 3; at level 1, start with 1; at level 2, start with 2; add the apparent constant 6 from level 3 to get successive level 2 data items; add successive level 2 items to get successive level 1 data items; *finally*, add successive level 1 items to get as many successive level 0 data items as needed for prediction.

This program, eventually output by M_0 when its input the whole top row of Table 1, correctly predicts the elements of the cubic polynomial, on successive values in $\mathbb{N}$ – the whole sequence $3, 4, 7, 18, 43, 88, 159, \ldots$. Along the way, though, just after the first data point, M_0 thinks the apparent constant is 0; just after the second that it is 1; just after the third that it is 2; and only after more of the data points does it converge for this cubic polynomial to the apparent (and, on *this* example, actual) constant 6. In general, M_0, on a polynomial of degree m, *changes its mind* up to m times until converging to its final program (of course on $f(x) = 2^x$, M_0 never converges, and each level of forward differences is just the sequence f again.).

Connections Between Inductive Inference and Machine Learning. Table 1 Example Sequence and Its Iterated Forward Differences

Sequence:	3		4		7		18		43
1st Diffs:		1		3		11		25	
2nd Diffs:			2		8		14		
3rd Diffs:				6		6			

Hence, M_0 above **Ex**-learns, e.g., the integer polynomials $f : N \to N$, but it does not in general one-shot learn these polynomials – since the data alone do not disclose the degree of a generating polynomial.

In this entry we survey some results from inductive inference but with an eye to topics having something to say regarding or to applied machine learning. In some cases, the theoretical results lend mathematical support to preexisting empirical observations about the efficacy of known machine learning techniques. In other cases, the theoretical results provide some, typically abstract, suggestions for the machine learning practitioner. In some of these cases, some of the suggestions apparently pay off in others, intriguingly, we do not know yet.

Multi-Task or Context Sensitive Learning

In empirical, applied machine learning, *multitask* or *context sensitive learning* involves trying to learn Y by first (de Garis, 1990a, b; Fahlman, 1991; Thrun, 1996; Thrun & Sullivan, 1996; Tsung & Cottrell, 1989; Waibel, 1989a, b) *or* simultaneously (Caruana, 1993, 1996; Dietterich, Hild, & Bakiri, 1995; Matwin & Kubat, 1996; Mitchell, Caruana, Freitag, McDermott, & Zabowski, 1994; Pratt, Mostow, & Kamm, 1991; Sejnowski & Rosenberg, 1986; Bartlmae, Gutjahr, & Nakhaeizadeh, 1997) trying to learn also X – even in cases where there may be no inherent interest in learning X (see also ▶Transfer Learning). There is, in many cases, an apparent *empirical* advantage in doing this for some X, Y. It can happen that Y is not apparently or easily learnable by itself, but is learnable if one learns X first or simultaneously in some case X itself can be a sequence of tasks $X_1, \ldots, X_n$. Here the X_is may need to be learned sequentially or simultaneously to learn Y. For example, to teach a robot to drive

a car, it is useful to train it also to predict the center of the road markings (see, e.g., Baluja & Pomerleau, 1995; Caruana, 1996). For another example: an experimental system to predict the value of German *Daimler* stock performed better when it was modified to track simultaneously the German stock-index DAX (Bartlmae et al., 1997). The value of the Daimler stock here was the primary or target concept and the value of the DAX – a related concept – provided useful auxiliary context.

Angluin, Gasarch, and Smith (1989) shows *mathematically* that, in effect, there are (mathematical) learning scenarios for which it was *provable* that Y could not be learned without learning X *first* – and, in other scenarios (Angluin et al., 1989; Kinber, Smith, Velauthapillai, & Wiehagen, 1995), Y could not be learned without *simultaneously* learning X. These *mathematical* results provide a kind of evidence that the *empirical* observations as to the *apparent* usefulness of multitask or context sensitive learning *may* not be illusionary, luck, or a mere accident of happening to use some data sets but not others.

For illustration, here is a particularly simple theoretical example needing to be learned *simultaneously* and similar to examples in Angluin et al. (1989). Let $\mathcal{R}$ be the set of all computable functions mapping $\mathbb{N}$ to $\mathbb{N}$. We use numerical names in $\mathbb{N}$ for programs. Let

$$\mathcal{S} = \{(f,g) \in \mathcal{R} \times \mathcal{R} \mid f(0) \text{ is a program for } g \wedge g(0) \text{ is a program for } f\}. \quad (3)$$

We say (p,q) is a program for $(f,g) \in \mathcal{R} \times \mathcal{R}$ iff p is a program for f and q is a program for g.

Consider a machine M which, if, as in (1), M is fed $d_0, d_1, \ldots$, but *where* each d_i is $(f(i), g(i))$, then M outputs each $e_i = (g(0), f(0))$. Clearly, M one-shot learns $\mathcal{S}$. It can be easily shown that the component f's and g's *for* $(f,g) \in \mathcal{S}$ are not *separately* even **Bc**-learnable. It is important to note that, perhaps quite unlike real-world problems, the definition of this example $\mathcal{S}$ employs a simple self-referential coding trick: useful programs are coded into values of the functions at argument zero. A number of inductive inference results have been proved by means of (sometimes more complicated) self-referential coding tricks (see, e.g., Case, 1994). Bārzdiņš indirectly (see Zeugmann, 1986) provided a kind of informal robustness idea in his attempt to be rid of such coding tricks in inductive inference. More formally, Fulk (1990) considered a learnability result involving a witnessing class $\mathcal{C}$ of (tuples of) functions to be *robust* iff each computable scrambling of $\mathcal{C}$ also witnesses the learnability result (the allowed computable scramblers are the *general recursive operators* of (Rogers, 1967), but we omit the formal details herein.) Example: A simple shift scrambler converting each f to f', where $f'(x) = f(x+1)$, would eliminate the coding tricks just above – since the values of f at argument zero would be lost in this scrambling. Some inductive inference results hold robustly and some not (see, e.g., Fulk, 1990; Jain, 1999; Jain, Smith, & Wiehagen, 2001; Jain et al., 1999; Case, Jain, Ott, Sharma, & Stephan, 2000). Happily, the $\mathcal{S} \subseteq \mathcal{R} \times \mathcal{R}$ above (that is, learnable, but its components not) can be replaced by a more complicated class $\mathcal{S}'$ that *robustly* witnesses the same result. This is better *theoretical* evidence that the empirically noticed efficacy of multitask or context sensitive learning is not just an accident. It is residually important to note that (Jain et al., 2001) shows, though, that the computable scramblers can not get rid of more sophisticated coding tricks they called topological. $\mathcal{S}'$ mentioned just above turns out to *employ* this latter kind of coding trick. It is hypothesized in (Case et al., 2000) that nature likely employs some sophisticated coding tricks itself. For a separate informal argument about coding tricks of nature, see (Case, 1999). Ott and Stephan (2002) introduces a finite invariance constraint on top of robustness. This so-called hyperrobustness does destroy all coding tricks, and the result about the *theoretical* efficacy of multitask or context sensitive learning is *not* hyperrobust. However, hyperrobustness, perhaps, leaves unrealistically sparse structure.

Final note: Machine learning is an engineering endeavor. However, philosophers of science as well as practitioners in classical scientific disciplines should likely be considering the relevance of multitask or context sensitive inductive inference to their endeavors.

Special Cases of Inductive Logic Programming

In this section we discuss some *learning in the limit* results for *elementary formal systems (EFSs)* (Smullyan, 1961). Essentially, EFSs are programs in a string rewriting system. It is well known (Arikawa, Shinohara, & Yamamoto, 1992) that EFSs are essentially (pure) logic

programs over strings. Hence, the results have possible relevance for ►inductive logic programming (ILP) (Bratko & Muggleton, 1995; Lavrač & Džeroski, 1994; Mitchell, 1997; Muggleton & De Raedt, 1994).

First we will discuss some important special cases based on Angulin's *pattern languages* (Angluin, 1980).

A *pattern language* is (by definition) one generated by all the positive length substitution instances in a *pattern*, such as,

$$abXYcbbZXa \tag{4}$$

— where the variables (for substitutions) are depicted in upper case and the constants/terminals in lower case and are from, say, the alphabet {a,b,c}. Just below is an EFS or logic program based on this example pattern.

$$abXYcbbZXa \leftarrow . \tag{5}$$

It must be understood, though, that in (5) and in the next example EFS below, only positive length strings are allowed to be substituted for the variables.

Angluin (1980) showed the **Ex**-learnability of the class of pattern languages from positive data. For these results, in the paradigm of (1) above $d_0, d_1, d_2, \ldots$ is a listing or presentation of some formal language L over a finite nonempty alphabet and the e_i's are programs that generate languages. In particular, for Angluin's M, for L a pattern language, the e_i's are patterns, and, for each presentation of L, all but finitely many of the corresponding e_i's are the same *correct* pattern for L.

Much work has been done on the learnability of pattern languages, e.g., Salomaa (1994a, b); Case, Jain, Kaufmann, Sharma, and Stephan (2001), and on bounded finite unions thereof, e.g., Shinohara (1993); Wright (1989); Kilpeläinen, Mannila, and Ukkonen (1995); Brazma, Ukkonen, and Vilo (1996); Case, Jain, Lange, and Zeugmann (1999).

Regarding bounded finite unions of pattern languages: an *n-pattern language* is the union of the pattern languages for some n patterns $P_1, \ldots, P_n$. Each n-pattern language is also **Ex**-learnable from positive data (see Wright (1989)). An EFS or logic program corresponding to the n-patterns $P_1, \ldots, P_n$ and generating the corresponding n-pattern language is just below.

$$\begin{aligned} P_1 &\leftarrow . \\ &\vdots \\ P_n &\leftarrow . \end{aligned}$$

Pattern language learning algorithms have been successfully applied toward some problems in molecular biology, see, e.g., Shimozono et al. (1994), Shinohara and Arikawa (1995).

Lange and Wiehagen (1991) presents an interesting *iterative* (Wiehagen, 1976) algorithm learning the class of pattern languages – from positive data only and with polynomial time constraints. *Iterative learners* are **Ex**-learners for which each output depends only on its just prior output (if any) and the input data element currently seen. Their algorithm works in polynomial time (actually quadratic time) in the length of the latest data item and the previous hypothesis. Furthermore, the algorithm has a linear set of good examples, in the sense that if the input data contains these good examples, then the algorithm already converges to the correct hypothesis. The number of good examples needed is at most $|P| + 1$, where P is a pattern generating the data $d_0, d_1, d_2, \ldots$ for the language being learned. This algorithm may be useful in practice due to its fast run time, and being able to converge quickly, *if* enough good data is available early. Furthermore, due to iterativeness, it does not need to store previous data!

Zeugmann (1998) considers *total* learning time up to convergence of the algorithm just discussed in the just prior paragraph. Note that, for arbitrary presentations, $d_0, d_1, d_2, \ldots$, of a pattern language, this time can be unbounded. In the best case it is polynomial in the length of a generating pattern P, where $d_0, d_1, d_2, \ldots$ is based on using P to get good examples early – in fact the time taken in the best case is $\Theta(|P|^2 log_s(s + k))$, where P is the pattern, s is the alphabet size, and k is the number of variables in P. Much more interesting is the case of *average time* taken up to convergence. The probability distribution (called *uniform* by Zeugmann) considered is as follows. A variable X is replaced by a string w with probability $\frac{1}{(2s)^{|w|}}$ (i.e., all strings of length r together have probability 2^{-r}, and the distribution is uniform among strings of length r). Different variables are replaced independently of each other. In this case the average total time up to convergence is $O(2^k k^2 s|P|^2 log_s(ks))$. The main thing is that for average case on probabilistic data (as can be expected in real life, though not necessarily with this kind of uniform distribution), the algorithm converges pretty fast and computations are done efficiently.

A number of papers consider **Ex**-learning of EFSs (Krishna Rao, 1996; Krishna Rao, 2000, 2004, 2005; Krishna Rao & Sattar, 1998) including with various bounds on the number of mind-changes until syntactic convergence to correct programs (Jain & Sharma, 1997, 2002). The EFSs considered are patterns, n-patterns, those with a constant bound on the length of clauses, and some with constant bounds on search trees. The mind-change bounds are typically more dynamic than those given by constants: they involve counting down from finite representations (called *notations*) for infinite constructive ordinals. *An example* of this kind of bound: one can algorithmically, based on some input parameters, decide how many mind-changes will be allowed. In *other* examples, the decision as to how many mind-changes will be allowed can be algorithmically revised some constant number of times. It is possible that not yet created special cases of some of these algorithms could be made feasible enough for practice.

Learning Drifting Concepts

A drifting concept to be learned is one which is a moving target (see ►Concept Drift). In some machine learning applications, concept drift must be dealt with (Bartlett, Ben-David, & Kulkarni, 1996; Blum & Chalasani, 1992; Devaney & Ram, 1994; Freund & Mansour, 1997; Helmbold and Long, 1994; Kubat, 1992; Widmer & Kubat, 1996; Wrobel, 1994). An inductive inference contribution is (Case et al., 2001) in which it is shown, for online *extrapolation* by computable martingale betting strategies, upper bounds on the "speed" of the moving target that permit success at all. Here success is to make unbounded amounts of "money" betting on correctness of ones extrapolations. Here is an illustrative result from (Case et al., 2001). For the pattern languages considered in the previous section, only *positive* length strings of terminals can be substituted for a variable in an associated pattern. The (difficult to learn) *pattern languages with erasing* are just the languages obtained by also allowing the substitution of the empty string for variables in a pattern. For our example, we restrict the terminal alphabet to be {0,1}. With each pattern language with erasing L (over this terminal alphabet) we associate its characteristic function χ_L, which is 1 on terminal strings in L and 0 on those not in L. For ε denoting the empty string, and for the terminal strings in length-lexicographical order, $\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$, we would input a χ_L itself to a potential extrapolating machine as the bit string, $\chi_L(\varepsilon), \chi_L(0), \chi_L(1), \chi_L(00), \chi_L(01), \ldots$. Let $\mathcal{E}$ be the class of these characteristic functions. Pick a positive integer constant p. To model *drift with permanence* p, we imagine that a potential extrapolator for $\mathcal{E}$ receives successive bits from a member of $\mathcal{E}$ but keeps switching to the next bits of another, etc., *but* it must see at least p bits in a row of each member of $\mathcal{E}$ it sees before it can see the next bits of another. p is, then, a speed limit on drift. The result is that some suitably clever computable martingale betting strategy is successful at extrapolating $\mathcal{E}$ with drift permanence (speed limit on drift) of $p = 7$.

Behavioral Cloning

Kummer and Ott (1996); Case, Ott, Sharma, and Stephan (2002) studied learning in the limit of winning control strategies for *closed computable games*. These games nicely model *reactive* process-control problems. Included are such example process-control games as regulating temperature of a room to be in a desired interval, forever after no more than some *fixed* number of moves between the thermostat and processes disturbing the temperature (Roughly, *closed computable games* are those so that one can tell algorithmically when one has lost. A temperature control game that requires stability forever after some *undetermined* finite number of moves is *not* a closed computable game. For a more formal treatment, see Cenzer and Remmel (1992); Maler, Pnueli, and Sifakis (1995); Thomas (1995); Kummer and Ott (1996)).

In machine learning, there are cases where one wants to teach a machine some motor skill possessed by human experts and where these human experts do not have access to verbalizable knowledge about how they perform expertly. Piloting an aircraft or expert operation of a swinging shipyard crane provide examples, and machine learning employs, in these cases, ►*behavioral cloning*, which uses direct performance data from the experts (Bain & Sammut, 1999; Bratko, Urbančič, & Sammut, 1998; Šuc, 2003).

Case et al. (2002) studies the effects on learning in the limit *closed computable games* where the learning procedures also had access to the *behavioral performance* (but not the algorithms) of masters/experts at the

games. For example, it is showed that, in some cases, there is better performance cloning $n+1$ disparate masters over cloning only n. For a while it was not known *in machine learning* how to clone multiple experts even after Case et al. (2002) was known to some; however, independently of Case et al., 2002, and later, Dorian Šuc (Šuc, 2003) found a way to clone behaviorally more than one human expert simultaneously (for the free-swinging shipyard crane problem) – by having more than one level of feedback control, *and* he got enhanced performance from cloning the multiple experts!

Learning To Coordinate

Montagna and Osherson (1999) begins the study of learning in the limit to *coordinate* (digital) moves between at least two agents.

The machines of Montagna and Osherson (1999) are, in effect, general extrapolating devices (Montagna & Osherson, 1999; Case et al., 2005). Technically, and without loss of generality of the results, we restrict the moves of each coordinator to bits, i.e., zeros and ones. *Coordination* is achieved between two coordinators iff each, reacting to the bit sequence of the other, eventually (in the limit) matches it bit for bit. Montagna and Osherson (1999) gives an example of two people who show up in a park each day at one of noon (bit 0) or 6pm (bit 1); each *silently* watches the other's past behavior; and each *tries*, based on the past behavior of the other, to show up eventually exactly when the other shows up. If they manage it, they have learned to coordinate.

A *blind* coordinator is one that reacts only to the *presence* of a bit from another process, *not* to *which* bit the other process has played (Montagna and Osherson, 1999).

In Case et al. (2005) is developed and studied the notion of probabilistically correct algorithmic coordinators. Next is a sample of theorems to the effect that just a few random bits can enhance learning to coordinate.

Theorem 1 (Case et al., 2005) Suppose $0 \leq p < 1$. There exists a class of deterministic algorithmic coordinators $\mathcal{C}$ such that

(1) *No* deterministic algorithmic coordinator can coordinate with all of $\mathcal{C}$; and

(2) For k chosen so that $1 - 2^{-k} \geq p$, there exists a blind, *probabilistic* algorithmic coordinator **PM**, such that:
   (i) For each member of $\mathcal{C}$, **PM** can coordinate with with probability $1 - 2^{-k} \geq p$; and
   (ii) **PM** is *k-memory limited* in the sense of (Osherson, Stob, & Weinstein, 1986, P. 66); more specifically, **PM** needs to remember whether it is outputting one of its first k bits — *which are its only random bits* (e.g., for $p = \frac{255}{256}$, a mere $k = 8$ random bits suffice.).

Regarding possible eventual applicability: Maye, Hsieh, Sugihara, and Brembs (2007) cites finding deterministic chaos but *not* randomness in the behavior of *animals*. Hence, animals may not be exploiting random bits in learning anything, including to coordinate. However, one might build artifactual devices to exploit randomness, say, from radioactive decay, including, then, for enhancing learning to coordinate.

Learning Geometric Clustering

Case, Jain, Martin, Sharma, and Stephen (2006) showed that learnability in the limit of ▶*clustering*, with or without additional information, depends strongly on geometric constraints on the shape of the clusters. In this approach the hypothesis space of possible clusters is pre-given in each setting. It was hoped to obtain thereby insight into the difficulty of clustering when the clusters are restricted to preassigned *geometrically* defined classes.

This is interestingly complementary to the *conceptual clustering* approach (see, e.g., Mishra, Ron, & Swaminathan, 2004; Pitt & Reinke, 1988) where one restricts the possible clusters to have good "verbal" descriptions in some language.

Clustering of many of the geometric classes investigated was shown to *require* information *in addition* to a presentation, $d_0, d_1, d_2, \ldots$, of the set of points to be clustered. For example, for clusters as convex hulls of finitely many points in a rational vector space, clustering can be done – but with the number of clusters as additional information. Let $\mathcal{S}$ consist of all polygons including their interiors – in the rational two-dimensional plane *without intersections and degenerated angles* (Attention was restricted to spaces of rationals since: 1. computer

reals are rationals, 2. this avoids the uncountability of the set of reals, and 3. this avoids dealing with *un*computable real points.) The class $\mathcal{S}$ *can* be clustered – but with the number of vertices of the polygons of the clusters involved as additional information.

Correspondingly, then, it was shown that the class $\mathcal{S}'$ containing $\mathcal{S}$ *together with* all such polygons but with one hole (the nondegenerate differences of two members in $\mathcal{S}$) can*not* be clustered with the number of vertices as additional information, yet $\mathcal{S}'$ can be clustered with *area* as additional information – and this even in higher dimensions and with any number of holes (Case et al., 2006).

It remains to be seen if some forms of geometrically constrained clustering can be usefully complementary to, say, conceptually/verbally constrained clustering.

Insights for Limitations of Science

We briefly treat below in some problems regarding parsimonious, refutable, and consistent hypotheses.

It is common wisdom in science that one should fit parsimonious explanations, hypotheses, or programs to data. In machine learning, this has been successfully applied, e.g., (Wallace, 2005; Wallace & Dowe, 1999).

Curiously, though, there are many results in inductive inference in which we see sometimes severe *degradations* of learning power caused by demanding *parsimonious* predictive programs (see, e.g., Freivalds (1975); Kinber (1977); Chen (1982); Case, Jain, and Sharma (1996); Ambainis, Case, Jain, and Suraj (2004)).

It is an interesting problem to resolve the seeming, likely not actual contradiction between the just prior two paragraphs.

Popper's Refutability (Popper, 1962) asserts that hypotheses in science should be subject to refutation. Besides the well-known difficulties of Duhem–Quine (Harding, 1976) of knowing which component hypothesis to throw out when a compound hypothesis badly fails to make correct predictions, inductive inference theorems have provided very different difficulties. Case and Smith (1983) outlines cases of usefully *in*complete (hence wrong) hypothesis that cannot be refuted, and Case and Suraj (2007) (see also Case, 2007) provides cases of inductively inferable higher order hypothesis not totally subject to refutation in cases where ordinary hypotheses subject to full refutation cannot be inductively inferred.

While Duhem–Quine may impact machine learning eventually, it remains to be seen about the inductive inference results of the just prior paragraph.

Requiring ►inductive inference procedures always to output an hypothesis in various senses *consistent* with (e.g., not ignoring) the data on which that hypothesis is based seems like mere common sense. However, from Bārzdiņš (1974a); Blum and Blum (1975); Wiehagen (1976), Case, Jain, Stephan, and Wiehagen (2004) we see that strict adherence to various consistency principles can severely attenuate the learning power of inductive inference machines. Furthermore, interestingly, *even when inductive inference is polytime constrained*, we see similar counterintuitive results to the effect that a kind of consistency can strictly attenuate learning power (Wiehagen & Zeugmann, 1994).

A machine learning analog might be Breiman's bagging (Breiman, 1996) and random forests (Breiman, 2001), where data is purposely ignored. However, in these cases, the purpose of ignoring data is to avoid overfitting to noise.

It remains to be seen, whether, in applied machine learning involving cases of practically noiseless data, one can also obtain some advantage in ignoring some consistency principles. Again the potential lesson from inductive inference is abstract and provides only a hint of something to work out in real machine learning problems.

Cross References

►Behavioural Cloning
►Clustering
►Concept Drift
►Inductive Logic Programming
►Transfer Learning

Recommended Reading

Ambainis, A., Case, J., Jain, S., & Suraj, M. (2004). Parsimony hierarchies for inductive inference. *Journal of Symbolic Logic, 69*, 287–328.

Angluin, D., Gasarch, W., & Smith, C. (1989). Training sequences. *Theoretical Computer Science, 66*(3), 255–272.

Angluin, D. (1980). Finding patterns common to a set of strings. *Journal of Computer and System Sciences, 21*, 46–62.

Arikawa, S., Shinohara, T., & Yamamoto, A. (1992). Learning elementary formal systems. *Theoretical Computer Science, 95*, 97–113.

Bain, M., & Sammut, C. (1999). A framework for behavioural cloning. In K. Furakawa, S. Muggleton, & D. Michie (Eds.), *Machine intelligence 15*. Oxford: Oxford University Press.

Baluja, S., & Pomerleau, D. (1995). Using the representation in a neural network's hidden layer for task specific focus of attention. *Technical Report CMU-CS-95-143*, School of Computer Science, CMU, May 1995. Appears in Proceedings of the 1995 IJCAI.

Bartlett, P., Ben-David, S., & Kulkarni, S. (1996). Learning changing concepts by exploiting the structure of change. In *Proceedings of the ninth annual conference on computational learning theory*, Desenzano del Garda, Italy. New York: ACM Press.

Bartlmae, K., Gutjahr, S., & Nakhaeizadeh, G. (1997). Incorporating prior knowledge about financial markets through neural multitask learning. In *Proceedings of the fifth international conference on neural networks in the capital markets*.

Bārzdiņš, J. (1974a). Inductive inference of automata, functions and programs. In *Proceedings of the international congress of mathematicians*, Vancouver (pp. 771–776).

Bārzdiņš, J. (1974b). Two theorems on the limiting synthesis of functions. In *Theory of algorithms and programs* (Vol. 210, pp. 82–88). Latvian State University, Riga.

Blum, L., & Blum, M. (1975). Toward a mathematical theory of inductive inference. *Information and Control, 28*, 125–155.

Blum, A., & Chalasani, P. (1992). Learning switching concepts. In *Proceedings of the fifth annual conference on computational learning theory*, Pittsburgh, Pennsylvania, (pp. 231–242). New York: ACM Press.

Bratko, I., & Muggleton, S. (1995). Applications of inductive logic programming. *Communications of the ACM, 38*(11), 65–70.

Bratko, I., Urbančič, T., & Sammut, C. (1998). Behavioural cloning of control skill. In R. S. Michalski, I. Bratko, & M. Kubat (Eds.), *Machine learning and data mining: Methods and applications*, (pp. 335–351). New York: Wiley.

Brazma, A., Ukkonen, E., & Vilo, J. (1996). Discovering unbounded unions of regular pattern languages from positive examples. In *Proceedings of the seventh international symposium on algorithms and computation (ISAAC'96), Lecture notes in computer science*, (Vol. 1178, pp. 95–104), Berlin: Springer-Verlag.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32.

Caruana, R. (1993). Multitask connectionist learning. In *Proceedings of the 1993 connectionist models summer school* (pp. 372–379). NJ: Lawrence Erlbaum.

Caruana, R. (1996). Algorithms and applications for multitask learning. In *Proceedings 13th international conference on machine learning* (pp. 87–95). San Francisco, CA: Morgan Kaufmann.

Case, J. (1994). Infinitary self-reference in learning theory. *Journal of Experimental and Theoretical Artificial Intelligence, 6*, 3–16.

Case, J. (1999). The power of vacillation in language learning. *SIAM Journal on Computing, 28*(6), 1941–1969.

Case, J. (2007). Directions for computability theory beyond pure mathematical. In D. Gabbay, S. Goncharov, & M. Zakharyaschev (Eds.), *Mathematical problems from applied logic II. New logics for the XXIst century, International Mathematical Series*, (Vol. 5). New York: Springer.

Case, J., & Lynes, C. (1982). Machine inductive inference and language identification. In M. Nielsen & E. Schmidt, (Eds.), *Proceedings of the 9th International Colloquium on Automata, Languages and Programming, Lecture notes in computer science*, (Vol. 140, pp. 107–115). Berlin: Springer-Verlag.

Case, J., & Smith, C. (1983). Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science, 25*, 193–220.

Case, J., & Suraj, M. (2010). Weakened refutability for machine learning of higher order definitions, 2010. (Working paper for eventual journal submission).

Case, J., Jain, S., Kaufmann, S., Sharma, A., & Stephan, F. (2001). Predictive learning models for concept drift (Special Issue for ALT'98). *Theoretical Computer Science, 268*, 323–349.

Case, J., Jain, S., Lange, S., & Zeugmann, T. (1999). Incremental concept learning for bounded data mining. *Information and Computation, 152*, 74–110.

Case, J., Jain, S., Montagna, F., Simi, G., & Sorbi, A. (2005). On learning to coordinate: Random bits help, insightful normal forms, and competency isomorphisms (Special issue for selected learning theory papers from COLT'03, FOCS'03, and STOC'03). *Journal of Computer and System Sciences, 71*(3), 308–332.

Case, J., Jain, S., Martin, E., Sharma, A., & Stephan, F. (2006). Identifying clusters from positive data. *SIAM Journal on Computing, 36*(1), 28–55.

Case, J., Jain, S., Ott, M., Sharma, A., & Stephan, F. (2000). Robust learning aided by context (Special Issue for COLT'98). *Journal of Computer and System Sciences, 60*, 234–257.

Case, J., Jain, S., & Sharma, A. (1996). Machine induction without revolutionary changes in hypothesis size. *Information and Computation, 128*, 73–86.

Case, J., Jain, S., Stephan, F., & Wiehagen, R. (2004). Robust learning – rich and poor. *Journal of Computer and System Sciences, 69*(2), 123–165.

Case, J., Ott, M., Sharma, A., & Stephan, F. (2002). Learning to win process-control games watching gamemasters. *Information and Computation, 174*(1), 1–19.

Cenzer, D., & Remmel, J. (1992). Recursively presented games and strategies. *Mathematical Social Sciences, 24*, 117–139.

Chen, K. (1982). Tradeoffs in the inductive inference of nearly minimal size programs. *Information and Control, 52*, 68–86.

de Garis, H. (1990a). Genetic programming: Building nanobrains with genetically programmed neural network modules. In *IJCNN: International Joint Conference on Neural Networks*, (Vol. 3, pp. 511–516). Piscataway, NJ: IEEE Service Center.

de Garis, H. (1990b). Genetic programming: Modular neural evolution for Darwin machines. In M. Caudill (Ed.), *IJCNN-90-WASH DC; International joint conference on neural networks* (Vol. 1, pp. 194–197). Hillsdale, NJ: Lawrence Erlbaum Associates.

de Garis, H. (1991). Genetic programming: Building artificial nervous systems with genetically programmed neural network modules. In B. Soušek, & The IRIS group (Eds.), *Neural and intelligenct systems integeration: Fifth and sixth generation integerated reasoning information systems* (Chap. 8, pp. 207–234). New York: Wiley.

Devaney, M., & Ram, A. (1994). Dynamically adjusting concepts to accommodate changing contexts. In M. Kubat, G. Widmer (Eds.), *Proceedings of the ICML-96 Pre-conference workshop on learning in context-sensitive domains*, Bari, Italy (Journal submission).

Dietterich, T., Hild, H., & Bakiri, G. (1995). A comparison of ID3 and backpropogation for English text-tospeech mapping. *Machine Learning, 18*(1), 51–80.

Fahlman, S. (1991). The recurrent cascade-correlation architecture. In R. Lippmann, J. Moody, and D. Touretzky (Eds.), *Advances in neural information processing systems* (Vol. 3, pp. 190–196). San Mateo, CA: Morgan Kaufmann Publishers.

Freivalds, R. (1975). Minimal Gödel numbers and their identification in the limit. In *Lecture notes in computer science* (Vol. 32, pp. 219–225). Berlin: Springer-Verlag.

Freund, Y., & Mansour, Y. (1997). Learning under persistent drift. In S. Ben-David, (Ed.), *Proceedings of the third European conference on computational learning theory (EuroCOLT'97), Lecture notes in artificial intelligence*, (Vol. 1208, pp. 94–108). Berlin: Springer-Verlag.

Fulk, M. (1990). Robust separations in inductive inference. In *Proceedings of the 31st annual symposium on foundations of computer science* (pp. 405–410). St. Louis, Missouri. Washington, DC: IEEE Computer Society.

Harding, S. (Ed.). (1976). *Can theories be refuted? Essays on the Duhem-Quine thesis*. Dordrecht: Kluwer Academic Publishers.

Helmbold, D., & Long, P. (1994). Tracking drifting concepts by minimizing disagreements. *Machine Learning, 14*, 27–46.

Hildebrand, F. (1956). *Introduction to numerical analysis*. New York: McGraw-Hill.

Jain, S. (1999). Robust behaviorally correct learning. *Information and Computation, 153*(2), 238–248.

Jain, S., & Sharma, A. (1997). Elementary formal systems, intrinsic complexity, and procrastination. *Information and Computation, 132*, 65–84.

Jain, S., & Sharma, A. (2002). Mind change complexity of learning logic programs. *Theoretical Computer Science, 284*(1), 143–160.

Jain, S., Osherson, D., Royer, J., & Sharma, A. (1999). *Systems that learn: An introduction to learning theory* (2nd ed.). Cambridge, MA: MIT Press.

Jain, S., Smith, C., & Wiehagen, R. (2001). Robust learning is rich. *Journal of Computer and System Sciences, 62*(1), 178–212.

Kilpeläinen, P., Mannila, H., & Ukkonen, E. (1995). MDL learning of unions of simple pattern languages from positive examples. In P. Vitányi (Ed.), *Computational learning theory, second European conference, EuroCOLT'95*, *Lecture notes in artificial intelligence*, (Vol. 904, pp. 252–260). Berlin: Springer-Verlag.

Kinber, E. (1977). On a theory of inductive inference. In *Lecture notes in computer science* (Vol. 56, pp. 435–440). Berlin: Springer-Verlag.

Kinber, E., Smith, C., Velauthapillai, M., & Wiehagen, R. (1995). On learning multiple concepts in parallel. *Journal of Computer and System Sciences, 50*, 41–52.

Krishna Rao, M. (1996). A class of prolog programs inferable from positive data. In A. Arikawa & A. Sharma (Eds.), *Seventh international conference on algorithmic learning theory (ALT' 96)*, *Lecture notes in artificial intelligence* (Vol. 1160, pp. 272–284). Berlin: Springer-Verlag.

Krishna Rao, M. (2000). Some classes of prolog programs inferable from positive data (Special Issue for ALT'96). *Theoretical Computer Science A, 241*, 211–234.

Krishna Rao, M. (2004). Inductive inference of term rewriting systems from positive data. In S. Ben-David, J. Case, & A. Maruoka (Eds.), *Algorithmic learning theory: Fifteenth international conference (ALT' 2004)*, *Lecture notes in artificial intelligence* (Vol. 3244, pp. 69–82). Berlin: Springer-Verlag.

Krishna Rao, M. (2005). A class of prolog programs with non-linear outputs inferable from positive data. In S. Jain, H. U. Simon, & E. Tomita (Eds.), *Algorithmic learning theory: Sixteenth international conference (ALT' 2005)*, *Lecture notes in artificial intelligence*, (Vol. 3734, pp. 312–326). Berlin: Springer-Verlag.

Krishna Rao, M., & Sattar, A. (1998). Learning from entailment of logic programs with local variables. In M. Richter, C. Smith, R. Wiehagen, & T. Zeugmann (Eds.), *Ninth international conference on algorithmic learning theory (ALT' 98)*, *Lecture notes in artificial intelligence* (Vol. 1501, pp. 143–157). Berlin: Springer-Verlag.

Kubat, M. (1992). A machine learning based approach to load balancing in computer networks. *Cybernetics and Systems, 23*, 389–400.

Kummer, M., & Ott, M. (1996). Learning branches and learning to win closed recursive games. In *Proceedings of the ninth annual conference on computational learning theory*, Desenzano del Garda, Italy. New York: ACM Press.

Lange, S., & Wiehagen, R. (1991). Polynomial time inference of arbitrary pattern languages. *New Generation Computing, 8*, 361–370.

Lavrač, N., & Džeroski, S. (1994). *Inductive logic programming: Techniques and applications*. New York: Ellis Horwood.

Maler, O., Pnueli, A., & Sifakis, J. (1995). On the synthesis of discrete controllers for timed systems. In *Proceedings of the annual symposium on the theoretical aspects of computer science*, *LNCS* (Vol. 900, pp. 229–242). Berlin: Springer-Verlag.

Matwin, S., & Kubat, M. (1996). The role of context in concept learning. In M. Kubat & G. Widmer (Eds.), *Proceedings of the ICML-96 pre-conference workshop on learning in context-sensitive domains*, Bari, Italy, (pp. 1–5).

Maye, A., Hsieh, C., Sugihara, G., & Brembs, B. (2007). Order in spontaneous behavior. *PLoS One*, May, 2007. See: http://brembs.net/spontaneous/

Mishra, N., Ron, D., & Swaminathan, R. (2004). A new conceptual clustering framework. *Machine Learning, 56*(1–3), 115–151.

Mitchell, T. (1997). *Machine learning*. New York: McGraw Hill.

Mitchell, T., Caruana, R., Freitag, D., McDermott, J., & Zabowski, D. (1994). Experience with a learning, personal assistant. *Communications of the ACM, 37*, 80–91.

Montagna, F., & Osherson, D. (1999). Learning to coordinate: A recursion theoretic perspective. *Synthese, 118*, 363–382.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19/20*, 669–679.

Odifreddi, P. (1999). *Classical recursion theory* (Vol. II). Amsterdam: Elsivier.

Osherson, D., Stob, M., & Weinstein, S. (1986). Systems that learn: An introduction to learning theory for cognitive and computer scientists. Cambridge, MA: MIT Press.

Ott, M., & Stephan, F. (2002). Avoiding coding tricks by hyperrobust learning. *Theoretical Computer Science, 284*(1), 161–180.

Pitt, L., & Reinke, R. (1988). Criteria for polynomial-time (conceptual) clustering. *Machine Learning, 2*, 371–396.

Popper, K. (1992). Conjectures and refutations: The growth of scientific knowledge. New York: Basic Books.

Pratt, L., Mostow, J., & Kamm, C. (1991). Direct transfer of learned information among neural networks. In *Proceedings of the 9th national conference on artificial intelligence (AAAI-91)*, Anaheim, California. Menlo Park, CA: AAAI press.

Rogers, H. (1987). *Theory of recursive functions and effective computability*. New York: McGraw Hill (Reprinted, MIT Press, 1987).

Salomaa, A. (1994a). Patterns (The formal language theory column). *EATCS Bulletin, 54*, 46–62.

Salomaa, A. (1994b). Return to patterns (The formal language theory column). *EATCS Bulletin, 55*, 144–157.

Sejnowski, T., & Rosenberg, C. (1986). NETtalk: A parallel network that learns to read aloud. *Technical Report JHU-EECS-86-01*, Johns Hopkins University.

Shimozono, S., Shinohara, A., Shinohara, T., Miyano, S., Kuhara, S., & Arikawa, S. (1994). Knowledge acquisition from amino acid sequences by machine learning system BONSAI. *Transactions of Information Processing Society of Japan, 35*, 2009–2018.

Shinohara, T. (1983). Inferring unions of two pattern languages. *Bulletin of Informatics and Cybernetics, 20*, 83–88.

Shinohara, T., & Arikawa, A. (1995). Pattern inference. In K. P. Jantke & S. Lange (Eds.), *Algorithmic learning for knowledge-based systems*, Lecture notes in artificial intelligence (Vol. 961, pp. 259–291). Berlin: Springer-Verlag.

Smullyan, R. (1961). Theory of formal systems. In *Annals of Mathematics Studies* (Vol. 47). Princeton, NJ: Princeton University Press.

Šuc, D. (2003). Machine reconstruction of human control strategies. *Frontiers in artificial intelligence and applications* (Vol. 99). Amsterdam: IOS Press.

Thomas, W. (1995). On the synthesis of strategies in infinite games. In *Proceedings of the annual symposium on the theoretical aspects of computer science*, *LNCS* (Vol. 900, pp. 1–13). Berlin: Springer-Verlag.

Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems, 8*. San Mateo, CA: Morgan Kaufmann.

Thrun, S., & Sullivan, J. (1996). Discovering structure in multiple learning tasks: The TC algorithm. In *Proceedings of the thirteenth international conference on machine learning (ICML-96)* (pp. 489–497). San Francisco, CA: Morgan Kaufmann.

Tsung, F., & Cottrell, G. (1989). A sequential adder using recurrent networks. In *IJCNN-89-WASHINGTON DC: International joint conference on neural networks* June 18–22 (Vol. 2, pp. 133–139). Piscataway, NJ: IEEE Service Center.

Waibel, A. (1989a). Connectionist glue: Modular design of neural speech systems. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), *Proceedings of the 1988 connectionist models summer school* (pp. 417–425). San Mateo, CA: Morgan Kaufmann.

Waibel, A. (1989b). Consonant recognition by modular construction of large phonemic time-delay neural networks. In D. S. Touretzky (Ed.), *Advances in neural information processing systems I* (pp. 215–223). San Mateo, CA: Morgan Kaufmann.

Wallace, C. (2005). *Statistical and inductive inference by minimum message length*. (*Information Science and Statistics*). New York: Springer (Posthumously published).

Wallace, C., & Dowe, D. (1999). Minimum message length and kolmogorov complexity (Special Issue on Kolmogorov Complexity). *Computer Journal, 42*(4), 123–155. http://comjnl.oxfordjournals.org/cgi/reprint/42/4/270.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning, 23*, 69–101.

Wiehagen, R. (1976). Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. *Electronische Informationverarbeitung und Kybernetik, 12*, 93–99.

Wiehagen, R., & Zeugmann, T. (1994). Ignoring data may be the only way to learn efficiently. *Journal of Experimental and Theoretical Artificial Intelligence, 6*, 131–144.

Wright, K. (1989). Identification of unions of languages drawn from an identifiable class. In R. Rivest, D. Haussler, & M. Warmuth (Eds.), *Proceedings of the second annual workshop on computational learning theory*, Santa Cruz, California, (pp. 328–333). San Mateo, CA: Morgan Kaufmann Publishers.

Wrobel, S. (1994). Concept formation and knowledge revision. Dordrecht: Kluwer Academic Publishers.

Zeugmann, T. (1986). On Bārzdiņš' conjecture. In K. P. Jantke (Ed.), *Analogical and inductive inference, Proceedings of the international workshop*, *Lecture notes in computer science*, (Vol. 265, pp. 220–227). Berlin: Springer-Verlag.

Zeugmann, T. (1998). Lange and Wiehagen's pattern language learning algorithm: An average case analysis with respect to its total learning time. *Annals of Mathematics and Artificial Intelligence, 23*, 117–145.

Connectivity

►Topology of a Neural Network

Consensus Clustering

Synonyms

Clustering aggregation; Clustering ensembles

Definition

In Consensus Clustering we are given a set of n objects V, and a set of m clusterings $\{C_1, C_2, \ldots, C_m\}$ of the objects in V. The aim is to find a single clustering C that *disagrees* least with the input clusterings, that is, C minimizes

$$D(C) = \sum_{C_i} d(C, C_i),$$

for some metric d on clusterings of V. Meilă (2003) proposed the principled *variation of information* metric on clusterings, but it has been difficult to analyze theoretically. The Mirkin metric is the most widely used, in which $d(C, C')$ is the number of pairs of objects (u, v) that are clustered together in C and apart in C', or vice versa; it can be calculated in time $O(mn)$.

We can interpret each of the clusterings C_i in Consensus Clustering as evidence that pairs ought be put together or separated. That is, w^+_{uv} is the number of C_i in which $C_i[u] = C_i[v]$ and w^-_{uv} is the number of C_i in which $C_i[u] \neq C_i[v]$. It is clear that $w^+_{uv} + w^-_{uv} = m$ and

that Consensus clustering is an instance of Correlation clustering in which the w^-_{uv} weights obey the triangle inequality.

Constrained Clustering

Kiri L. Wagstaff
Pasadena, CA, USA

Definition

Constrained clustering is a semisupervised approach to ►clustering data while incorporating domain knowledge in the form of constraints. The constraints are usually expressed as pairwise statements indicating that two items must, or cannot, be placed into the same cluster. Constrained clustering algorithms may enforce every constraint in the solution, or they may use the constraints as guidance rather than hard requirements.

Motivation and Background

►Unsupervised learning operates without any domain-specific guidance or preexisting knowledge. Supervised learning requires that all training examples be associated with labels. Yet it is often the case that existing knowledge for a problem domain fits neither of these extremes. Semisupervised learning methods fill this gap by making use of both labeled and unlabeled data. Constrained clustering, a form of semisupervised learning, was developed to extend clustering algorithms to incorporate existing domain knowledge, when available. This knowledge may arise from labeled data or from more general rules about the concept to be learned.

One of the original motivating applications was noun phrase coreference resolution, in which noun phrases in a text must be clustered together to represent distinct entities (e.g., "Mr. Obama" and "the President" and "he", separate from "Sarah Palin" and "she" and "the Alaska governor"). This problem domain contains several natural rules for when noun phrases should (such as appositive phrases) or should not (such as a mismatch on gender) be clustered together. These rules can be translated into a collection of pairwise constraints on the data to be clustered.

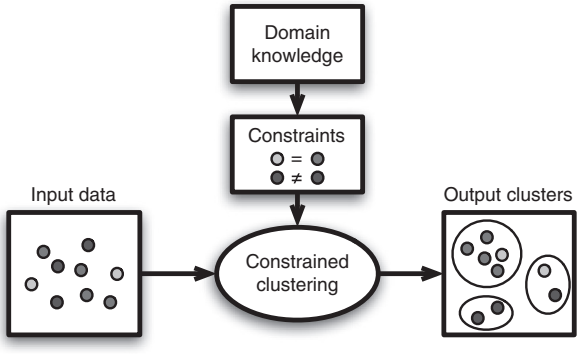
Constrained clustering algorithms have now been applied to a rich variety of domain areas, including hyperspectral image analysis, road lane divisions from GPS data, gene expression microarray analysis, video object identification, document clustering, and web search result grouping.

Structure of the Learning System

Constrained clustering arises out of existing work with unsupervised clustering algorithms. In this description, we focus on clustering algorithms that seek a partition of the data into disjoint clusters, using a distance or similarity measure to place similar items into the same cluster. Usually, the desired number of clusters, k, is specified as an input to the algorithm. The most common clustering algorithms are k-means (MacQueen, 1967) and expectation maximization or EM (Dempster, Laird, & Rubin, 1977) (Fig. 1).

A constrained clustering algorithm takes the same inputs as a regular (unsupervised) clustering algorithm and also accepts a set of pairwise constraints. Each constraint is a ►must-link or ►cannot-link constraint. The must-link constraints form an equivalence relation, which permits the inference of additional transitively implied must-links as well as additional entailed cannot-link constraints between items from distinct must-link cliques. Specifying a significant number of pairwise constraints might be tedious for large data sets, so often they may be generated from a manually labeled subset of the data or from domain-specific rules.

The algorithm may interpret the constraints as hard constraints that must be satisfied in the output or as soft preferences that can be violated, if necessary. The former approach was used in the first constrained clustering algorithms, COP-COBWEB (Wagstaff & Cardie,



Constrained Clustering. Figure 1. The constrained clustering algorithm takes in nine items and two pairwise constraints (one must-link and one cannot-link). The output clusters respect the specified constraints

2000) and COP-kmeans (Wagstaff, Cardie, Rogers, & Schroedl, 2001). COP-kmeans accommodates the constraints by restricting item assignments to exclude any constraint violations. If a solution that satisfies the constraints is not found, COP-kmeans terminates without a solution. Later, algorithms such as PCK-means and MPCK-means (Bilenko, Basu, & Mooney, 2004) permitted the violation of constraints when necessary by introducing a violation penalty. This is useful when the constraints may contain noise or internal inconsistencies, which are especially relevant in real-world domains. Constrained versions of other clustering algorithms such as EM (Shental, Bar-Hillel, Hertz, & Weinshall, 2004) and spectral clustering (Kamvar, Klein, & Manning, 2003) also exist. Penalized probabilistic clustering (PPC) is a modified version of EM that interprets the constraints as (soft) probabilistic priors on the relationships between items (Lu & Leen, 2005).

In addition to constraining the assignment of individual items, constraints can be used to learn a better distance metric for the problem at hand (Bar-Hillel, Hertz, Shental, & Weinshall, 2005; Klein, Kamvar, & Manning, 2002; Xing, Ng, Jordan, & Russell, 2003). Must-link constraints hint that the effective distance between those items should be low, while cannot-link constraints suggest that their pairwise distance should be high. Modifying the metric accordingly permits the subsequent application of a regular clustering algorithm, which need not explicitly work with the constraints at all. The MPCK-means algorithm fuses these approaches together, providing both constraint satisfaction and metric learning simultaneously (Basu, Bilenko, & Mooney, 2004; Bilenko et al., 2004).

More information about subsequent advances in constrained clustering algorithms, theory, and novel applications can be found in a compilation edited by Basu, Davidson, and Wagstaff (2008).

Programs and Data

The MPCK-means algorithm is available in a modified version of the Weka machine learning toolkit (Java) at http://www.cs.utexas.edu/users/ml/risc/code/.

Recommended Reading

Bar-Hillel, A., Hertz, T., Shental, N., & Weinshall, D. (2005). Learning a Mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research, 6*, 937–965.

Basu, S., Bilenko, M., & Mooney, R. J. (2004). A probabilistic framework for semi-supervised clustering. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 59–68). Seattle, WA.

Basu, S., Davidson, I., & Wagstaff, K. (Eds.). (2008). *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Boca Raton, FL: CRC Press.

Bilenko, M., Basu, S., & Mooney, R. J. (2004). Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the Twenty-first International Conference on Machine Learning* (pp. 11–18). Banff, AB, Canada.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, 39*(1), 1–38.

Kamvar, S., Klein, D., & Manning, C. D. (2003). Spectral learning. In *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 561–566). Acapulco, Mexico.

Klein, D., Kamvar, S. D., & Manning, C. D. (2002). From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 307–313). Sydney, Australia.

Lu, Z. & Leen, T. (2005). Semi-supervised learning with penalized probabilistic clustering. In *Advances in Neural Information Processing Systems* (Vol. 17, pp. 849–856). Cambridge, MA: MIT Press.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Symposium on Math, Statistics, and Probability* (Vol. 1, pp. 281–297). California: University of California Press.

Shental, N., Bar-Hillel, A., Hertz, T., & Weinshall, D. (2004). Computing Gaussian mixture models with EM using equivalence constraints. In *Advances in Neural Information Processing Systems* (Vol. 16, pp. 465–472). Cambridge, MA: MIT Press.

Wagstaff, K. & Cardie, C. (2000). Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 1103–1110). San Francisco: Morgan Kaufmann.

Wagstaff, K., Cardie, C., Rogers, S., & Schroedl, S. (2001). Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning* (pp. 577–584). San Francisco: Morgan Kaufmann.

Xing, E. P., Ng, A. Y., Jordan, M. I., & Russell, S. (2003). Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems* (Vol. 15, pp. 505–512). Cambridge, MA: MIT Press.

Constraint-Based Mining

Siegfried Nijssen
Katholieke Universiteit Leuven, Leuven, Belgium

Definition

Constraint-based mining is the research area studying the development of data mining algorithms that search

through a pattern or model space restricted by constraints. The term is usually used to refer to algorithms that search for patterns only. The most well-known instance of constraint-based mining is the mining of ►frequent patterns. Constraints are needed in pattern mining algorithms to increase the efficiency of the search and to reduce the number of patterns that are presented to the user, thus making knowledge discovery more effective and useful.

Motivation and Background

Constraint-based pattern mining is a generalization of frequent itemset mining. For an introduction to frequent itemset mining, see ►Frequent Patterns. A constraint-based mining problem is specified by providing the following elements:

- A database $\mathcal{D}$, usually consisting of independent transactions (or instances)
- A ►hypothesis space $\mathcal{L}$ of patterns
- A constraint $q(\theta, \mathcal{D})$ expressing criteria that a pattern θ in the hypothesis space should fulfill on the database

The general constraint-based mining problem is to find the set

$$\mathrm{Th}(\mathcal{D}, \mathcal{L}, q) = \{\theta \in \mathcal{L} | q(\theta, \mathcal{D}) = \mathrm{true}\}.$$

Alternative problem settings are obtained by making different choices for $\mathcal{D}$, $\mathcal{L}$ and q. For instance,

- If the database and hypothesis space consist of itemsets, and the constraint checks if the support of a pattern exceeds a predefined threshold in data, the frequent itemset mining problem is obtained (see ►Frequent Patterns)
- If the database and the hypothesis space consist of graphs or trees instead of itemsets, a graph mining or a tree mining problem is obtained. For more information about these topics, see ►Graph Mining and ►Tree Mining
- Additional syntactic constraints can be imposed

An overview of important types of constraints is given below.

One can generalize the constraint-based mining problem beyond pattern mining. Also models, such as ►Decision Trees, could be seen as languages of interest. In the broadest sense, topics such as ►Constrained Clustering, ►Cost-Sensitive Learning, and even learning ►Support Vector Machines (SVMs) may be seen as constraint-based mining problems. However, it is currently not common to categorize these topics as *constraint-based mining*; in practice, the term refers to constraint-based *pattern* mining.

From the perspective of constraint-based mining, the knowledge discovery process can be seen as a process in which a user repeatedly specifies constraints for data mining algorithms; the data mining system is a solver that finds patterns or models that satisfy the constraints.

This approach to data mining is very similar to querying relational databases. Whereas relational databases are usually queried using operations such as projections, selections, and joins, in the constraint-based mining framework data is queried to find patterns or models that satisfy constraints that cannot be expressed in these primitives. A database which supports constraint-based mining queries, stores patterns and models, and allows later reuse of patterns and models, is sometimes also called an *inductive database* (Imielinski & Mannila, 1996).

Structure of the Learning System

Constraints

Frequent pattern mining algorithms can be generalized along several dimensions.

One way to generalize pattern mining algorithms is to allow them to deal with arbitrary ►coverage relations, which determine when a pattern matches a transaction in the data. In the example of mining itemsets, the subset relation determines the coverage relation. The coverage relation is at the basis of constraints such as minimum support; an alternative coverage relation would be the superset relation.

From the coverage relation follows a generality relationship. A pattern θ_1 is defined to be more specific than a pattern θ_2 (denoted by $\theta_1 > \theta_2$) if any transaction that is covered by θ_1 is also covered by θ_2 (see ►Generalization). In frequent itemset mining, itemset I_1 is more general than itemset I_2 if and only $I_1 \subseteq I_2$.

Generalization and coverage relationships can be used to identify the following types of constraints.

Monotonic and Anti-Monotonic Constraints An essential property which is exploited in ▶frequent pattern mining, is that all subsets of a frequent pattern are also frequent. This is a property that can be generalized:

- A constraint is called *monotonic* if any generalization of a pattern that satisfies the constraint, also satisfies the constraint
- A constraint is called *anti-monotonic* if any specialization of a pattern that satisfies the constraint, also satisfies the constraint

In some publications, the definitions of monotonic and anti-monotonic are used reversely.

The following are examples of monotonic constraints:

- Minimum support
- Syntactic constraints, for instance: a constraint that requires that patterns specializing a given pattern x are excluded a constraint requiring patterns to be small given a definition of pattern size
- Disjunctions or conjunctions of monotonic constraints
- Negations of anti-monotonic constraints

The following are examples of anti-monotonic constraints:

- Maximum support
- Syntactic constraints, for instance, a constraint that requires that patterns generalizing a given pattern x are excluded
- Disjunctions or conjunctions of anti-monotonic constraints
- Negations of monotonic constraints

Succinct Constraints Constraints that can be pushed in the mining process by adapting the pattern space or data, are called succinct constraints. An example of a succinct constraint is the monotonic constraint that an itemset should contain the item A. This constraint could be dealt with by deleting all transactions that do not contain A. For any frequent itemset found in the new dataset, it is now known that the item A can be added to it.

Convertible Constraints Some constraints that are not monotonic, can still be *convertible* monotonic (Pei & Han, 2002). A constraint is convertible monotonic if for every pattern θ one least general generalization θ' can be identified such that if θ satisfies the constraint, then θ' also satisfies the constraint. An example of a convertible constraint is a maximum average cost constraint. Assume that every item in an itemset has a cost as defined by a function $c(i)$. The constraint $c(I) = \sum_{i \in I} c(i)/|I| \leq maxcost$ is not monotonic. However, for every itemset I with $c(I) \leq maxcost$, if an item i is removed with $c(i) = \max_{i \in I} c(i)$, an itemset with $c(I - \{i\}) \leq c(I) \leq maxcost$ is obtained.

Maximum average cost has the desirable property that no access to the data is needed to identify the generalization that should satisfy the constraints. If it is not possible to identify the necessary least general generalization before accessing the data, the convertible constraint is also sometimes called weak (anti-)monotone (Zhu, Yan, Han, & Yu, 2007).

Boundable Constraints Constraints on non-monotonic measures for which a monotonic bound exist, are called boundable. An example of such a constraint is a minimum accuracy constraint in a database with binary class labels. Assume that every itemset is interpreted as a rule **if** I **then** 1 **else** 2 (thus, class label 1 is predicted if a transaction contains itemset I, or class label 2 otherwise; see ▶Supervised Descriptive Rule Discovery). A minimum accuracy constraint can be formalized by the formula $(\mathrm{fr}(I, D_1) + |D_2| - \mathrm{fr}(I, D_2))/|D| \geq minacc$, where D_k is the database containing only the examples labeled with class label k. It can be derived from this that

$$\mathrm{fr}(I, D_1) \geq |D|minacc - |D_2| + \mathrm{fr}(I, D_2) \geq |D|minacc - |D_2|.$$

In other words, if a high accuracy is desirable, a minimum number of examples of class 1 is required to be covered, and a minimum frequency constraint can thus be derived. Therefore, minimum support can be used as a bound for minimum accuracy.

The principle of deriving bounds for non-monotonic measures can be applied widely (Bayardo, Agrawal, & Gunopulos, 1999; Morishita & Sese, 2000).

Borders If constraints are not restrictive enough, the number of patterns can be huge. Ignoring statistics about patterns such as their exact frequency, the set of patterns can be represented more compactly only by

listing the patterns in the *border*(*s*) (Mannila & Toivonen, 1997), similar to the idea of ►version spaces. An example of a border is the set of maximal frequent itemsets (see ►Frequent Patterns). Borders can be computed for other types of both monotonic and anti-monotonic constraints as well. There are several complications compared to the simple frequent pattern mining setting:

- If there is an anti-monotonic constraint, such as maximum support, not only is it needed to compute a border for the most specific elements in the set (S-Set), but also a border for the least general elements in the set (G-Set)
- If the formula is a disjunction of conjunctions, the result of a query becomes a union of version spaces, which is called a multi-dimensional version space (see Fig. 1) (De Raedt, Jaeger, Lee, & Mannila, 2002); the G-Set of one version space may be more general than the G-Set of another version space

Both the S-Set and the G-Set can be represented by listing elements just within the version space (the positive border), or elements just outside the version space (the negative border). For instance, the positive border of the G-Set consists of those patterns which are part of the version space, and for which no generalizations exist which are part of the version space.

Similarly, there may exist several representations of multi-dimensional version spaces; optimizing the representation of multi-dimensional version spaces is analogous to optimizing queries in relational databases (De Raedt et al., 2002).

Borders form a *condensed representations*, that is, they compactly represent the solution space; see ►Frequent Patterns.

Algorithms For many of the constraints specified in the previous section specialized algorithms have been developed in combination with specific hypothesis spaces. It is beyond the scope of this chapter to discuss all these algorithms; only the most common ideas are provided here.

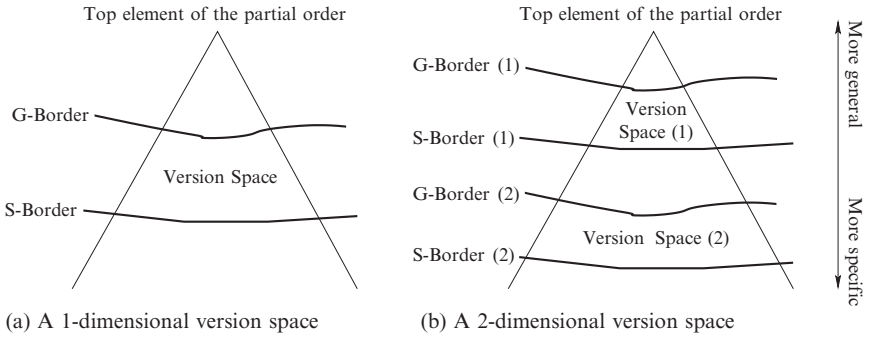
The main idea is that ►Apriori can easily be updated to deal with general monotonic constraints in arbitrary hypothesis spaces. The concept of a specialization ►refinement operator is essential to operate on other hypothesis spaces than itemsets. A specialization operator $\rho(\theta)$ computes a set of specializations in the hypothesis space for a given input pattern. In pattern mining, this operator should have the following properties:

- Completeness: every pattern in the hypothesis space should be reachable by repeated application of the refinement operator starting from the most general pattern in the hypothesis space
- Nonredundancy: every pattern in the hypothesis space should be reachable in only one way starting from the most general pattern in the hypothesis space

In itemset mining, optimal refinement is usually obtained by first ordering the items (for instance, alphabetically, or by frequency), and then adding items that are higher in the chosen order to a set than the items already in the set. For instance, for the itemset $\{A, C\}$, the specialization operator returns $\rho(\{A, C\}) = \{\{A, C, D\}, \{A, C, E\}\}$, assuming that the domain of items $\{A, B, C, D, E\}$ is considered. Other refinement operators are needed while dealing with other hypothesis spaces, such as in ►graph mining.

The search in Apriori proceeds ►breadth-first. Each level, the specialization operator is applied on patterns satisfying the monotonic constraints to generate candidates for the next level. For every new candidate it is checked whether its generalizations satisfy the monotonic constraints. To create a set of generalizations, a generalization refinement operator can be used. In frequent itemset mining, usually single items are removed from the itemset to generate generalizations.

More changes are required to deal with *anti-monotonic* constraints. A simple way of dealing with both monotonic and anti-monotonic constraints is to first compute all patterns that satisfy the monotonic constraints, and then to prune the patterns that fail to satisfy the anti-monotonic constraints. More challenging is to "push" anti-monotonic constraints in the mining process. An observation which is often exploited is that generalizations of patterns that do not satisfy the anti-monotonic constraint need not be considered. Well-known strategies are:



(a) A 1-dimensional version space (b) A 2-dimensional version space

Constraint-Based Mining. Figure 1. Version spaces

- In a breadth-first setting: traverse the lattice in reverse order for monotonic constraints, after the patterns have been determined satisfying the anti-monotonic constraints (De Raedt et al., 2002)
- In a depth-first setting: during the search for patterns, try to guess the largest pattern that can still be reached, and prune a branch in the search if the pattern does not satisfy the monotonic constraint on this pattern (Bucila, Gehrke, Kifer, & White, 2003; Kifer, Gehrke, Bucila, & White, 2003)

It is beyond the scope of this chapter to discuss how to deal with other types of constraints; however, it should be pointed out that not all combinations of constraints and hypothesis spaces have been studied; it is not obvious whether all constraints can be pushed usefully in a pattern search for any hypothesis space, for instance, when boundable constraints in more complex hypothesis spaces (such as graphs) are involved. Research in this area is ongoing.

Cross References

►Constrained Clustering
►Frequent Pattern Mining
►Graph Mining
►Tree Mining

Recommended Reading

Bayardo, R. J., Jr., Agrawal, R., & Gunopulos, D. (1999). Constraint-based rule mining in large, dense databases. In *Proceedings of the 15th international conference on data engineering (ICDE)* (pp. 188–197). Sydney, Australia.

Bucila, C., Gehrke, J., Kifer, D., & White, W. M. (2003). DualMiner: A dual-pruning algorithm for itemsets with constraints. *Data Mining and Knowledge Discovery, 7*(3), 241–272.

De Raedt, L., Jaeger, M., Lee, S. D., & Mannila, H. (2002). A theory of inductive query answering (extended abstract). In *Proceedings of the second IEEE international conference on data mining (ICDM)* (pp. 123–130). Los Alamitos, CA: IEEE Press.

Imielinski, T., & Mannila, H. (1996). A database perspective on knowledge discovery. *Communications of the ACM, 39*, 58–64.

Kifer, D., Gehrke, J., Bucila, C., & White, W. M. (2003). How to quickly find a witness. In *Proceedings of the twenty-second ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems* (pp. 272–283). San Diego, CA: ACM Press.

Mannila, H., & Toivonen, H. (1997). Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery, 1*(3), 241–258.

Morishita, S., & Sese, J. (2000). Traversing itemset lattices with statistical metric pruning. In *Proceedings of the nineteenth ACM SIGACT-SIGMOD-SIGART symposium on database systems (PODS)* (pp. 226–236). San Diego, CA: ACM Press.

Pei, J., & Han, J. (2002). Constrained frequent pattern mining: A pattern-growth view. *SIGKDD Explorations, 4*(1), 31–39.

Zhu, F., Yan, X., Han, J., & Yu, P. S. (2007). gPrune: A constraint pushing framework for graph pattern mining. In *Proceedings of the sixth Pacific-Asia conference on knowledge discovery and data mining (PAKDD). Lecture notes in computer science* (Vol. 4426, pp. 388–400). Berlin: Springer.

Constructive Induction

Constructive induction is any form of ►induction that generates new descriptors not present in the input data (Dietterich & Michalski, 1983).

Recommended Reading

Dietterich, T. G., & Michalski, R. S. (1983). A comparative review of selected methods for learning from examples. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.). *Machine learning: An artificial intelligence approach*, pp. 41–81. Tioga.

Content Match

►Text Mining for Advertising

Content-Based Filtering

Synonyms
Content-based recommending

Definition
Content-based filtering is prevalent in ►Information Retrieval, where the text and multimedia content of documents is used to select documents relevant to a user's query. In the context this refers to content-based recommenders, that provide recommendations by comparing representations of content describing an item to representations of content that interests a user.

Content-Based Recommending

►Content-Based Filtering

Context-Sensitive Learning

►Concept Drift

Contextual Advertising

►Text Mining for Advertising

Continual Learning

Synonyms
Life-Long Learning

Definition
A learning system that can continue adding new data without the need to ever stop or freeze the updating. Usually continual learning requires incremental and ►online learning as a component, but not every incremental learning system has the ability to achieve continual learning, i.e., the learning may deterioate after some time.

Cross References
►Cumulative Learning

Continuous Attribute

A **continuous attribute** can assume all values on the number line within the value range. See ►Attribute and ►Measurement Scales.

Contrast Set Mining

Definition
Contrast set mining is an area of ►supervised descriptive rule induction. The contrast set mining problem is defined as finding contrast sets, which are conjunctions of attributes and values that differ meaningfully in their distributions across groups (Bay & Pazzani, 2001). In this context, groups are the properties of interest.

Recommended Reading
Bay, S.D., & Pazzani, M. J. (2001). Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery, 5*(3), 213–246.

Cooperative Coevolution

►Compositional Coevolution

Co-Reference Resolution

►Entity Resolution

Correlation Clustering

Anthony Wirth
The University of Melbourne, Victoria, Australia

Synonyms

Clustering with advice; Clustering with constraints; Clustering with qualitative information; Clustering with side information

Definition

In its rawest form, *correlation clustering* is graph optimization problem. Consider a ▶clustering C to be a mapping from the elements to be clustered, V, to the set $\{1, \ldots, |V|\}$, so that u and v are in the same cluster if and only if $C[u] = C[v]$. *Given* a collection of items in which each pair (u, v) has two weights w^+_{uv} and w^-_{uv}, we must *find* a clustering C that minimizes

$$\sum_{C[u]=C[v]} w^-_{uv} + \sum_{C[u]\neq C[v]} w^+_{uv}\,, \tag{1}$$

or, equivalently, maximizes

$$\sum_{C[u]=C[v]} w^+_{uv} + \sum_{C[u]\neq C[v]} w^-_{uv}\,. \tag{2}$$

Note that although w^+_{uv} and w^-_{uv} may be thought of as positive and negative evidence towards coassociation, the actual weights are nonnegative.

Motivation and Background

The notion of *clustering with advice*, that is nonmetric-driven relations between items, had been studied in other communities (Ferligoj & Batagelj, 1982) prior to its appearance in theoretical computer science. Traditional clustering problems, such as k-median and k-center, assume that there is some type of distance measure (metric) on the data items, and often specify the number of clusters that should be formed. In the clustering with advice framework, however, the number of clusters to be built need not be specified in advance: it can be an outcome of the objective function. Furthermore, instead of, or in addition to, a distance function, we are given advice as to which pairs of items are similar. The two weights w^+_{uv} and w^-_{uv} correspond to external advice about whether the pair should be clustered together or separately. Bansal, Blum, and Chawla (2002) introduced the problem to the theoretical computer science and machine-learning communities. They were motivated by database consistency problems, in which the same entity appeared in different forms in various databases. Given a collection of such records from multiple databases, the aim is to cluster together the records that appear to correspond to the same entity. From this viewpoint, the log odds ratio from some classifier,

$$\log\left(\frac{\Pr(\text{same})}{\Pr(\text{different})}\right),$$

corresponds to a label w_{uv} for the pair. In many applications only one of the + and − weights for the pair is nonzero, that is

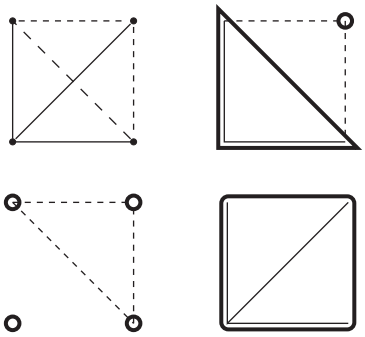
$$(w^+_{uv}, w^-_{uv}) = \begin{cases} (w_{uv}, 0) & \text{for } w_{uv} \geq 0 \\ (0, -w_{uv}) & \text{for } w_{uv} \leq 0\,. \end{cases}$$

In addition, if every pair has weight ±1, then the instance is called *complete*, otherwise it is referred to as *general*. Demaine, Emanuel, Fiat, and Immorlica (2006) suggest the following motivation. Suppose we have a set of guests at a party. Each guest has preferences for whom they would like to sit with, and for whom they would like to avoid. We must group the guests into tables in a way that enhances the amicability of the party.

The notion of producing good clusterings when given inconsistent advice first appeared in the work of Ben-Dor, Shamir, and Yakhini (1999). A canonical example of inconsistent advice is this: items u and v are similar, items v and y are similar, but u and y are dissimilar. It is impossible to find a clustering that satisfies all the advice. Figure 1 shows a very simple example of inconsistent advice. In addition, although Correlation clustering is an NP-hard problem, recent algorithms for clustering with advice *guarantee* that their solutions are only a specified factor worse than the optimal: that is, they are *approximation algorithms*.

Theory

In setting out the correlation clustering framework, Bansal et al. (2002) noted that the following algorithm



Correlation Clustering. Figure 1. Top left is a toy *clustering with advice* example showing three similar pairs (*solid edges*) and three dissimilar pairs (*dashed edges*). Bottom left is a clustering solution for this example with four singleton clusters, while bottom right has one cluster. Top right is a partitioning into two clusters that appears to best respect the advice

produces a 2-approximation for the maximization problem:

- If the total of the positive weights exceeds the total of the negative weights then, place all the items in a single cluster; otherwise, make each item a singleton cluster.

They then showed that complete instances are NP-hard to optimize, and how to minimize the penalty (1) with a constant factor approximation. The constant for this combinatorial algorithm was rather large. The algorithm relied heavily on the completeness of the instance; it iteratively *cleans* clusters until every cluster is δ-*clean*. That is, for each item at most a fraction δ $(0 < \delta < 1)$ of the other items in its cluster have a negative relation with it, and at most δ outside its cluster a positive relation. Bansal et al. also demonstrated that the minimization problem on general instances is APX-hard: there is some constant, larger than 1, below which approximation is NP-hard. Finally, they provided a polynomial time approximation scheme (PTAS) for maximizing (2) in complete instances.

The constant factor for minimizing (1) on complete instances was improved to 4 by Charikar, Guruswami, and Wirth (2003). They employed a region-growing type procedure to round the solution of a linear programming relaxation of the problem:

$$\begin{aligned} &\text{minimize} \\ &\quad \sum_{ij} w_{ij}^{+} \cdot x_{ij} + w_{ij}^{-} \cdot (1 - x_{ij}) \\ &\text{subject to} \qquad\qquad (3) \\ &\quad x_{ik} \le x_{ij} + x_{jk} \quad \text{for all } i, j, k \\ &\quad x_{ij} \in [0, 1] \quad \text{for all } i, j \end{aligned}$$

In this setting, $x_{ij} = 1$ implies i and j's separation, while $x_{ij} = 0$ implies coclustering, with values in between representing partial evidence. In practice solving this linear program is very slow and has huge memory demands (Bertolacci & Wirth, 2007). Charikar et al. also showed that this version of problem is APX-hard.

For the maximization problem (2), they showed that instances with general weights were APX-hard and provided a rounding of the following semidefinite program (SDP) that yields a 0.7664 factor approximation algorithm.

$$\begin{aligned} &\text{maximize} \\ &\quad \sum_{+(ij)} w_{ij}(v_i \cdot v_j) + \sum_{-(ij)} w_{ij}(1 - v_i \cdot v_j) \\ &\text{subject to} \qquad\qquad (4) \\ &\quad v_i \cdot v_i = 1 \quad \text{for all } i \\ &\quad v_i \cdot v_j \ge 0 \quad \text{for all } i, j \end{aligned}$$

In this case we interpret $v_i \cdot v_j = 1$ as evidence that i and j are in the same cluster, but $v_i \cdot v_j = 0$ as evidence toward separation.

Emanuel and Fiat (2003) extended the work of Bansal et al. by drawing a link between Correlation Clustering and the Minimum Multicut problem. This reduction to Multicut provided an $O(\log n)$ approximation algorithm for minimizing general instances of Correlation Clustering. Interestingly, Emanuel and Fiat also showed that there was reduction in the opposite direction: an optimal solution to Correlation Clustering induced an optimal solution to Minimum Multicut.

Demaine and Immorlica (2003) also drew the link from Correlation Clustering to Minimum multicut and its $O(\log n)$ approximation algorithm. In addition, they described an $O(r^3)$-approximation algorithm for graphs that exclude the complete bipartite graph $K_{r,r}$ as a minor.

Swamy (2004), using the same SDP (4) as Charikar et al., but different rounding techniques, showed how to maximize (2) within factor 0.7666 in general instances.

The factor 4 approximation for minimization (1) of complete instances was lowered to 2.5 by Ailon, Charikar, and Newman (2005). Using the *distances* obtained by solving the linear program (3), they repeat the following steps:

- form a cluster around random item i by including each (unclustered) j with probability $1 - x_{ij}$; set the cluster aside.

Since solving the linear program is highly resource hungry, Ailon et al. provided a combinatorial alternative: add j to i's cluster if $w_{ij}^{+} > w_{ij}^{-}$. Not only is this algorithm very fast, it is actually a factor 3 approximation.

Recently, Tan (2007) has shown that the $79/80 + \epsilon$ inapproximability for maximizing (2) on general weighted graphs extends to general unweighted graphs.

A further variant in the Correlation Clustering family of problems is the maximization of (2)–(1), known as *maximizing correlation*. Charikar and Wirth (2004) proved an $\Omega(1/\log n)$ approximation for the general problem of maximizing

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j, \quad \text{s.t. } x_i \in \{-1,1\} \text{ for all } i, \tag{5}$$

for a matrix A with null diagonal entries, by rounding the canonical SDP relaxation. This effectively maximized correlation with the requirement that two clusters be formed; it was not hard to extend this to general instances. The gap between the vector SDP solution and the integral solution to maximizing the quadratic program (5) was in fact shown to be $\Theta(1/\log n)$ in general (Alon, Makarychev, Makarychev, & Naor, 2006). However, in other instances such as those with a bounded number of nonzero weights for each item, a constant factor approximation was possible. Arora, Berger, Hazan, Kindler, and Safra (2005) went further and showed that it is *quasi*-NP-hard to approximate the maximization to a factor better than $\Omega(1/\log^{\gamma} n)$ for some $\gamma > 0$.

Shamir, Sharan, and Tsur (2004) showed that ▶Cluster Editing and p-Cluster Editing, in which p clusters must be formed, are NP-complete (for $p \geq 2$). Gramm, Guo, Hüffner, and Niedermeier (2004) took an innovative approach to solving the Clustering Editing problem exactly. They had previously produced an $O(2.27^k + n^3)$ time hand-made search tree algorithm, where k is the number of edges that need to be modified. This "awkward and error-prone work" was then replaced with a computer program that itself designed a search tree algorithm, involving automated case analysis, that ran in $O(1.92^k + n^3)$ time.

Kulis, Basu, Dhillon, and Mooney (2005) unify various forms of clustering, correlation clustering, spectral clustering, and clustering with constraints in their kernel-based approach to k-means. In this, they have a general objective function that includes penalties for violating pairwise constraints and for having points spread far apart from their cluster centers, where the *spread* is measured in some high-dimensional space.

Applications

The work of Demaine and Immorlica (2003) on Correlation Clustering was closely linked with that of Bejerano et al. on Location Area Planning. This problem is concerned with the allocation of cells in a cellular network to clusters known as *location areas*. There are costs associated with traffic between the location areas (cuts between clusters) and with the size of clusters themselves (related to paging phones within individual cells). These costs drive the clustering solution in opposite directions, on top of which there are constraints on cells that must (or cannot) be in the same cluster. The authors show that the same $O(\log n)$ region-growing algorithm for minimizing Correlation Clustering and Multicut applies to Location Area Planning.

Correlation clustering has been directly applied to the coreference problem in natural language processing and other instances in which there are multiple references to the same object (Daume, 2006; McCallum & Wellner, 2005). Assuming some sort of undirected graphical model, such as a Conditional Random Field, algorithms for correlation clustering are used to partition a graph whose edge weights corresponding to log-potentials between node pairs. The machine learning community has applied some of the algorithms for Correlation clustering to problems such as email clustering and image segmentation. With similar applications in mind, Finley and Joachims (2005) explore the idea of adapting the pairwise input information to fit example

clusterings given by a user. Their objective function is the same as Correlation Clustering (2), but their main tool is the ►Support Vector Machine.

There has been considerable interest in the ►consensus clustering problem, which is an excellent application of Correlation clustering techniques. Gionis, Mannila, and Tsaparas (2005) note several sources of motivation for the Consensus Clustering; these include identifying the correct number of clusters and improving clustering robustness. They adapt Charikar et al.'s region-growing algorithm to create a three-approximation that performs reasonably well in practice, though not as well as local search techniques. Gionis et al. also suggest using sampling as a tool for handling large data sets. Bertolacci and Wirth (2007) extended this study by implementing Ailon et al.'s algorithms with sampling, and therefore a variety of ways of developing a full clustering from the clustering of the sample. They noted that LP-based methods performed best, but placed a significant strain on resources.

Applications of Clustering with Advice

The ►*k*-means clustering algorithm is perhaps the most-used clustering technique: Wagstaff et al. incorporated constraints into a highly cited *k*-means variant called COP-KMEANS. They applied this algorithm to the task of identifying lanes of traffic based on input GPS data.

In the constrained-clustering framework, the constraints are usually assumed to be consistent (noncontradictory) and hard. In addition to the usual must- and cannot-link constraints, Davidson and Ravi (2005) added constraints enforcing various requirements on the distances between points in particular clusters. They analyzed the computational feasibility of the problem of establishing the (in) feasibility of a set of constraints, for various constraint types. Their constrained *k*-means algorithms were used to help a robot discover objects in a scene.

Recommended Reading

Ailon, N., Charikar, M., & Newman, A. (2005). Aggregating inconsistent information: Ranking and clustering. In *Proceedings of the Thirty-Seventh ACM Symposium on the Theory of Computing* (pp. 684–693). New York: ACM Press.

Alon, N., Makarychev, K., Makarychev, Y., & Naor, A. (2006). Quadratic forms on graphs. *Inventiones Mathematicae, 163*(3), 499–522.

Arora, S., Berger, E., Hazan, E., Kindler, G., & Safra, S. (2005). On non-approximability for quadratic programs. In *Proceedings of Forty-Sixth Symposium on Foundations of Computer Science.* (pp. 206–215). Washington DC: IEEE Computer Society.

Bansal, N., Blum, A., & Chawla, S. (2002). Correlation clustering. In *Correlation clustering* (pp. 238–247). Washington, DC: IEEE Computer Society.

Ben-Dor, A., Shamir, R., & Yakhini, Z. (1999). Clustering gene expression patterns. *Journal of Computational Biology, 6*, 281–297.

Bertolacci, M., & Wirth, A. (2007). Are approximation algorithms for consensus clustering worthwhile? In *Proceedings of Seventh SIAM International Conference on Data Mining.* (pp. 437–442). Philadelphia: SIAM.

Charikar, M., Guruswami, V., & Wirth, A. (2003). Clustering with qualitative information. In *Proceedings of forty fourth FOCS* (pp. 524–533).

Charikar, M., & Wirth, A. (2004). Maximizing quadratic programs: Extending Grothendieck's inequality. In *Proceedings of forty fifth FOCS* (pp. 54–60).

Daume, H. (2006). Practical structured learning techniques for natural language processing. PhD thesis, University of Southern California.

Davidson, I., & Ravi, S. (2005). Clustering with constraints: Feasibility issues and the *k*-means algorithm. In *Proceedings of Fifth SIAM International Conference on Data Mining.*

Demaine, E., Emanuel, D., Fiat, A., & Immorlica, N. (2006). Correlation clustering in general weighted graphs. *Theoretical Computer Science, 361*(2), 172–187.

Demaine, E., & Immorlica, N. (2003). Correlation clustering with partial information. In *Proceedings of Sixth Workshop on Approximation Algorithms for Combinatorial Optimization Problems.* (pp. 1–13).

Emanuel, D., & Fiat, A. (2003). Correlation clustering – minimizing disagreements on arbitrary weighted graphs. In *Proceedings of Eleventh European Symposium on Algorithms* (pp. 208–220).

Ferligoj, A., & Batagelj, V. (1982). Clustering with relational constraint. *Psychometrika, 47*(4), 413–426.

Finley, T., & Joachims, T. (2005). Supervised clustering with support vector machines. In *Proceedings of Twenty-Second International Conference on Machine Learning.*

Gionis, A., Mannila, H., & Tsaparas, P. (2005). Clustering aggregation. In *Proceedings of Twenty-First International Conference on Data Engineering.* To appear.

Gramm, J., Guo, J., Hüffner, F., & Niedermeier, R. (2004). Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica, 39*(4), 321–347.

Kulis, B., Basu, S., Dhillon, I., & Mooney, R. (2005). Semi-supervised graph clustering: A kernel approach. In *Proceedings of Twenty-Second International Conference on Machine Learning* (pp. 457–464).

McCallum, A., & Wellner, B. (2005). Conditional models of identity uncertainty with application to noun coreference. In L. Saul,

Y. Weiss, & L. Bottou, (Eds.), *Advances in neural information processing systems 17* (pp. 905–912). Cambridge, MA: MIT Press.

Meilă, M. (2003). Comparing clusterings by the variation of information. In *Proceedings of Sixteenth Conference on Learning Theory* (pp. 173–187).

Shamir, R., Sharan, R., & Tsur, D. (2004). Cluster graph modification problems. *Discrete Applied Mathematics, 144*, 173–182.

Swamy, C. (2004). Correlation Clustering: Maximizing agreements via semidefinite programming. In *Proceedings of Fifteenth ACM-SIAM Symposium on Discrete Algorithms* (pp. 519–520).

Tan, J. (2007). A Note on the inapproximability of correlation clustering. Technical Report 0704.2092, eprint arXiv, 2007.

Correlation-Based Learning

►Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Cost

In ►*Markov decision processes*, negative *rewards* are often expressed as *costs*. A reward of $-x$ is expressed as a cost of x. In ►supervised learning, *cost* is used as a synonym for ►loss.

Cross References

►Loss

Cost Function

►Loss Function

Cost-Sensitive Classification

►Cost-Sensitive Learning

Cost-Sensitive Learning

Charles X. Ling, Victor S. Sheng
The University of Western Ontario, Canada

Synonyms

Cost-sensitive classification; Learning with different classification costs

Definition

Cost-Sensitive Learning is a type of learning that takes the misclassification costs (and possibly other types of cost) into consideration. The goal of this type of learning is to minimize the total cost. The key difference between cost-sensitive learning and cost-insensitive learning is that cost-sensitive learning treats different misclassifications differently. That is, the cost for labeling a positive example as negative can be different from the cost for labeling a negative example as positive. Cost-insensitive learning does not take misclassification costs into consideration.

Motivation and Background

Classification is an important task in inductive learning and machine learning. A classifier, trained from a set of training examples with class labels, can then be used to predict the class labels of new examples. The class label is usually discrete and finite. Many effective classification algorithms have been developed, such as ►naïve Bayes, ►decision trees, ►neural networks, and ►support vector machines. However, most classification algorithms seek to minimize the error rate: the percentage of the incorrect prediction of class labels. They ignore the difference between types of misclassification errors. In particular, they implicitly assume that all misclassification errors have equal cost.

In many real-world applications, this assumption is not true. The differences between different misclassification errors can be quite large. For example, in medical diagnosis of a certain cancer (where having cancer is regarded as the positive class, and non-cancer (healthy) as negative), misdiagnosing a cancer patient as healthy (the patient is actually positive but is classified as negative; thus it is also called "false negative") is much more serious (thus expensive) than a false-positive error. The patient could lose his/her life because of a delay in correct diagnosis and treatment. Similarly, if carrying a bomb is positive, then it is much more expensive to miss a terrorist who carries a bomb onto a flight than searching an innocent person.

Cost-sensitive learning takes costs, such as the misclassification cost, into consideration. Turney (2000) provides a comprehensive survey of a large variety of different types of costs in data mining and machine

learning, including misclassification costs, data acquisition cost (instance costs and attribute costs), ▶active learning costs, computation cost, human–computer interaction cost, and so on. The misclassification cost is singled out as the most important cost, and it has received the most attention in recent years.

Theory

The theory of cost-sensitive learning (Elkan, 2001; Zadrozny and Elkan, 2001) describes how the misclassification cost plays its essential role in various cost-sensitive learning algorithms.

Without loss of generality, binary classification is assumed (i.e., positive and negative class) in this paper. In cost-sensitive learning, the costs of false positive (actual negative but predicted as positive; denoted as *FP*), false negative (*FN*), true positive (*TP*), and true negative (*TN*) can be given in a cost matrix, as shown in Table 1. In the table, the notation $C(i,j)$ is also used to represent the misclassification cost of classifying an instance from its actual class j into the predicted class i (1 is used for positive, and 0 for negative). These misclassification cost values can be given by domain experts, or learned via other approaches. In cost-sensitive learning, it is usually assumed that such a cost matrix is given and known. For multiple classes, the cost matrix can be easily extended by adding more rows and more columns.

Note that $C(i,i)$ (*TP* and *TN*) is usually regarded as the "benefit" (i.e., negated cost) when an instance is predicted correctly. In addition, cost-sensitive learning is often used to deal with datasets with very imbalanced class distributions (see ▶Class Imbalance Problem) (Japkowicz & Stephen, 2002). Usually (and without loss of generality), the minority or rare class is regarded as the positive class, and it is often more expensive to misclassify an actual positive example into negative, than an actual negative example into positive. That is, the value of $FN = C(0,1)$ is usually larger than that of $FP = C(1,0)$. This is true for the cancer example mentioned earlier (cancer patients are usually rare in the population, but predicting an actual cancer patient as negative is usually very costly) and the bomb example (terrorists are rare).

Given the cost matrix, an example should be classified into the class that has the minimum expected cost. This is the minimum expected cost principle. The expected cost $R(i|x)$ of classifying an instance x into class i (by a classifier) can be expressed as:

$$R(i|x) = \sum_j P(j|x)\, C(j,i), \tag{1}$$

where $P(j|x)$ is the probability estimation of classifying an instance into class j. That is, the classifier will classify an instance x into positive class if and only if:

$$P(0|x)\, C(1,0) + P(1|x)\, C(1,1) \le P(0|x)\, C(0,0) + P(1|x)\, C(0,1)$$

This is equivalent to:

$$P(0|x)\,(C(1,0) - C(0,0)) \le P(1|x)\,(C(0,1) - C(1,1))$$

Thus, the decision (of classifying an example into positive) will not be changed if a constant is added into a column of the original cost matrix. Thus, the original cost matrix can always be converted to a simpler one by subtracting $C(0,0)$to the first column, and $C(1,1)$ to the second column. After such conversion, the simpler cost matrix is shown in Table 2. Thus, any given cost-matrix can be converted to one with $C(0,0) = C(1,1) = 0$. (Here it is assumed that the misclassification cost is the same for

Cost-Sensitive Learning. Table 1 An Example of Cost Matrix for Binary Classification

	Actual negative	Actual positive
Predict negative	$C(0,0)$, or *TP*	$C(0,1)$, or *FN*
Predict positive	$C(1,0)$, or FP	$C(1,1)$, or TP

Cost-Sensitive Learning. Table 2 A Simpler Cost Matrix with an Equivalent Optimal Classification

	True negative	True positive
Predict negative	0	$C(0,1) - C(1,1)$
Predict positive	$C(1,0) - C(0,0)$	0

all examples. This property is a special case of the one discussed in Elkan (2001).) In the rest of the paper, it will be assumed that $C(0,0) = C(1,1) = 0$. Under this assumption, the classifier will classify an instance x into positive class if and only if:

$$P(0|x)\,C(1,0) \le P(1|x)\,C(0,1)$$

As $P(0|x) = 1 - P(1|x)$, a threshold p^* can be obtained for the classifier to classify an instance x into positive if $P(1|x) \ge p^*$, where

$$p^* = \frac{C(1,0)}{C(1,0) + C(0,1)}. \quad (2)$$

Thus, if a cost-insensitive classifier can produce a posterior probability estimation $p(1|x)$ for each test example x, one can make the classifier cost-sensitive by simply choosing the classification threshold according to (2), and classify any example to be positive whenever $P(1|x) \ge p^*$. This is what several cost-sensitive meta-learning algorithms, such as *Relabeling*, are based on (see later for details). However, some cost-insensitive classifiers, such as C4.5, may not be able to produce accurate probability estimation; they return a class label without a probability estimate. *Empirical Thresholding* (Sheng & Ling, 2006) does not require accurate estimation of probabilities – an accurate ranking is sufficient. It simply uses ►cross-validation to search for the best probability value p^* to use as a threshold.

Traditional cost-insensitive classifiers are designed to predict the class in terms of a default, fixed threshold of 0.5. Elkan (2001) shows that one can "rebalance" the original training examples by sampling, such that the classifiers with the 0.5 threshold is equivalent to the classifiers with the p^* threshold as in (2), in order to achieve cost-sensitivity. The rebalance is done as follows. If all positive examples (as they are assumed as the rare class) are kept, then the number of negative examples should be multiplied by $C(1,0)/C(0,1) = FP/FN$. Note that as usually $FP < FN$, the multiple is less than 1. This is, thus, often called "under-sampling the majority class." This is also equivalent to "proportional sampling," where positive and negative examples are sampled by the ratio of:

$$p(1)\,FN : p(0)\,FP \quad (3)$$

where $p(1)$ and $p(0)$ are the prior probability of the positive and negative examples in the original training set. That is, the prior probabilities and the costs are interchangeable: doubling $p(1)$ has the same effect as doubling FN, or halving FP (Drummond & Holte, 2000). Most sampling meta-learning methods, such as costing (Zadrozny, Langford, & Abe, 2003), are based on (3) above (see later for details).

Almost all meta-learning approaches are either based on (2) or (3) for the thresholding- and sampling-based meta-learning methods, respectively, to be discussed in the next section.

Structure of Learning System

Broadly speaking, cost-sensitive learning can be categorized into two categories. The first one is to design classifiers that are cost-sensitive in themselves.They are called the direct method. Examples of direct cost-sensitive learning are ICET (Turney, 1995) and cost-sensitive decision tree (Drummond & Holte, 2000; Ling, Yang, Wang, & Zhang, 2004). The other category is to design a "wrapper" that converts any existing cost-insensitive (or cost-blind) classifiers into cost-sensitive ones. The wrapper method is also called cost-sensitive meta-learning method, and it can be further categorized into thresholding and sampling. Here is a hierarchy of the cost-sensitive learning and some typical methods. This paper will focus on cost-sensitive meta-learning that considers the misclassification cost only.

Cost-Sensitive learning

- Direct methods
  - ICET (Turney, 1995)
  - Cost-sensitive decision trees (Drummond & Holte, 2000; Ling et al., 2004)
- Meta-learning
  - Thresholding
    - MetaCost (Domingos, 1999)
    - CostSensitiveClassifier (CSC in short) (Witten & Frank, 2005)
    - Cost-sensitive naïve Bayes (Chai, Deng, Yang, & Ling, 2004)
    - Empirical Thresholding (ET in short) (Sheng & Ling, 2006)
  - Sampling
    - Costing (Zadrozny et al., 2003)
    - Weighting (Ting, 1998)

Direct Cost-Sensitive Learning

The main idea of building a direct cost-sensitive learning algorithm is to directly introduce and utilize misclassification costs into the learning algorithms. There are several works on direct cost-sensitive learning algorithms, such as ICET (Turney, 1995) and cost-sensitive decision trees (Ling et al., 2004).

ICET (Turney, 1995) incorporates misclassification costs in the fitness function of genetic algorithms. On the other hand, cost-sensitive decision tree (Ling et al., 2004), called CSTree here, uses the misclassification costs directly in its tree building process. That is, instead of minimizing entropy in attribute selection as in C4.5, CSTree selects the best attribute by the expected total cost reduction. That is, an attribute is selected as a root of the (sub) tree if it minimizes the total misclassification cost.

Note that as both ICET and CSTree directly take costs into model building, they can also take easily attribute costs (and perhaps other costs) directly into consideration, while meta cost-sensitive learning algorithms generally cannot.

Drummond and Holte (2000) investigate the cost-sensitivity of the four commonly used attribute selection criteria of decision tree learning: accuracy, Gini, entropy, and DKM. They claim that the sensitivity of cost is highest with the accuracy, followed by Gini, entropy, and DKM.

Cost-Sensitive Meta-Learning

Cost-sensitive meta-learning converts existing cost-insensitive classifiers into cost-sensitive ones without modifying them. Thus, it can be regarded as a middleware component that preprocesses the training data, or post-processes the output, from the cost-insensitive learning algorithms.

Cost-sensitive meta-learning can be further classified into two main categories: *thresholding* and *sampling*, based on (2) and (3) respectively, as discussed in the theory section.

Thresholding uses (2) as a threshold to classify examples into positive or negative if the cost-insensitive classifiers can produce probability estimations. *MetaCost* (Domingos, 1999) is a *thresholding* method. It first uses bagging on decision trees to obtain reliable probability estimations of training examples, relabels the classes of training examples according to (2), and then uses the relabeled training instances to build a cost-insensitive classifier. *CSC* (Witten & Frank, 2005) also uses (2) to predict the class of test instances. More specifically, *CSC* uses a cost-insensitive algorithm to obtain the probability estimations $P(j \mid x)$ of each test instance. (CSC is a meta-learning method and can be applied to any classifiers.) Then it uses (2) to predict the class label of the test examples. Cost-sensitive naïve Bayes (Chai et al., 2004) uses (2) to classify test examples based on the posterior probability produced by the naïve Bayes.

As seen, all *thresholding*-based meta-learning methods rely on accurate probability estimations of $p(1 \mid x)$ for the test example x. To achieve this, Zadrozny and Elkan (2001) propose several methods to improve the calibration of probability estimates. *ET* (Empirical Thresholding) (Sheng and Ling, 2006) is a thresholding-based meta-learning method. It does not require accurate estimation of probabilities – an accurate ranking is sufficient. *ET* simply uses cross-validation to search the best probability from the training instances as the threshold, and uses the searched threshold to predict the class label of test instances.

On the other hand, *sampling* first modifies the class distribution of the training data according to (3), and then applies cost-insensitive classifiers on the sampled data directly. There is no need for the classifiers to produce probability estimations, as long as they can classify positive or negative examples accurately. Zadrozny et al. (2003) show that proportional sampling with replacement produces duplicated cases in the training, which in turn produces overfitting in model building. Instead, Zadrozny et al. (2003) proposes to use "rejection sampling" to avoid duplication. More specifically, each instance in the original training set is drawn once, and accepted into the sample with the accepting probability $C(j,i)/Z$, where $C(j,i)$ is the misclassification cost of class i, and Z is an arbitrary constant such that $Z \geq max\ C(j,i)$. When $Z = max_{ij}C(j,i)$, this is equivalent to keeping all examples of the rare class, and sampling the majority class without replacement according to $C(1,0)/C(0,1)$ – in accordance with (3). Bagging is applied after rejection sampling to improve the results further. The resulting method is called *Costing*.

Weighting (Ting, 1998) can also be viewed as a sampling method. It assigns a normalized weight to each instance according to the misclassification costs

specified in (3). That is, examples of the rare class (which carries a higher misclassification cost) are assigned, proportionally, high weights. Examples with high weights can be viewed as example duplication – thus oversampling. *Weighting* then induces cost-sensitivity by integrating the instances' weights directly into C4.5, as C4.5 can take example weights directly in the entropy calculation. It works whenever the original cost-insensitive classifiers can accept example weights directly. (Thus, it can be said that *Weighting* is a semi meta-learning method.) In addition, *Weighting* does not rely on bagging as *Costing* does, as it "utilizes" all examples in the training set.

Recommended Reading

Chai, X., Deng, L., Yang, Q., & Ling, C. X. (2004). Test-cost sensitive naïve Bayesian classification. In *Proceedings of the fourth IEEE international conference on data mining*. Brighton: IEEE Computer Society Press.

Domingos, P. (1999). MetaCost: A general method for making classifiers cost-sensitive. In *Proceedings of the fifth international conference on knowledge discovery and data mining, San Diego* (pp. 155–164). New York: ACM.

Drummond, C., & Holte, R. (2000). Exploiting the cost (in)sensitivity of decision tree splitting criteria. In *Proceedings of the 17th international conference on machine learning* (pp. 239–246).

Elkan, C. (2001). The foundations of cost-sensitive learning. *In Proceedings of the 17th international joint conference of artificial intelligence* (pp. 973–978). Seattle: Morgan Kaufmann.

Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5), 429–450.

Ling, C. X., Yang, Q., Wang, J., & Zhang, S. (2004). Decision trees with minimal costs. In*Proceedings of 2004 international conference on machine learning (ICML'2004)*.

Sheng, V. S., & Ling, C. X. (2006). Thresholding for making classifiers cost-sensitive. In *Proceedings of the 21st national conference on artificial intelligence* (pp. 476–481), 16–20 July 2006, Boston, Massachusetts.

Ting, K. M. (1998). Inducing cost-sensitive trees via instance weighting. In *Proceedings of the second European symposium on principles of data mining and knowledge discovery* (pp. 23–26). Heidelberg: Springer.

Turney, P. D. (1995). Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2, 369–409.

Turney, P. D. (2000). Types of cost in inductive concept learning. In *Proceedings of the workshop on cost-sensitive learning at the 17th international conference on machine learning, Stanford University, California*.

Witten, I. H., & Frank, E. (2005). *Data mining – Practical machine learning tools and techniques with Java implementations*. San Francisco: Morgan Kaufmann.

Zadrozny, B., & Elkan, C. (2001). Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the seventh international conference on knowledge discovery and data mining* (pp. 204–213).

Zadrozny, B., Langford, J., & Abe, N. (2003). Cost-sensitive learning by cost-proportionate instance weighting. In *Proceedings of the third International conference on data mining.*

Cost-to-Go Function Approximation

►Value Function Approximation

Covariance Matrix

Xinhua Zhang
Australian National University,
Canberra, Australia

Definition

It is convenient to define a covariance matrix by using multi-variate random variables (*mrv*): $\mathbf{X} = (X_1, \ldots, X_d)^\top$. For univariate random variables X_i and X_j, their covariance is defined as:

$$\mathrm{Cov}(X_i, X_j) = \mathbb{E}\left[(X_i - \mu_i)(X_j - \mu_j)\right],$$

where μ_i is the mean of X_i : $\mu_i = \mathbb{E}[X_i]$. As a special case, when $i = j$, then we get the variance of X_i, $\mathrm{Var}(X_i) = \mathrm{Cov}(X_i, X_i)$. Now in the setting of *mrv*, assuming that each component random variable X_i has finite variance under its marginal distribution, the covariance matrix $\mathrm{Cov}(\mathbf{X}, \mathbf{X})$ can be defined as a d-by-d matrix whose (i,j)-th entry is the covariance:

$$\left(\mathrm{Cov}(\mathbf{X}, \mathbf{X})\right)_{ij} = \mathrm{Cov}(X_i, X_j) = \mathbb{E}\left[(X_i - \mu_i)(X_j - \mu_j)\right].$$

And its inverse is also called *precision matrix*.

Motivation and Background

The covariance between two univariate random variables measures how much they change together, and as a special case, the covariance of a random variable with itself is exactly its variance. It is important to note that covariance is an unnormalized measure of the correlation between the random variables.

As a generalization to multi-variate random variables $\mathbf{X} = (X_1, \ldots, X_d)^\top$, the covariance matrix is a

d-by-d matrix whose (i,j)-th component is the covariance between X_i and X_j.

In many applications, it is important to characterize the relations between a set of factors, hence the covariance matrix plays an important role in practice, especially in machine learning.

Theory

It is easy to rewrite the element-wise definition into the matrix form:

$$\mathrm{Cov}(\mathbf{X},\mathbf{X}) = \mathbb{E}\left[(\mathbf{X}-\mathbb{E}[\mathbf{X}])(\mathbf{X}-\mathbb{E}[\mathbf{X}])^\top\right], \quad (1)$$

which naturally generalizes the variance of univariate random variables: $\mathrm{Var}(X) = \mathbb{E}[(X-\mathbb{E}[X])^2]$.

Moreover, it is also straightforward to extend the covariance of a single *mrv* $\mathbf{X}$ to two *mrv* 's $\mathbf{X}$ (d dimensional) and $\mathbf{y}$ (s dimensional), under the name *cross-covariance*. It quantifies how much the component random variables in $\mathbf{X}$ and $\mathbf{y}$ change together. The cross-covariance matrix is defined as a $d \times s$ matrix $\mathrm{Cov}(\mathbf{X},\mathbf{y})$ whose (i,j)-th entry is

$$\begin{aligned}(\mathrm{Cov}(\mathbf{X},\mathbf{y}))_{ij} &= \mathrm{Cov}(X_i, Y_j)\\ &= \mathbb{E}\left[(X_i-\mathbb{E}[X_i])(Y_j-\mathbb{E}[Y_j])\right].\end{aligned}$$

$\mathrm{Cov}(\mathbf{X},\mathbf{y})$ can also be written in the matrix form as

$$\mathrm{Cov}(\mathbf{X},\mathbf{y}) = \mathbb{E}\left[(\mathbf{X}-\mathbb{E}[\mathbf{X}])(\mathbf{y}-\mathbb{E}[\mathbf{y}])^\top\right],$$

where the expectation is with respect to the joint distribution of $(\mathbf{X},\mathbf{y})$. Obviously, $\mathrm{Cov}(\mathbf{X},\mathbf{y})$ becomes $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ when $\mathbf{y} = \mathbf{X}$.

Properties

Covariance $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ has the following properties:

1. Positive semi-definiteness. It follows from (1) that $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ is positive semi-definite. $\mathrm{Cov}(\mathbf{X},\mathbf{X}) = \mathbf{0}$ if, and only if, $\mathbf{X}$ is a constant almost surely, i.e., there exists a constant $\mathbf{x}$ such that $\Pr(\mathbf{X} \neq \mathbf{x}) = 0$. $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ is not positive definite if, and only if, there exists a constant $\boldsymbol{\alpha}$ such that $\langle \boldsymbol{\alpha}, \mathbf{X}\rangle$ is constant almost surely.
2. Relating cumulant to moments: $\mathrm{Cov}(\mathbf{X},\mathbf{X}) = \mathbb{E}[\mathbf{X}\mathbf{X}^\top] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{X}]^\top$.
3. Linear transform: If $\mathbf{y} = \mathbf{A}\mathbf{X} + \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{s\times d}$ and $\mathbf{b} \in \mathbb{R}^s$, then $\mathrm{Cov}(\mathbf{y},\mathbf{y}) = \mathbf{A}\mathrm{Cov}(\mathbf{X},\mathbf{X})\mathbf{A}^\top$.

Cross-covariance $\mathrm{Cov}(\mathbf{X},\mathbf{y})$ has the following properties.

1. Symmetry: $\mathrm{Cov}(\mathbf{X},\mathbf{y}) = \mathrm{Cov}(\mathbf{y},\mathbf{X})$.
2. Linearity: $\mathrm{Cov}(\mathbf{X}_1 + \mathbf{X}_2, \mathbf{y}) = \mathrm{Cov}(\mathbf{X}_1,\mathbf{y}) + \mathrm{Cov}(\mathbf{X}_2,\mathbf{y})$.
3. Relating to covariance: If $\mathbf{X}$ and $\mathbf{y}$ have the same dimension, then $\mathrm{Cov}(\mathbf{X}+\mathbf{y},\mathbf{X}+\mathbf{y}) = \mathrm{Cov}(\mathbf{X},\mathbf{X}) + \mathrm{Cov}(\mathbf{y},\mathbf{y}) + 2\mathrm{Cov}(\mathbf{y},\mathbf{X})$.
4. Linear transform: $\mathrm{Cov}(\mathbf{A}\mathbf{X},\mathbf{B}\mathbf{y}) = \mathbf{A}\mathrm{Cov}(\mathbf{X},\mathbf{y})\mathbf{B}$.

It is highly important to note that $\mathrm{Cov}(\mathbf{X},\mathbf{y}) = 0$ is a necessary but not sufficient condition for $\mathbf{X}$ and $\mathbf{y}$ to be independent.

Correlation Coefficient

Entries in the covariance matrix are sometimes presented in a normalized form by dividing each entry by its corresponding standard deviations. This quantity is called the correlation coefficient, represented as ρ_{X_i,X_j}, and defined as

$$\rho_{X_i,X_j} = \frac{\mathrm{Cov}(X_i,X_j)}{\mathrm{Cov}(X_i,X_i)^{1/2}\mathrm{Cov}(X_j,X_j)^{1/2}}.$$

The corresponding matrix is called the correlation matrix, and for Γ_X set to $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ with all non-diagonal entries zeroed, and Γ_Y likewise, then the correlation matrix is given by

$$\mathrm{Corr}(\mathbf{X},\mathbf{y}) = \Gamma_X^{-1/2}\mathrm{Cov}(\mathbf{X},\mathbf{y})\Gamma_Y^{-1/2}.$$

The correlation coefficient takes on values between $[-1,1]$.

Parameter Estimation

Given observations $\mathbf{x}_1,\ldots,\mathbf{x}_n$ of a *mrv* $\mathbf{X}$, an unbiased estimator of $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ is:

$$S = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i-\bar{\mathbf{x}})(\mathbf{x}_i-\bar{\mathbf{x}})^\top,$$

where $\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$. The denominator $n-1$ reflects the fact that the mean is unknown and the sample mean is used in place. Note the maximum likelihood estimator in this case replaces the denominator $n-1$ by n.

Conjugate Priors

A covariance matrix is used to define the Gaussian distribution. In this case, the inverse Wishart distribution is the conjugate prior for the covariance matrix. Since the Gamma distribution is a 1-D version of the Wishart distribution, in the 1-D case the Gamma is the conjugate prior for precision matrix.

Applications

Several key uses of the covariance matrix are reviewed here.

Correlation and Kernel Methods

In many machine learning problems, we often need to quantify the correlation of two *mrv* s which may be from two different spaces. For example, we may want to study how much the image stream of a movie is correlated with the comments it receives. For simplicity, we consider a r-dimensional *mrv* $\mathbf{X}$ and a s-dimensional *mrv* $\mathbf{y}$. To study their correlation, suppose we have n pairs of observations $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ drawn *iid* from certain underlying joint distribution of $(\mathbf{X}, \mathbf{y})$. Let $\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i$ and $\bar{\mathbf{y}} = \frac{1}{n}\sum_{i=1}^n \mathbf{y}_i$, and stack $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$ into $\tilde{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$ and $\tilde{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)^\top$ respectively. Then the cross-covariance matrix $\mathrm{Cov}(\mathbf{X}, \mathbf{y})$ can be estimated by $\frac{1}{n}\sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{y}_i - \bar{\mathbf{y}})^\top$. To quantify the cross-correlation by a real number, we need to apply some norm of the cross-covariance matrix, and the simplest one is the Frobenius norm, $\|A\|_F^2 = \sum_{ij} A_{ij}^2$. Therefore, we obtain a measure of cross-correlation,

$$\left\| \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{y}_i - \bar{\mathbf{y}})^\top \right\|_F^2 = \frac{1}{n} H\tilde{x}\tilde{x}^\top H\tilde{Y}\tilde{Y}^\top, \quad (2)$$

where $H_{ij} = \delta_{ij} - \frac{1}{n}$, and $\delta_{ij} = 1$ if $i = j$ and 0 otherwise.

It is important to notice that (1) in this measure, inner product is performed only in the space of $\mathbf{X}$ and $\mathbf{y}$ separately, i.e., no transformation between $\mathbf{X}$ and $\mathbf{y}$ is required, (2) the data points affect the measure only via inner products $\mathbf{x}_i^\top \mathbf{x}_j$ as the (i,j)-*th* entry of $\tilde{x}\tilde{x}^\top$ (and similarly for $\mathbf{y}_i$). Hence we can endow new inner products on $\mathbf{X}$ and $\mathbf{y}$, which eventually allows us to apply kernels, e.g., Gretton, Herbrich, Smola, Bousquet, & Schölkopf (2005). In a nutshell, kernel methods (Schölkopf & Smola, 2002) redefine the inner product $\mathbf{x}_i^\top \mathbf{x}_j$ by mapping $\mathbf{x}_i$ to a richer feature space via $\phi(\mathbf{x}_i)$ and then compute the inner product there: $k(\mathbf{x}_i, \mathbf{x}_j) := \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$. Since the measure in (2) only needs inner products, one can even directly define $k(,)$ without explicitly specifying ϕ. This allows us to

- Implicitly use a rich feature space whose dimension can be infinitely high.
- Apply this measure of cross correlation to non-Euclidean spaces as long as a kernel $k(\mathbf{x}_i, \mathbf{x}_j)$ can be defined on it.

Correlation and Least Squares Approximation

The measure of (2) can be equivalently motivated by least square ▶linear regression. That is, we look for a linear transform $T : \mathbb{R}^d \to \mathbb{R}^s$ which minimizes

$$\frac{1}{n}\sum_{i=1}^n \left\| (\mathbf{y}_i - \bar{\mathbf{y}}) - T(\mathbf{x}_i - \bar{\mathbf{x}}) \right\|^2.$$

And one can show that its minimum objective value is exactly equal to (2) up to a constant, as long as all $\mathbf{y}_i - \bar{\mathbf{y}}$ and $\mathbf{x}_i - \bar{\mathbf{x}}$ have unit length. In practice, this can be achieved by normalization. Or, the measure in (2) itself can be normalized by replacing the covariance matrix with the correlation matrix.

Principal Component Analysis

The covariance matrix plays a key role in principal component analysis (PCA). Assume that we are given n *iid* observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of a *mrv* $\mathbf{X}$, and let $\bar{\mathbf{x}} = \frac{1}{n}\sum_i \mathbf{x}_i$. PCA tries to find a set of orthogonal directions $\mathbf{w}_1, \mathbf{w}_2, \ldots$, such that the projection of $\mathbf{X}$ to the direction $\mathbf{w}_1$, $\mathbf{w}_1^\top \mathbf{X}$, has the highest variance among all possible directions in the d-dimensional space. After subtracting from $\mathbf{X}$ the projection to $\mathbf{w}_1$, $\mathbf{w}_2$ is chosen as the highest variance projection direction for the remainder. This procedure goes on for the required number of components.

To find $\mathbf{w}_1 := \mathrm{argmax}_{\mathbf{w}} \mathrm{Var}(\mathbf{w}^\top \mathbf{X})$, we need an empirical estimate of $\mathrm{Var}(\mathbf{w}^\top \mathbf{X})$. Estimating $\mathbb{E}[(\mathbf{w}^\top \mathbf{X})^2]$ by $\mathbf{w}^\top \left(\frac{1}{n}\sum_i \mathbf{x}_i \mathbf{x}_i^\top\right) \mathbf{w}$, and $\mathbb{E}[\mathbf{w}^\top \mathbf{X}]$ by $\frac{1}{n}\sum_i \mathbf{w}^\top \mathbf{x}_i$, we get

$$\mathbf{w}_1 = \mathrm{argmax}_{\mathbf{w}} : \|\mathbf{w} = 1\| \, \mathbf{w}^\top S \mathbf{w},$$
$$\text{where} \quad S = \frac{1}{n}\sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\top,$$

i.e., S is $\frac{n}{n-1}$ times the unbias empirical estimate of the covariance of $\mathbf{X}$, based on samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$. $\mathbf{w}_1$ turns

out to be exactly the eigenvector of S corresponding to the greatest eigenvalue.

Note that PCA is independent of the distribution of $\mathbf{X}$. More details on PCA can be found at Jolliffe (2002).

Gaussian Processes

Gaussian processes are another important framework in machine learning that rely on the covariance matrix. It is a distribution over functions $f(\cdot)$ from certain space $\mathcal{X}$ to $\mathbb{R}$, such that for any $n \in \mathbb{N}$ and any n points $\{\mathbf{x}_i \in \mathcal{X}\}_{i=1}^n$, the set of values of f evaluated at $\{\mathbf{x}_i\}_i$, $\{f(x_1), \ldots, f(x_n)\}$, will have an n-dimensional Gaussian distribution. Different choices of the covariance matrix of the multi-variate Gaussian lead to different stochastic processes such as Wiener process, Brownian motion, Ornstein–Uhlenbeck process, etc. In these cases, it makes more sense to define a covariance function $C : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, such that given any set $\{\mathbf{x}_i \in \mathcal{X}\}_{i=1}^n$ for any $n \in \mathbb{N}$, the n-by-n matrix $\left(C(\mathbf{x}_i, \mathbf{x}_j)\right)_{ij}$ is positive semi-definite and can be used as the covariance matrix. This further allows straightforward kernelization of a Gaussian process by using the kernel function as the covariance function.

Although the space of functions is infinite dimensional, the marginalization property of multi-variate Gaussian distributions guarantees that the user of the model only needs to consider the observed $\mathbf{x}_i$, and ignore all the other possible $\mathbf{x} \in \mathcal{X}$. This important property says that for a *mrv* $\mathbf{X} = (\mathbf{X}_1^\top, \mathbf{X}_2^\top)^\top \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, the marginal distribution of $\mathbf{X}_1$ is $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_{11})$, where Σ_{11} is the submatrix of Σ corresponding to $\mathbf{X}_1$ (and similarly for $\boldsymbol{\mu}_1$). So taking into account the random variable $\mathbf{X}_2$ will not change the marginal distribution of $\mathbf{X}_1$.

For a complete treatment of covariance matrix from a statistical perspective, see Casella and Berger (2002), and Mardia, Kent, and Bibby (1979) provides details for the multi-variate case. PCA is comprehensively discussed in Jolliffe (2002), and kernel methods are introduced in Schölkopf and Smola (2002). Williams & Rasmussen (2006) gives the state of the art on how Gaussian processes can be utilized for machine learning.

Cross References

►Gaussian Distribution
►Gaussian Processes
►Kernel Methods

Recommended Reading

Casella, G., & Berger, R. (2002). *Statistical inference* (2nd ed.). Pacific Grove, CA: Duxbury.

Gretton, A., Herbrich, R., Smola, A., Bousquet, O., & Schölkopf, B. (2005). Kernel methods for measuring independence. *Journal of Machine Learning Research, 6*, 2075–2129.

Jolliffe, I. T. (2002) *Principal component analysis* (2nd ed.). *Springer series in statistics*. New York: Springer.

Mardia, K. V., Kent, J. T., & Bibby, J. M. (1979). *Multivariate analysis*. London: Academic Press.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels*. Cambridge, MA: MIT Press.

Williams, C. K. I., & Rasmussen, C. E. (2006). *Gaussian processes for regression*. Cambridge, MA: MIT Press.

Covering Algorithm

►Rule Learning

Credit Assignment

Claude Sammut
The University of New South Wales

Synonyms

Structural credit assignment; Temporal credit assignment

Definition

When a learning system employs a complex decision process, it must assign credit or blame for the outcomes to each of its decisions. Where it is not possible to directly attribute an individual outcome to each decision, it is necessary to apportion credit and blame between each of the combinations of decisions that contributed to the outcome. We distinguish two cases in the credit assignment problem. *Temporal credit assignment* refers to the assignment of credit for outcomes to actions. *Structural credit assignment* refers to the assignment of credit for actions to internal decisions. The first subproblem involves determining when the actions that deserve credit were taken and the second involves assigning credit to the internal structure of actions (Sutton, 1984).

Motivation

Consider the problem of learning to balance a pole that is hinged on a cart (Michie & Chambers, 1968, Anderson & Miller, 1991). The cart is constrained to run along a track of finite length and a fixed force can be applied to push the cart left or right. A controller for the pole and cart system must make a decision whether to push left or right at frequent, regular time intervals, for example, 20 times a second. Suppose that this controller is capable of learning from trial-and-error. If the pole falls over, then it must determine which actions it took helped or hurt its performance. Determining that action is the problem of *temporal credit assignment*. Although the actions are directly responsible for the outcome of a trial, the internal process for choosing the action indirectly affects the outcome. Assigning credit or blame to those internal processes that lead to the choice of action is the *structural credit assignment* problem. In the case of pole balancing, the learning system will typically keep statistics such as how long, on average, the pole remained balanced after taking a particular action in a particular state, or after a failure, it may count back and determine the average amount of time to failure after taking a particular action in a particular state. Using these statistics, the learner attempts to determine the best action for a given state.

The above example is typical of many problems in ►reinforcement learning (Sutton & Barto, 1998), where an agent interacts with its environment and through that interaction, learns to improve its performance in a task. Although Samuel (1959) was the first to use a form of reinforcement learning in his checkers playing program, Minksy (1961) first articulated the credit assignment, as follows:

> Using devices that also learn which events are associated with reinforcement, i.e., reward, we can build more autonomous "secondary reinforcement" systems. In applying such methods to complex problems, one encounters a serious difficulty – in distributing credit for success of a complex strategy among the many decisions that were involved.

The BOXES algorithm of Michie and Chambers (1968) learned to control a pole balancer and performed credit assignment but the problem of credit assignment later became central to reinforcement learning, particularly following the work of Sutton (1984). Although credit assignment has become most strongly identified with reinforcement learning, it may appear in any learning system that attempts to assess and revise its decision-making process.

Structural Credit Assignment

The setting for our learning system is that we have an agent that interacts with an environment. The environment may be a virtual one, as in game playing, or it may be physical, as in a robot performing some task. The agent receives input, possibly through sensing devices, that allows it to characterize the state of the world. Somehow, the agent must map these inputs to appropriate responses. These responses may change the state of the world. In reinforcement learning, we assume that the agent will receive some reward signal after an action or sequence of actions. Its job is to maximize these rewards over time.

Structural credit assignment is associated with generalization over the input space of the agent. For example, a game player may have to respond to a very large number of potential board positions or a robot may have to respond to a stream of camera images. It is infeasible to learn a complete mapping from every possible input to every possible output. Therefore, a learning agent will typically use some means of grouping input signals. In the case of the BOXES pole balancer, Michie and Chambers discretized the state space. The state is characterized by the cart's position and velocity and the pole's angle and angular velocity. These parameters create a four-dimensional space, which was broken into three regions (left, center, right) for the pole angle, five for the angular velocity, and three for the cart position and velocity. These choices were arbitrary and other combinations also worked.

Having divided the input space into non-overlapping regions, Michie and Chambers associated a push-left and push-right action with each region, or box. The learning algorithm maintains a score for each action and chooses the next action based on that score. BOXES was an early, and simple example, of creating an internal representation for mapping inputs to outputs. The problem with this method is that the structure of the decision-making system is fixed at the start and the learner is incapable of changing the representation. This may be needed if, for example, the subdivisions

that were chosen do not correspond to a real decision boundary. A learning system that could adapt its representation has an advantage, in this case.

The BOXES representation can be thought of as a lookup table that implements a function that maps an input to an output. The fixed lookup table can be replaced by a ►function approximator that, given examples from the desired function, generalizes from them to construct an approximation of that function. Different function approximation techniques can be used. For example, Moore's (1990) function approximator was a ►nearest-neighbor algorithm, implemented using ►kd-tree to improve efficiency. Other function approximation methods may also be used, e.g., Albus' CMAC algorithm (1975), ►locally weighted regression (Atkeson, Schaal, & Moore, 1997), ►perceptrons (Rosenblatt, 1962), ►multi-layer networks (Hinton, Rumelhart, & Williams, 1985), ►radial basis functions, etc. Structural credit assignment is also addressed in the creation of hierarchical representations. See ►hierarchical reinforcement learning. Other approaches to structural credit assignment include ►Value function approximation (Bertsekas & Tsitsiklis, 1996) and automatic basis generation (Mahadevan, 2009). See the entry on ►Gaussian Processes for examples of recent Bayesian and kernel method based approaches to solving the credit assignment problem.

Temporal Credit Assignment

In the pole balancing example described above, the learning system receives a signal when the pole has fallen over. How does it know which actions leading up to the failure contributed to the fall? The system will receive a high-level punishment in the event of a failure or a reward in tasks where there is a goal to be achieved. In either case, it makes sense to assign the greatest credit or blame to the most recent actions and assign progressively less to the preceding actions. Each time a learning trial is repeated, the value of an action is updated so that if it leads to another action of higher value, its weight is increased. Thus, the reward or punishment propagates back through the sequence of decisions taken by the system. The credit assignment problem was addressed by Michie and Chambers, in the BOXES, algorithm but many other solutions have subsequently been proposed. See the entries on ►Q-learning (Watkins, 1989; Watkins & Dayan, 1992) and ►temporal difference learning (Barto, Sutton, & Anderson, 1983; Sutton, 1984).

Although temporal credit assignment is usually associated with reinforcement learning, it also appears in other forms of learning. In ►learning by imitation or ►behavioral cloning, an agent observes the actions of another agent and tries to learn from traces of behaviors. In this case, the learner must judge which actions of the other agent should receive credit or blame. Plan learning also encounters the same problem (Benson & Nilsson, 1995; Wang, Simon, & Lehman, 1996), as does ►explanation-based learning (Mitchell, Keller, & Kedar-Cabelli, 1986; Dejong & Mooney, 1986; Laird, Newell, & Rosenbloom, 1987).

To illustrate the connection with explanation-based learning, we use one of the earliest examples of this kind of learning, Mitchell and Utgoff's, LEX program (Mitchell, Utgoff, & Banerji, 1983). The program was intended to learn heuristics for performing symbolic integration. Given a mathematical expression that included an integral sign, the program tried to transform the expression into one they did not. The standard symbolic integration operators were known to the program but not when it is best to apply them. The task of the learning system was to learn the heuristics for when to apply the operators. This was done by experimentation. If no heuristics were available, the program attempted a brute force search. If the search was successful, all the operators applied, leading to the success were assumed to be positive examples for a heuristic, whereas operators applied during a failed attempt became negative examples. Thus, LEX performed a simple form of credit assignment, which is typical of any system that learns how to improve sequences of decisions.

►Genetic algorithms can also be used to evolve rules that perform sequences of actions (Holland, 1986). When situation-action rules are applied in a sequence, we have a credit assignment problem that is similar to when we use a reinforcement learning. That is, how do we know which rules were responsible for success or failure and to what extent? Grefenstette (1988) describes a *bucket brigade* algorithm in which rules are given strengths that are adjusted to reflect credit or blame.

This is similar to temporal difference learning except that in the bucket brigade the strengths apply to rules rather than states. See Classifier Systems and for a more comprehensive survey of bucket brigade methods, see Goldberg (1989).

Transfer Learning

After a person has learned to perform some task, learning a new, but related, task is usually easier because knowledge of the first learning episode is *transferred* to the new task. *Transfer Learning* is particularly useful for acquiring new concepts or behaviors when given only a small amount for training data. It can be viewed as a form of credit assignment because successes or failures in previous learning episodes bias future learning. Reid (2004, 2007) identifies three forms of ►inductive bias involved in transfer learning for rules: language bias, which determines what kinds of rules can be constructed by the learner; the search bias, which determines the order in which rules will be searched; and the evaluation bias, which determines how the quality of the rules will be assessed. Note that learning language bias is a form of structural credit assignment. Similarly, where rules are applied sequentially, evaluation bias becomes temporal credit assignment. Taylor and Stone (2009) give a comprehensive survey of transfer in ►reinforcement learning, in which they describe a variety of techniques for transferring the structure of an RL task from one case to another. They also survey methods for transferring evaluation bias.

Transfer learning can be applied in many different settings. Caruana (1997) developed a system for transferring inductive bias in ►neural networks performing multitask learning and more recent research has been directed toward transfer learning in ►Bayesian Networks (Niculescu-mizil & Caruana, 2007).

See ►Transfer Learning and Silver et al. (2005) and Banerjee, Liu, and Youngblood (2006) for recent work on transfer learning.

Cross References

►Bayesian Network
►Classifier Systems
►Genetic Algorithms
►Hierarchical Reinforcement Learning
►Inductive Bias
►kd-Trees
►Locally Weighted Regression
►Nearest-Neighbor
►Perceptrons
►Radial Basis Function
►Reinforcement Learning
►Temporal Difference Learning
►Transfer Learning

Recommended Reading

Albus, J. S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control, Transactions ASME, 97*(3), 220–227.

Anderson, C. W., & Miller, W. T. (1991). A set of challenging control problems. In W. Miller, R. S. Sutton, & P. J. Werbos (Eds.), *Neural Networks for Control*. Cambridge: MIT Press.

Atkeson, C., Schaal, S., & Moore, A. (1997). Locally weighted learning. *AI Review, 11*, 11–73.

Banerjee, B., Liu, Y., & Youngblood, G. M. (Eds.), (2006). *Proceedings of the ICML workshop on "Structural knowledge transfer for machine learning."* Pittsburgh, PA.

Barto, A., Sutton, R., & Anderson, C. (1983). Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13, 834–846.

Benson, S., & Nilsson, N. J. (1995). Reacting, planning and learning in an autonomous agent. In K. Furukawa, D. Michie, & S. Muggleton (Eds.), *Machine Intelligence 14*. Oxford: Oxford University Press.

Bertsekas, D. P., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Nashua, NH: Athena Scientific.

Caruana, R. (1997). Multitask learning. *Machine Learning, 28*, 41–75.

Dejong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1*, 145–176.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Boston: Addison-Wesley Longman Publishing.

Grefenstette, J. J. (1988). Credit assignment in rule discovery systems based on genetic algorithms. *Machine Learning, 3*(2–3), 225–245.

Hinton, G., Rumelhart, D., & Williams, R. (1985). Learning internal representation by back-propagating errors. In D. Rumelhart, J. McClelland, & T. P. R. Group (Eds.), *Parallel distributed computing: Explorations in the microstructure of cognition* (Vol. 1., pp. 31–362). Cambridge: MIT Press.

Holland, J. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos: Morgan Kaufmann.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence, 33*(1), 1–64.

Mahadevan, S. (2009). Learning representation and control in Markov decision processes: New frontiers. *Foundations and Trends in Machine Learning, 1*(4), 403–565.

Michie, D., & Chambers, R. (1968). Boxes: An experiment in adaptive control. In E. Dale & D. Michie (Eds.), *Machine Intelligence 2*. Edinburgh: Oliver and Boyd.

Minsky, M. (1961). Steps towards artificial intelligence. *Proceedings of the IRE, 49*, 8–30.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation based generalisation: A unifying view. *Machine Learning, 1*, 47–80.

Mitchell, T. M., Utgoff, P. E., & Banerji, R. B. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine kearning: An artificial intelligence approach*. Palo Alto: Tioga.

Moore, A. W. (1990). *Efficient memory-based learning for robot control*. Ph.D. Thesis, UCAM-CL-TR-209, Computer Laboratory, University of Cambridge, Cambridge.

Niculescu-mizil, A., & Caruana, R. (2007). Inductive transfer for Bayesian network structure learning. In *Proceedings of the 11th International Conference on AI and Statistics* (*AISTATS 2007*). San Juan, Puerto Rico.

Reid, M. D. (2004). Improving rule evaluation using multitask learning. In *Proceedings of the 14th International Conference on Inductive Logic Programming* (pp. 252–269). Porto, Portugal.

Reid, M. D. (2007). *DEFT guessing: Using inductive transfer to improve rule evaluation from limited data*. Ph.D. thesis, School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanics*. Washington, DC: Spartan Books.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development, 3*(3), 210–229.

Silver, D., Bakir, G., Bennett, K., Caruana, R., Pontil, M., Russell, S., et al. (2005). NIPS workshop on "Inductive transfer: 10 years later". Whistler, Canada.

Sutton, R. (1984). *Temporal credit assignment in reinforcement learning*. Ph.D. thesis, Department of Computer and Information Science, University of Massachusetts, Amherst, MA.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge: MIT Press.

Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research, 10*, 1633–1685.

Wang, X., Simon, H. A., Lehman, J. F., & Fisher, D. H. (1996). Learning planning operators by observation and practice. In *Proceedings of the Second International Conference on AI Planning Systems, AIPS-94* (pp. 335–340). Chicago, IL.

Watkins, C. (1989). *Learning with delayed rewards*. Ph.D. thesis, Psychology Department, University of Cambridge, Cambridge.

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3–4), 279–292.

Cross-Language Document Categorization

Document Categorization is the task consisting in assigning a document to zero, one or more categories in a predefined taxonomy. *Cross-language document categorization* describes the specific case in which one is interested in automatically categorize a document in a same taxonomy regardless of the fact that the document is written in one of several languages. For more details on the methods used to perform this task see ►cross-lingual text mining.

Cross-Language Information Retrieval

Cross-language information retrieval (CLIR) is the task consisting in recovering the subset of a document collection D relevant to a query q, in the special case in which D contains documents written in more than one language. Generally, it is additionally assumed that the subset of relevant documents must be returned as an ordered list, in decreasing order of relevance. For more details on methods and applications see ►cross-lingual text mining.

Cross-Language Question Answering

Question answering is the task consisting in finding in a document collection the answer to a question. CLCat is the specific case in which the question and the documents can be in different languages. For more details on the methods used to perform this task see ►*cross-lingual text mining*.

Cross-Lingual Text Mining

Nicola Cancedda, Jean-Michel Renders
Xerox Research Centre Europe, Meylan,
France

Definition

Cross-lingual text mining is a general category denoting tasks and methods for accessing the information in sets of documents written in several languages, or whenever the language used to express an information need is different from the language of the documents. A distinguishing feature of cross-lingual text mining is the necessity to overcome some language translation barrier.

Motivation and Background

Advances in mass storage and network connectivity make enormous amounts of information easily accessible to an increasingly large fraction of the world population. Such information is mostly encoded in the form of running text which, in most cases, is written in a language different from the native language of the user. This state of affairs creates many situations in which the main barrier to the fulfillment of an information need is not technological but linguistic. For example, in some cases the user has some knowledge of the language in which the text containing a relevant piece of information is written, but does not have a sufficient control of this language to express his/her information needs. In other cases, documents in many different languages must be categorized in a same categorization schema, but manually categorized examples are available for only one language.

While the automatic translation of text from a natural language into another (machine translation) is one of the oldest problems on which computers have been used, a palette of other tasks has become relevant only more recently, due to the technological advances mentioned above. Most of them were originally motivated by needs of government Intelligence communities, but received a strong impulse from the diffusion of the World-Wide Web and of the Internet in general.

Tasks and Methods

A number of specific tasks fall under the term of Cross-lingual text mining (CLTM), including:

- *Cross-language information retrieval*
- *Cross-language document categorization*
- *Cross-language document clustering*
- *Cross-language question answering*

These tasks can in principle be performed using methods which do not involve any ▶Text Mining, but as a matter of fact all of them have been successfully approached relying on the statistical analysis of multilingual document collections, especially *parallel corpora*. While CLTM tasks differ in many respect, they are all characterized by the fact that they require to reliably measure the similarity of two text spans written in different languages. There are essentially two families of approaches for doing this:

1. In *translation-based* approaches one of the two text spans is first translated into the language of the other. Similarity is then computed based on any measure used in mono-lingual cases. As a variant, both text spans can be translated in a third *pivot* language.
2. In *latent semantics* approaches, an abstract vector space is defined based on the statistical properties of a *parallel corpus* (or, more rarely, of a *comparable corpus*). Both text spans are then represented as vectors in such *latent semantic* space, where any similarity measure for vector spaces can be used.

The rest of this entry is organized as follows: first Translation-related approaches will be introduced, followed by Latent-semantic approaches. Finally, each of the specific CLTM tasks will be discussed in turn.

Translation-Based Approaches

The simplest approach consists in using a manually-written machine-readable bilingual dictionary: words from the first span are looked up and replaced with words in the second language (see e.g., Zhang & Vines, 2005). Since typically dictionaries contain entries for "citation forms" only (e.g., the singular for nouns, the infinitive for verbs etc.), words in both spans are preliminarily *lemmatized*, i.e., replaced with the corresponding

citation form. In all cases when the lexica and morphological analyzers required to perform lemmatization are not available, a frequently adopted crude alternative consists in *stemming* (i.e., truncating by taking away a suffix) both the words in the span to be translated and in the corresponding side in the lexicon. Some languages (e.g., Germanic languages) are characterized by a very productive *compounding*: simpler words are connected together to form complex words. Compound words are rarely in dictionaries as such: in order to find them it is first necessary to break compounds into their elements. This can be done based on additional linguistic resources or by means of heuristics, but in all cases it is a challenging operation in itself. If the method used afterward to compare the two spans in the target language can take weights into account, translations are "normalized" in such a way that the cumulative weight of all translations of a word is the same regardless of the number of alternative translations. Most often, the weight is simply distributed uniformly among all alternative translations. Sometimes, only the first translation for each word is kept, or the first two or three.

A second approach consists in extracting a bilingual lexicon from a *parallel corpus* instead of using a manually-written one. Methods for extracting probabilistic lexica look at the frequencies with which a word s in one language was translated with a word t to estimate the translation probability $p(t|s)$. In order to determine which word is the translation of which other word in the available examples, these examples are preliminarily aligned, first at the sentence level (to know what sentence is the translation of what other sentence) and then at the word level. Several methods for aligning sentences at the word level have been proposed, and this problem is a lively research topic in itself (see Brown, Della Pietra, Della Pietra, & Mercer, 1993 for a seminal paper).

Once a probabilistic bilingual dictionary is available, it can be used much in the same way as human-written dictionaries, with the notable difference that the estimated conditional probabilities provide a natural way to distribute weight across translations. When the example documents used for extracting the bilingual dictionaries are of the same style and domain as the text spans to be translated, this can result in a significant increase in accuracy for the final task, whatever this is.

It is often the case that a parallel corpus sufficiently similar in topic and style to the spans to be translated is unavailable, or it is too small to be used for reliably estimating translation probabilities. In such cases, it can be possible to replace or complement the parallel corpus with a "comparable" corpus. A comparable corpus is a pair of collections of documents, one in each of the languages of interest, which are known to be similar in content, although not the translation of one another. A typical case might be two sets of articles from corresponding sections of different newspapers collected during a same period of time. If some additional bilingual *seed* dictionary (human-written or extracted from a parallel corpus) is also available, then the comparable corpus can be leveraged as well: a word t is likely to be the translation of a word s if it turns out that the words often appearing near s are translations of the words often appearing near t. Using this observation it is thus possible to estimate the probability that t is a valid translation of s even though they are not contained in the original dictionary. Most approaches proceed by associating with s a *context vector*. This vector, with one component for each word in the source language, can simply be formed by summing together the count histograms of the words occurring within a fixed window centered in all occurrences of s in the corpus, but is often constructed using statistically more robust association measures, such as mutual information. After a possible normalization step, the context vector $CV(s)$ is translated using the seed dictionary into the target language. A context vector is also extracted from the corpus for all target words t. Eventually, a translation score between s and t is computed as $\langle Tr(CV(s)), CV(t)\rangle$:

$$\begin{aligned}\mathcal{S}(s,t) &= \langle CV(s), Tr(CV(t))\rangle \\ &= \sum_{(s',t')\in\mathcal{D}} a(s,s')a(t,t'),\end{aligned}$$

where a is the association score used to construct the context vector. While effective in many cases, this approach can provide inaccurate similarity values when polysemous words and synonyms appear in the corpus. To deal with this problem, Gaussier, Renders, Matveeva, Goutte, and Déjean (2004) propose the following extension:

$$\begin{aligned}\mathcal{S}(s,t) = \sum_{(s',t')\in\mathcal{D}} &\Big(\sum_{s'} a(s',s'')a(s,s'')\Big) \\ &\Big(\sum_{t''} a(t',t'')a(t,t'')\Big),\end{aligned}$$

which is more robust in cases when the entries in the seed bilingual dictionary do not cover all senses

actually present in the two sides of the comparable corpus.

Although these methods for building bilingual dictionaries can be (and often are) used in isolation, it can be more effective to combine them.

Using a bilingual dictionary directly is not the only way for translating a span from one language into another. A second alternative consists in using a *machine translation* (MT) system. While the MT system, in turn, relies on a bilingual dictionary of some sort, it is in general in the position of leveraging contextual clues to select the correct words and put them in the right order in the translation. This can be more or less useful depending on the specific task. MT systems fall, broadly speaking, into two classes: rule-based and statistical. Systems in the first class rely on sets of hand-written rules describing how words and syntactic structures should be translated. Statistical machine translation (SMT) systems learn this mapping by performing a statistical analysis of a parallel corpus. Some authors (e.g., Savoy & Berger, 2005) also experimented with combining translation from multiple machine translation systems.

Latent Semantic Approaches

In CLTM, *Latent Semantic* approaches rely on some interlingua (language-independent) representation. Most of the time, this interlingua representation is obtained by linear or non-linear statistical analysis techniques and more specifically ►dimensionality reduction methods with ad-hoc optimization criterion and constraints. But, others adopt a more manual approach by exploiting multilingual thesauri or even multilingual ontologies in order to map textual objects towards a list – possibly weighted – of interlingua concepts.

For any textual object (typically a document or a section of document), the *interlingua* concept representation is derived from a sequence of operations that encompass:

1. Linguistic preprocessing (as explained in previous sections, this step amounts to extract the relevant, normalized "terms" of the textual objects, by tokenisation, word segmentation/decompounding, lemmatisation/stemming, part-of-speech tagging, stopword removal, corpus-based term filtering, Noun-phrase extractions, etc.).
2. Semantic enrichment and/or monolingual dimensionality reduction.
3. Interlingua semantic projection.

A typical semantic enrichment method is the *generalized vector space model*, that adds related terms – or neighbour terms – to each term of the textual object, neighbour terms being defined by some co-occurrence measures (for instance, mutual information). Semantic enrichment can alternatively be achieved by using (monolingual) thesaurus, exploiting relationships such as synonymy, hyperonymy and hyponymy. Monolingual dimensionality reduction consists typically in performing some *latent semantic analysis* (LSA), some form of principal component analysis on the textual object/term matrix. Dimensionality reduction techniques such as LSA or their discrete/probabilistic variants such as *probabilistic semantic analysis* (PLSA) and *latent dirichlet allocation* (LDA) offer to some extent a semantic robustness to deal with the effects of polysemy/synonymy, adopting a language-dependent concept representation in a space of dimension much smaller than the size of the vocabulary in a language.

Of course, steps (1) and (2) are highly language-dependent. Textual objects written in different languages will not follow the same linguistic processing or semantic enrichment/ dimensionality reduction. The last step (3), however, aims at projecting textual objects in the same language-independent concept space, for any source language. This is done by first extracting these common concepts, typically from a parallel corpus that offers a natural multiple-view representation of the same objects. Starting from these multiple-view observations, common factors are extracted through the use of canonical correlation analysis (CCA), cross-language latent semantic analysis, their kernelized variants (eg. Kernel-CCA) or their discrete, probabilistic extensions (cross-language latent dirichlet allocation, multinomial CCA, …). All these methods try to discover latent factors that simultaneously explain as much as possible the "intra-language" variance and the "inter-language" correlation. They differ in the choice of the underlying distributions and how they precisely define and combine these two criteria. The following subsections will describe them in more details.

As already emphasized, CLTM mainly relies on defining appropriate similarities between textual objects

expressed in different languages. Numerous categorization, clustering and retrieval algorithms focus on defining efficient and powerful measures of similarity between objects, as strengthened recently by the development of kernel methods for textual information access. We will see that the (linear) statistical algorithms used for performing steps (2) and (3) can most of the time be embedded into one valid (Mercer) kernel, so that we can very easily obtain non-linear variants of these algorithms, just by adopting some standard non-linear kernels.

Cross-Language Semantic Analysis

This amounts to concatenate the vectorial representation of each view of the objects of the parallel collection (typically, objects are aligned sentences), and then to perform standard singular value decomposition of the global object/term matrix. Equivalently, defining the kernel similarity matrix between all pairs of multi-view objects as the sum of the mono-lingual textual similarity matrices, this amounts to perform the eigenvalue decomposition of the corresponding kernel Gram matrix, if a dual formulation is adopted. The number of eigenvalues/eigenvectors that are retained to define the latent factors and the corresponding projections is typically from several hundreds of components to several thousands, still much fewer than the original sizes of the vocabulary. Note that this process does not really control the formation of *interlingua* concepts: nothing prevents the method from extracting factors that are linear combination of terms in one language only.

Cross-Language Latent Dirichlet Allocation

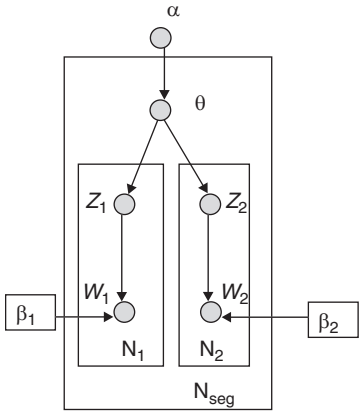
The extraction of *interlingua components* is realised by using LDA to model the set of parallel objects, by imposing the same proportion of components (topics) for all views of the same object. This is represented in Fig. 1.

LDA is performing some form of clustering, with a predefined number of components (K) and with the constraint that the two views of the same object belongs to the clusters with the same membership values. This results in 2.K component profiles that are then used for "folding in" (projecting) new documents by launching some form of EM to derive their posterior probabilities to belong to each of the language-independent component. The similarity between two documents written in different languages is obtained by comparing their posterior distribution over these latent classes. Note that this approach could easily integrate supervised topic information and provides a nice framework for semi-supervised *interlingua* concept extraction.

Cross-Language Canonical Correlation Analysis

The Primal Formulation CCA is a standard statistical method to perform multi-block multivariate analysis, the goal being to find linear combinations of variables for each block (i.e., each language) that are maximally correlated. In other words, CCA is able to enforce the commonality of latent concept formations by extracting maximally correlated projections. Starting from a set of paired views of the same objects (typically, aligned sentences of a parallel corpus) in languages L1 and L2, the algebraic formulation of this optimization problem leads to a generalized eigenvalue problem of size $(n_1 + n_2)$, where n_1 and n_2 are the sizes of the vocabularies in L1 and L2 respectively. For obvious scalability reasons, the dual – or kernel – formulation (of size N, the number of paired objects in the training set) is often preferred.

Kernel Canonical Correlation Analysis Basically, Kernel Canonical Correlation Analysis amounts to do CCA on some implicit, but more complex feature space and to express the projection coefficients as linear combination of the training paired objects. This results in the dual formulation, which is a generalized eigenvalue/vector



Cross-Lingual Text Mining. Figure 1. Latent dirichlet allocation of a parallel corpus

problem of size $2N$, that involves only the monolingual kernel gram matrices K_1 and K_2 (matrices of monolingual textual similarities between all pairs of objects in the training set in language L1 and L2 respectively). Note that it is easy to show that the eigenvalues go by pairs: we always have two symmetrical eigenvalues $+\lambda$ and $-\lambda$. This kernel formulation has the advantage to include any text specific prior properties in the kernel (e.g., use of N-gram kernels, word-sequence kernels, and any semantically-smoothed kernel). After extraction of the first k generalized eigenvalues/eigenvectors, the similarity between any pair of test objects in languages L1 and L2 can be computed by using projection matrices composed of extracted eigenvector as well as the (monolingual) kernels of the test objects with the training objects.

Regularization and Partial Least Squares Solution When the number of training examples (N) is less than n_1 and n_2 (the dimensions of the monolingual feature spaces), the eigenvalue spectrum of the KCCA problem has generally two null eigenvalues (due to data centering), $(N-1)$ eigenvalues in +1 and $(N-1)$ eigenvalues in −1, so that, as such, the KCCA problem only results in trivial solutions and is useless. When using kernel methods, the case $(N < n_1, n_2)$ is frequent, so that some regularization scheme is needed. One way of realizing this regularization is to resort to finding the directions of maximum covariance (instead of correlation): this can be considered as a partial least squares (PLS) problem, whose formulation is very similar to the CCA problem. Adopting a mixed criterion CCA/PLS (trying to maximize a combination of covariance and correlation between projections) turns out to both avoid overfitting (or spurious solutions) and to enhance numerical stability.

Approximate Solutions Both CCA and KCCA suffer from a lack of scalability, due to the fact the complexity of generalized eigenvalue/vector decomposition is $O(N^3)$ for KCCA or $O(\min(n_1, n_2)^3)$ for CCA. As it can be shown that performing a complete KCCA (or KPLS) analysis amounts to do first complete PCA's, and then a linear CCA (or PLS) on the resulting new projections, it is obvious that we could reduce the complexity by working on a reduced-rank approximation (incomplete KPCA) of the kernel matrices. However, the implicit projections derived from incomplete KPCA may be not optimal with respect to cross-correlation or covariance criteria. Another idea to decrease the complexity is to perform some incomplete Cholesky decomposition of the (monolingual) kernel matrices K_1 and K_2 (that is equivalent to partial Gram-Schmit orthogonalisation in the feature space): $K_1 = G_1.G_1^t$ and $K_2 = G_2.G_2^t$, with G_i of rank $k \ll N$. Considering G_i as the new representation of the training data, KCCA now reduces to solving a generalized eigenvalue problem of size $2.k$.

Specific Applications

The previous sections illustrated a number of different ways of solving the core problem of cross-language text mining: quantifying the similarity between two spans of text in different languages. In this section we turn to describing some actual applications relying on these methods.

Cross-Language Information Retrieval (CLIR)

Given a collection of documents in several languages and a single query, the CLIR problem consists in producing a single ranking of all documents according to their relevance to the query. CLIR is in particular useful whenever a user has some knowledge of the languages in which documents are written, but not enough to express his/her information needs in those languages by means of a precise query. Sometimes CLIR engines are coupled with translation tools to help the user access the content of relevant documents written in languages unknown to him/her. In this case document collections in an even larger number of languages can be effectively queried.

It is probably fair to say that the vast majority of the CLIR systems use a translation-based approach. In most cases it is the query which is translated in all languages before being sent to monolingual search engines. While this limits the amount of translation work that needs be done, it requires doing it on-line at query time. Moreover, when queries are short it can be difficult to translate them correctly, since there is little context to help identifying the correct sense in which words are used. For these reasons several groups also proposed translating all documents at indexing time instead. Regardless of whether queries or documents

are translated, whenever similarity scores between (possibly translated) queries and (possibly translated) documents are not directly comparable, all methods then face the problem of merging multiple monolingual rankings in a single multilingual ranking.

Research in CLIR and cross-language question answering (see below) has been significantly stimulated by at least three government-sponsored evaluation campaigns:

- The NII Test Collection for IR Systems (NTCIR) (http://research.nii.ac.jp/ntcir/), running yearly since 1999, focusing on Asian languages (Japanese, Chinese, Korean) and English.
- The Cross-Language Evaluation Forum (CLEF) (http://www.clef-campaign.org), running yearly since 2000, focusing on European languages.
- A cross-language track at the Text Retrieval Conference (TREC) (http://trec.nist.gov/), which was run until 2002, focused on querying documents in Arabic using queries in English.

The respective websites are ideal starting points for any further exploration on the subject.

Cross-Language Question Answering (CLQA)

Question answering is the task of automatically finding the answer to a specific question in a document collection. While in practice this vague description can be instantiated in many different ways, the sense in which the term is mostly understood is strongly influenced by the task specification formulated by the National Institute of Science and Technology (NIST) of the United States for its TREC evaluation conferences (see above). In this sense, the task consists in identifying a *text snippet*, i.e., a substring, of a predefined maximal length (e.g., 50 characters, or 200 characters) within a document in the collection containing the answer. Different classes of questions are considered:

- Questions around facts and events.
- Questions requiring the definition of people, things and organizations.
- Questions requiring as answer lists of people, objects or data.

Most proposals for solving the QA problem proceed by first identifying promising documents (or document segments) by using information retrieval techniques treating the question as a query, and then performing some finer-grained analysis to converge to a sufficiently short snippet. Questions are classified in a hierarchy of possible "question types." Also, documents are preliminarily indexed to identify elements (e.g., person names) that are potential answers to questions of relevant types (e.g., "Who" questions).

Cross-language question answering (CLQA) is the extension of this task to the case where the collection contains documents in a language different than the language of the question. In this task a CLIR step replaces the monolingual IR step to shortlist promising documents. The classification of the question is generally done in the source language.

Both CLEF and NTCIR (see above) organize cross-language question answering comparative evaluations on an annual basis.

Cross-Language Categorization (CLCat) and Clustering (CLCLu)

Cross-language categorization tackles the problem of categorizing documents in different languages in a same categorization scheme.

The vast majority of document categorization systems rely on machine learning techniques to automatically acquire the necessary knowledge (often referred to as a *model*) from a possibly large collection of manually categorized documents. Most often the model is based on frequency counts of words, and is thus intrinsically language-dependent. The most direct way to perform categorization in different languages would consist in manually categorizing a sufficient amount of documents in all languages of interest and then train a set of independent categorizer. In some cases, however, it is impractical to manually categorize a sufficient number of documents to ensure accurate categorization in all languages, while it can be easier to identify bilingual dictionaries or parallel (or comparable) corpora for the language pairs and in the application domain of interest. In such cases it is then preferable to obtain manually categorized documents only for a single language A and use them to train a monolingual categorizer. Any of the translation-based approaches described above can then be used to translate a document originally in language B – or most often its representation as a *bag of*

words– into language *A*. Once the document is translated, it can be categorized using the monolingual *A* system.

As an alternative, latent-semantics approaches can be used as well. An existing parallel corpus can be used to identify an abstract vector space common to *A* and *B*. The manually categorized documents in *A* can then be represented in this space, and a model can be learned which operates directly on this latent-semantic representation. Whenever a document in *B* needs be categorized, it is first projected in the common semantic space and then categorized using the same model.

All these considerations carry unchanged to the cross-language clustering task, which consists in identifying subsets of documents in a multilingual document collection which are mutually similar to one another according to some criterion. Again, this task can be effectively solved by either translating all documents into a single language or by learning a common semantic space and performing the clustering task there.

While CLCat and Clustering are relevant tasks in many real-world situations, it is probably fair to say that less effort has been devoted to them by the research community than to CLIR and CLQA.

Recommended Reading

Brown, P. E., Della Pietra, V. J., Della Pietra, S. A., & Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics, 12*(2), 263–311.

Gaussier, E., Renders, J.-M., Matveeva, I., Goutte, C., & Déjean, H. (2004). A geometric view on bilingual lexicon extraction from comparable corpora. In *Proceedings of the 42nd annual meeting of the association for computational linguistics*, Barcelona, Spain. Morristown, NJ: Association for Computational Linguistics.

Savoy, J., & Berger, P. Y. (2005). Report on CLEF-2005 evaluation campaign: Monolingual, bilingual and GIRT information retrieval. In *Proceedings of the cross-language evaluation forum (CLEF)* (pp. 131–140). Heidelberg: Springer.

Zhang, Y., & Vines, P. (2005). Using the web for translation disambiguation. In *Proceedings of the NTCIR-5 workshop meeting*, Tokyo, Japan.

Cross-Validation

Definition

Cross-validation is a process for creating a distribution of pairs of ►training and ►test sets out of a single ►data set. In cross validation the data are partitioned into k subsets, $S_1 \ldots S_k$, each called a *fold*. The folds are usually of approximately the same size. The learning algorithm is then applied k times, for $i = 1$ to k, each time using the union of all subsets other than S_i as the ►training set and using S_i as the ►test set.

Cross References

►Algorithm Evaluation
►Leave-One-Out Cross-Validation

Cumulative Learning

Pietro Michelucci[1], Daniel Oblinger[2]
[1]Strategic Analysis, Inc., Arlington, VA, USA
[2]DARPA/IPTO, Arlington, VA , USA

Synonyms

Continual learning; Lifelong learning; Sequential inductive transfer

Definition

Cumulative learning (CL) exploits knowledge acquired on prior tasks to improve learning performance on subsequent related tasks. Consider, for example, a CL system that is learning to play chess. Here, one might expect the system to learn from prior games concepts (e.g., favorable board positions, standard openings, end games, etc.) that can be used for future learning. This is in contrast to base learning (Vilalta & Drissi, 2002) in which a fixed learning algorithm is applied to a single task and performance tends to improve only with more exemplars. So, in CL there tends to be explicit reuse of learned knowledge to constrain new learning, whereas base learning depends entirely upon new external inputs.

Relevant techniques for CL operate over multiple tasks, often at higher levels of abstraction, such as new problem space representations, task-based selection of learning algorithms, dynamic adjustment of learning parameters, and iterative analysis and modification of the learning algorithms themselves. Though actual usage of this term is varied and evolving, CL typically connotes sequential ►inductive transfer. It should be noted that the word "inductive" in this connotation

qualifies the transfer of knowledge to new tasks, not the underlying learning algorithms.

Related Terminology

The terms "meta-learning" and "learning to learn" are sometimes used interchangeably with CL. However each of these concepts has a specific relationship to CL.

►Meta-learning (Brazdil et al., 2009; Vilalta & Drissi, 2002) involves the application of learning algorithms to meta-data, which are abstracted representations of input data or learning system knowledge. In the case that abstractions of system knowledge are themselves learning algorithms, meta-learning involves assessing the suitability of these algorithms for previous tasks and, on that basis, selecting algorithms for new tasks (see entry on "meta-learning"). In general, the sharing of abstracted knowledge across tasks in a CL system implies the use of meta-learning techniques. However, the converse is not true. Meta-learning can and does occur in learning systems that do not accumulate and transfer knowledge across tasks.

Learning to learn is a synonym for inductive transfer. Thus, learning to learn is more general than CL. Though it specifies the application of knowledge learned in one domain to another, it does not stipulate whether that knowledge is accumulated and applied sequentially or shared in a parallel learning context.

Motivation and Background

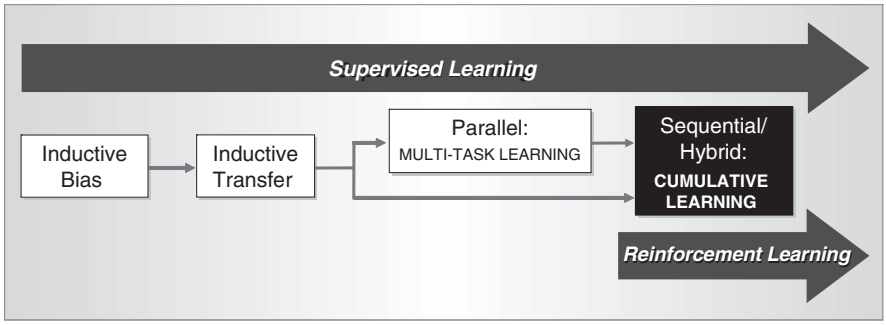
Traditional ►supervised learning approaches require large datasets and extensive training in order to generalize to new inputs in a single task. Furthermore, traditional (non-CL) ►reinforcement learning approaches require tightly constrained environments to ensure a tractable state space. In contrast, humans are able to generalize across tasks in dynamic environments from brief exposure to small datasets. The human advantage seems to derive from the ability to draw upon prior task and context knowledge to constrain hypothesis development for new tasks. Recognition of this disparity between human learning and traditional machine learning had led to the pursuit of methods that seek to emulate the accumulation and exploitation of task-based knowledge that is observed in humans. A coarse evolution of this work is depicted in Fig. 1.

History

Advancements in CL have resulted from two classes of innovation: the development of techniques for ►inductive transfer and the integration of those techniques into autonomous learning systems.

Alan Turing (1950) was the first to propose a cumulative learning system. His 1950 paper is best remembered for the imitation game, later known as the Turing test. However, the final sections of the paper address the question of how a machine could be made sufficiently complex to be able to pass the test. He posited that programming it would be too difficult a task. Therefore, it should be instructed as one might teach a child, starting with simple concepts and working up to more complex ones.

Banerji (1964) introduced the use of predicate logic as a description language for machine learning. Thus, Banerji was one of the earliest advocates of what would later become ►ILP. His concept description language allowed the use of background knowledge and therefore was an extensible language. The first implementation of a cumulative learning system based on Banerji's ideas was Cohen's CONFUCIUS (Cohen, 1978;



Cumulative Learning. Figure 1. Evolution of cumulative learning

Cohen & Sammut, 1982). In this work, an instructor teaches the system concepts that are stored in a long-term memory. When examples of a new concept are seen, their descriptions are matched against stored concepts, which allow the system to re-describe the examples in terms of the background knowledge. Thus, as more concepts are accumulated, the system is capable of describing complex objects more compactly than if it had not had the background knowledge. Compact representations generally allow complex concepts to be learned more efficiently. In many cases, learning would be intractable without the prior knowledge. See the entries on ►Inductive Logic Programming, which describe the use of background knowledge further.

Independent of the research in symbolic learning, much of the ►inductive transfer research that underlies CL took root in ►artificial neural network research, a traditional approach to ►supervised learning. For example, Abu-Mostafa (1990) introduced the notion of reducing the hypothesis space of a neural network by introducing "hints" either as hard-wired additions to the network or via examples designed to teach a particular invariance. The task of a neural network can be thought of as the determination of a function that maps exemplars into a classification space. So, in this context, hints constitute an articulation of some aspect of the target mapping function. For example, if a neural network is tasked with mapping numbers into primes and composites, one "hint" would be that all even numbers (besides 2) are composite. Leveraging such a priori knowledge about the mapping function may facilitate convergence on a solution. An inherent limitation to neural networks, however, is their immutable architecture, which does not lend itself to the continual accumulation of knowledge. Consequently, Ring (1991) introduced a neural network that constructs new nodes on demand in a reinforcement learning context in order to support ongoing hierarchical knowledge acquisition and transfer. In this model, nodes called "bions" correspond simultaneously to the enactment and perception of a single behavior. If two bions are activated in sequence repeatedly, a new bion is created to join the coincident pair and represent their collective functionality.

Contemporaneously, Pratt, Mostow, and Kamm (1991) investigated the hypothesis that knowledge acquired by one neural network could be used to assist another neural network learn a related task. In the speech recognition domain, they trained three separate networks, each corresponding to speech segments of a different length, such that each network was optimized to learn certain types of phonemes. They then demonstrated that a direct transfer of information encoded as network weights from these three specialized networks to a single, combined speech recognition network resulted in a tenfold reduction in training epochs for the combined network compared with the number of training epochs required when no knowledge was transferred. This was one of the first empirical results in neural network-based transfer learning. Caruana (1993) extended this work to demonstrate the performance benefits associated with the simultaneous transfer of ►inductive bias in a "Multitask Learning" (MTL) methodology. In this work, Caruana hypothesized that training the same neural network simultaneously on related tasks would naturally induce additional constraints on learning for each individual task. The intuition was that converging on a mapping in support of multiple tasks with shared representations might best reveal aspects of the input that are invariant across tasks, thus obviating within-task regularities, which might be less relevant to classification. Those empirical results are supported by Baxter (1995) who proved that the number of examples required by a representation learner for learning a single task is an inverse linear function of the number of simultaneous tasks being learned.

Though the innovative underpinnings of inductive transfer that critically underlie CL evolved in a supervised learning context, it was the integration of those methods with classical reinforcement learning that has led to current models of CL. Early integration of this type comes from Thrun and Mitchell (1995), who applied an extension of explanation-based learning (EBL), called explanation-based neural networks (EBNN) (Mitchell & Thrun, 1993), to an agent-based "lifelong learning framework." This framework provides for the acquisition of different control policies for different environments and reward functions. Since the robot actuators, sensors, and the environment (largely) remain invariant, this framework supports the use of knowledge acquired from one control problem to be applied to another. By using EBNN to allow learning

from previous control problems to constrain learning on new control problems, learning is accelerated over the lifetime of the robot.

More recently, Silver and Mercer (2002) introduced a hybrid model that involves a combination of parallel and sequential inductive transfer in an autonomous agent framework. The so-called task rehearsal method (TRM) uses MTL to combine new training inputs with relevant exemplars that are generated from prior task knowledge. Thus, inductive bias is achieved by training the neural networks on new tasks while simultaneously rehearsing learned task knowledge.

Structure of the Learning System

CL is characterized by systems that use prior knowledge to bias future learning. The canonical interpretation is that knowledge transfer occurs at the task level. Although this description encompasses a broad research space, it is not boundless. In particular, CL systems must be able to (1) retain knowledge and (2) use that knowledge to restrict the hypothesis space for new learning. Nonetheless, learning systems can vary widely across numerous orthogonal dimensions and still meet these criteria.

Toward a CL Specification

Recognizing the empirical utility of a more specific delineation of CL systems, Silver and Poirier (2005) introduced a set of functional requirements, classification criteria, and performance specifications that characterize more precisely the scope of machines capable of lifelong learning. Any system that meets these requirements is considered a machine lifelong learning (ML3) system. A general CL architecture that conforms to the ML3 standard is depicted in Fig. 2.

Two basic memory constructs are typical of CL systems. Long term memory (LTM) is required for storing domain knowledge (DK) that can be used to bias new learning. Short term memory (STM) provides a working memory for building representations and testing hypotheses associated with new task learning. Most of the ML3 requirements specify the interplay of these constructs.

LTM and STM are depicted in Fig. 2, along with a comparison process, an assessment process, and the learning environment. In this model, the comparison process evaluates the training input in the context of LTM to determine the most relevant domain knowledge that can be used to constrain short term learning. The comparison process also determines the weight assigned to domain knowledge that is used to bias short term learning. Once the rate of performance improvement on the primary task falls below a threshold the assessment process compares the state of STM to the environment to determine which domain knowledge to extract and store in LTM.

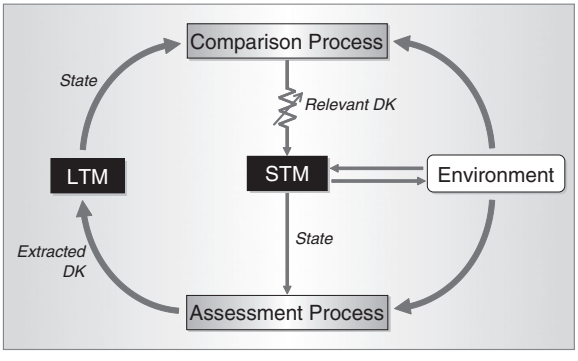
Classification of CL Systems

The simplicity of the architecture shown in Fig. 2 belies the richness of the feature space for CL systems. The following classification dimensions are derived largely from the ML3 specification. This list includes both qualitative and quantitative dimensions. They are presented in three overlapping categories: architectural features, characteristics of the knowledge base, and learning capabilities.

Architecture

The following architectural dimensions for a CL system range from paradigm choices to low-level interface considerations.

Learning paradigm – The learning paradigm(s) may include supervised learning (e.g., neural network, SVM, ILP, etc.), unsupervised learning (e.g., clustering), reinforcement learning (e.g., automated agent), or some combination thereof. Figure 2 depicts a general architecture with processes that are common across these



Cumulative Learning. Figure 2. Typical CL system

learning paradigms, and which could be elaborated to reflect the details of each.

Task order – CL systems may learn tasks sequentially (Thrun & Mitchell, 1995), in parallel (e.g., MTL (Caruana, 1993)), or via a hybrid methodology (e.g., TRM (Silver & Mercer, 2002)). One hybrid approach is to engage in practice (i.e., revisiting prior learned tasks). Transferring knowledge between learned tasks through practice may serve to improve generalization accuracy. Task order would be reflected in the sequence of events within and among process arrows in the Fig. 2 architecture. For example, a system may alternate between processing new exemplars and "practicing" with old, stored exemplars.

Transfer method – Knowledge transfer can also be representational or functional. Functional transfer provides implicit pressure from related training exemplars. For example, the environmental input in Fig. 2 may take the form of training exemplars drawn randomly from data representing two related tasks, such that learning to classify exemplars from one task implicitly constrains learning on the other task. Representational knowledge transfer involves the direct or indirect (Pratt et al., 1991) assignment of a hypothesis representation. A direct inductive transfer entails the assignment of an original hypothesis representation, such as a vector of trained neural network activation weights. This might take the form of a direct injection to LTM in Fig. 2. Indirect transfer implies that some level of abstraction analysis has been applied to the hypothesis representation prior to assignment.

Learning stages – A learning system may implement learning in a single stage or in a series of stages. An example of a two-stage system is one that waits to initiate the long-term storage of domain knowledge until after primary task learning in short-term memory is complete. Like task order, learning stages would be reflected in the sequence of events within and among process arrows in the Fig. 2 architecture. But in this case, ordering pertains to the manner in which learning is staged across encoding processes.

Interface cardinality – The interface cardinality can be fixed or variable. Fixing the number of inputs and outputs has the advantage of providing a consistent interface without posing restrictions on the growth of the internal representation.

Data type – The input and output data types can be fixed or variable. A type-flexible system can produce both categorical and scalar predictions.

Scalability – CL systems may or may not scale on a variety of dimensions including inputs, outputs, training examples, and tasks.

Knowledge

This category pertains to the long-term storage of learned knowledge. Thus, the following CL dimensions characterize knowledge representation, storage, and retrieval.

Knowledge representation – Stored knowledge can manifest as functional or representational. Functional knowledge retention involves the storage of specific exemplars or parameter values, which tends to be more accurate, whereas representational knowledge retention involves the storage of hypotheses derived from training on exemplars, which has the advantage of storage economy.

Retention efficacy – The efficacy of long term retention varies across CL systems. Effective retention implies that only domain knowledge with an acceptable level of accuracy is retained so that errors aren't propagated to future hypotheses. A related consideration is whether or not the consolidation of new domain knowledge degrades the accuracy of current or prior hypotheses.

Retention efficiency – The retention efficiency of long term memory can vary according to both economy of representation and computationally efficiency.

Indexing method – The input to the comparison process used to select appropriate knowledge for biasing new learning may simply be exemplars (as provided by LTM in Fig. 2) or may take a representational form (e.g., a vector of neural network weights).

Indexing efficiency – CL systems vary in terms of the speed and accuracy with which they can identify related prior knowledge that is suitable for inductive transfer during short term learning. The input to this selection process is the indexing method.

Meta-knowledge – CL systems differentially exhibit the ability to abstract, store, and utilize meta-knowledge, such as characteristics of the input space, learning system parameter values, etc.

Cumulative Learning. Table 1 CL System Dimensions

Category	Dimension	Values (ML3 guidance is indicated by ✓)
Architecture	Learning paradigm	Supervised learning
		Reinforcement learning
		Unsupervised learning
		✓ Hybrid
	Task order	Sequential
		Parallel
		✓ Revisit (practice)
		Hybrid
	Transfer method	Functional
		Representational – direct
		Representational – indirect
	Learning stages	✓ Single (computational retention efficiency)
		Multiple
	Interface cardinality	✓ Fixed
		Variable
	Data type	Fixed
		Variable
	Scalability	✓ Inputs
		✓ Outputs
		✓ Exemplars
		✓ Tasks
Knowledge	Representation	Functional
		Representational – disjoint
		✓ Representational – continuous
	Retention efficacy	✓ Improves prior task performance
		✓ Improves new task performance
	Retention efficiency	✓ Space (memory usage)
		✓ Time (computational processing)
	Indexing method	✓ Deliberative – functional
		✓ Deliberative – representational
		Reflexive

Cumulative Learning. Table 1 (Continued)

Category	Dimension	Values (ML3 guidance is indicated by ✓)
	Indexing efficiency	✓ Time < O(n^c), $c > 1$ (n = tasks)
	Meta-knowledge	✓ Probability distribution of input space
		Learning curve
		Error rate
Learning	Agency	Single learning method
		Task-based selection of learning method
	Utility	Single learning method
		Task-based selection of learning method
	Task awareness	Task boundary identification (begin/end)
	Bias modulation	✓ Estimated sample complexity
		✓ Number of task exemplars
		✓ Generalization accuracy of retained knowledge
		✓ Relatedness of retained knowledge
	Learning efficacy	✓ Generalization | bias ≥ generalization | no bias
	Learning efficiency	✓ Time | bias ≤ time | no bias

Learning

While all of the dimensions listed herein impact learning, the following dimensions correspond to specific learning capabilities or learning performance metrics.

Agency – The agency of a learning system is the degree of sophistication exhibited by its top-level controller. For example a learning system may be on the low end of the agency continuum if it always applies one predetermined learning method to one task or on the high end if it selects among many learning methods as a function of the learning task. One might imagine, for example, two process diagrams such as the one depicted in Fig. 2, that share the same LTM, but are otherwise distinct and differentially activated by a governing controller as a function of qualitative aspects of the input.

Utility – Domain knowledge acquisition can be deliberative in the sense that the learning system decides which hypotheses to incorporate based upon their estimated utility, or reflexive, in which case all hypotheses are stored irrespective of utility considerations.

Task awareness – Task awareness characterizes the system's ability to identify the beginning and end of a new task.

Bias modulation – A CL system may have the ability to determine the extent to which short-term learning would benefit from inductive transfer and, on that basis, assign a relevant weight. The depth of this analysis can vary and might consider factors such as the estimated sample complexity, number of exemplars, the generalization accuracy of retained knowledge, and relatedness of retained knowledge.

Learning efficacy – A measure of learning efficacy is derived by comparing generalization performance in the presence and absence of an inductive bias. Learning is considered effective when the application of an inductive bias results in greater generalization performance on the primary task than when the bias is absent.

Learning efficiency – Similarly, learning efficiency is assessed by comparing the computational time needed to generate a hypothesis in the presence and absence of an inductive bias. Lower computational time in the presence of bias signifies greater learning efficiency.

The Research Space

Table 1 summarizes the classification dimensions, providing an overview of the research space, an evaluative framework for assessing and contrasting CL approaches, and a generative framework for identifying new areas of exploration. In addition, checked items in the Values column indicate ML3 guidance. Specifically, an ideal ML3 system would correspond functionally to the called-out items and performance criteria. However, Silver and Poirier (2005) allude to the fact that it would be nigh impossible to generate a strictly compliant ML3 system since some of the recommended criteria do not coexist easily. For example, effective and efficient learning are mutually incompatible because they require different forms of knowledge transfer. Nonetheless, a CL system that falls within scope of the majority of the ML3 criteria would be well-positioned to exhibit lifelong learning behavior.

Future Directions

Emergent work (Oblinger, 2006; Swarup, Lakkaraju, Ray, & Gasser, 2006) in instructable computing has given rise to a new CL paradigm that is largely ML3 compliant and involves high degrees of task awareness and agency sophistication. Swarup et al. (2006) describe an approach in which domain knowledge is represented in the form of structured graphs. Short term (primary task) learning occurs via a genetic algorithm, after which domain knowledge is extracted by mining frequent subgraphs. The accumulated domain knowledge forms an ontology to which the learning system grounds symbols as a result of structured interactions with instructional agents. Subsequent interactions occur using the symbol system as a shared lexicon for communication between the instructor and the learning system. Knowledge acquired from these interactions bootstrap future learning.

The Bootstrapped Learning framework proposed by Oblinger (2006) provides for hierarchical, domain-independent learning that, like the effort described above, is also premised on a model of building concepts from structured lessons. In this case, however, there is no a priori knowledge acquisition. Instead, some "common" knowledge about the world is provided explicitly to the learning system, and then lessons are taught by a human teacher using the same natural instruction methods that would be used to teach another human. Rather than requiring a specific learning algorithm, this framework provides a context for evaluating and comparing learning algorithms. It includes a knowledge representation language that supports syntactic, logical, procedural, and functional knowledge, an interaction language for communication among the learning system, instructor, and environment, and an integration architecture that evaluates, processes, and responds to interaction language communiqués in the context of existing knowledge and through the selective utilization of available learning algorithms.

The learning performance advantages anticipated by these proposals for instructional computing seem to stem from the economy of representation afforded by hierarchical knowledge combined with the tremendous learning bias imposed by explicit instruction.

Recommended Reading

Abu-Mostafa, Y. (1990). Learning from hints in neural networks (invited). *Journal of Complexity*, *6*(2), 192–198.

Banerji, R. B. (1964). A Language for the Description of Concepts. *General Systems*, *9*, 135–141.

Baxter, J. (1995). Learning internal representations. In *(COLT): Proceeding of the workshop on computational learning theory*, Santa Cruz, California. Morgan Kaufmann.

Brazdil P., Giraud-Carrier, C., Soares, C., & Vilalta, R. (2009). *Metalearning – Applications to Data Mining*, Springer.

Caruana, R. (1993). Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of the tenth international conference on machine learning*, University of Massachusetts, Amherst (pp. 41–48).

Caruana, R. (1996). Algorithms and applications for multitask learning. In *Machine learning: Proceedings of the 13th international conference on machine learning (ICML 1996)*, Bari, Italy (pp. 87–95). Morgan Kauffmann.

Cohen, B. L. (1978). *A Theory of Structural Concept Formation and Pattern Recognition*. Ph.D. Thesis, Department of Computer Science, The University of New South Wales.

Cohen, B. L., & Sammut, C. A. (1982). Object Recognition and Concept Learning with CONFUCIUS. *Pattern Recognition Journal*, *15*(4), 309–316.

Mitchell, T. (1980). *The need for biases in learning generalizations*. Rutgers TR CBM-TR-117.

Mitchell, T. M., & Thrun, S. B. (1993). Explanation-based neural network learning for robot control. In Hanson, Cowan, &

Giles (Eds.), *Advances in neural information processing systems 5* (pp. 287–294). San Francisco, CA: Morgan-Kaufmann.
Nilsson, N. J. (1996). *Introduction to machine learning: An early draft of a proposed textbook* (p. 12). Online at http://ai.stanford.edu/$\sim$nilsson/MLBOOK.pdf. Accessed on July 22, 2010.
Oblinger, D. (2006). *Bootstrapped learning proposer information pamphlet* for broad agency announcement 07-04. Online at http://fs1.fbo.gov/EPSData/ODA/Synopses/4965/BAA07-04/BLPIPfinal.pdf.
Pratt, L. Y., Mostow, J., & Kamm, C. A. (1991). Direct transfer of learned information among neural networks. In *Proceedings of the ninth national conference on artificial intelligence (AAAI-91)*, Anaheim, CA (pp. 584–589).
Ring, M. (1991). Incremental development of complex behaviors through automatic construction of sensory-motor hierarchies. In *Proceedings of the eighth international workshop (ML91)*, San Mateo, California.
Silver, D., & Mercer, R. (2002). The task rehearsal method of life-long learning: Overcoming impoverished data. In R. Cohen & B. Spencer (Eds.), *Advances in artificial intelligence, 15th conference of the Canadian society for computational studies of intelligence (AI 2002)*, Calgary, Canada, May 27–29, 2002. *Lecture notes in computer science* (Vol. 2338, pp. 90–101). London: Springer.
Silver, D., & Poirier, R. (2005). Requirements for machine life-long learning. JSOCS Technical Report TR-2005-009, Acadia University.
Swarup, S., Lakkaraju, K., Ray, S. R., & Gasser, L. (2006). Symbol grounding through cumulative learning. In P. Vogt et al. (Eds.), *Symbol grounding and beyond: Proceedings of the third international workshop on the emergence and evolution of linguistic communication*, Rome, Italy (pp. 180–191). Berlin: Springer.
Swarup, S., Mahmud, M. M. H., Lakkaraju, K., & Ray, S. R. (2005). Cumulative learning: Towards designing cognitive architectures for artificial agents that have a lifetime. Tech. Rep. UIUCDCS-R-2005-2514.
Thrun, S. (1998). Lifelong learning algorithms. In S. Thrun & L. Y. Pratt (Eds.), *Learning to learn*. Norwell, MA: Kluwer Academic.
Thrun, S., & Mitchell, T. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, *15*, 25–46.
Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind Mind*, *59*(236), 433–460.
Vilalta, R., & Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial Intelligence Review*, *18*, 77–95.

Curse of Dimensionality

Eamonn Keogh, Abdullah Mueen
University California-Riverside,
Riverside, CA, USA

Definition

The curse of dimensionality is a term introduced by Bellman to describe the problem caused by the exponential increase in volume associated with adding extra dimensions to Euclidean space (Bellman, 1957).

For example, 100 evenly-spaced sample points suffice to sample a unit interval with no more than 0.01 distance between points; an equivalent sampling of a 10-dimensional unit hypercube with a grid with a spacing of 0.01 between adjacent points would require 10^{20} sample points: thus, in some sense, the 10D hypercube can be said to be a factor of 10^{18} "larger" than the unit interval.

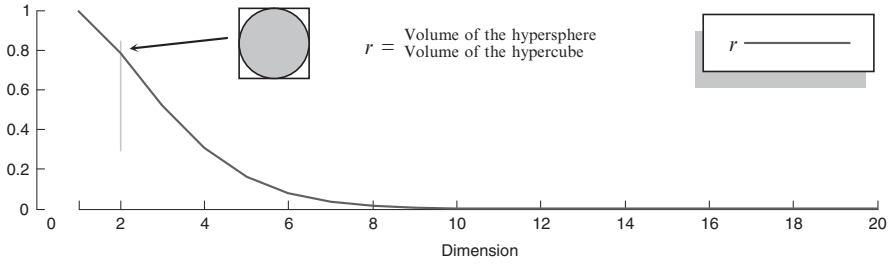
Informally, the phrase *curse of dimensionality* is often used to simply refer to the fact that one's intuitions about how data structures, similarity measures, and algorithms behave in low dimensions do typically generalize well to higher dimensions.

Background

Another way to envisage the vastness of high-dimensional Euclidean space is to compare the size of the unit sphere with the unit cube as the dimension of the space increases: as the dimension increases. As we can see in Fig. 1, the unit sphere becomes an insignificant volume relative to that of the unit cube. In other words, almost all of the high-dimensional space is *far away* from the center.

In research papers, the phrase *curse of dimensionality* is often used as shorthand for one of its many implications for machine learning algorithms. Examples of these implications include:

- ▶Nearest neighbor searches can be made significantly faster for low-dimensional data by indexing the data with an R-tree, a KD-tree, or a similar spatial access method. However, for high-dimensional data all such methods degrade to the performance of a simple linear scan across the data.
- For machine learning problems, a small increase in dimensionality generally requires a large increase in the numerosity of the data, in order to keep the same level of performance for regression, clustering, etc.
- In high-dimensional spaces, the normally intuitive concept of proximity or similarity may not be qualitatively meaningful. This is because the ratio of an object's nearest neighbor over its farthest neighbor approaches one for high-dimensional spaces (Aggarwal, Hinneburg, & Keim, 2001). In other



Curse of Dimensionality. Figure 1. The ratio of the volume of the hypersphere enclosed by the unit hypercube. The most intuitive example, the unit square and unit circle, are shown as an inset. Note that the volume of the hypersphere quickly becomes irrelevant for higher dimensionality

words, all objects are approximately equidistant from each other.

There are many ways to attempt to mitigate the *curse of dimensionality*, including ►feature selection and ►dimensionality reduction. However, there is no single solution to the many difficulties caused by the effect.

Recommended Reading

The major database (SIGMOD, VLDB, PODS), data mining (SIGKDD, ICDM, SDM), and machine learning (ICML, NIPS) conferences typically feature several papers which explicitly address the *curse of dimensionality* each year.

Aggarwal, C. C., Hinneburg, A., & Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional spaces. In *ICDT* (pp. 420–434). London, England.

Bellman, R. E. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.

Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., & Keogh, E. (2008). Querying and mining of time series data: Experimental comparison of representations and distance measures. In *Proceedings of the VLDB endowment* (Vol. 1, pp. 1542–1552). Auckland, NewZealand.

D

Data Mining On Text

►Text Mining

Data Preparation

Geoffrey I. Webb
Monash University, Victoria, Australia

Synonyms

Data preprocessing; Feature construction

Definition

Before data can be analyzed, they must be organized into an appropriate form. Data preparation is the process of manipulating and organizing data prior to analysis.

Motivation and Background

Data are collected for many purposes, not necessarily with machine learning in mind. Consequently, there is often a need to identify and extract relevant data for the given analytic purpose. Every learning system has specific requirements about how data must be presented for analysis and hence, data must be transformed to fulfill those requirements. Further, the selection of the specific data to be analyzed can greatly affect the models that are learned. For these reasons, data preparation is a critical part of any machine learning exercise. Data preparation is often the most time-consuming part of any nontrivial machine learning project.

Processes and Techniques

The manner in which data are prepared varies greatly depending upon the analytic objectives for which they are required and the specific learning techniques and software by which they are to be analyzed. The following are a number of key processes and techniques.

Sourcing, Selecting, and Auditing Appropriate Data

It is necessary to review the data that are already available, assess their suitability to the task at hand, and investigate the feasibility of sourcing new data collected specifically for the desired task.

Much of the theory on which learning systems are based assumes that the training data are a random sample of the population about which the user wishes to learn a model. However, much historical data represent biased samples, for example, data that have been easy to collect or that have been considered interesting for some other purpose. It is desirable to consider whether the available data are sufficiently representative of the future data to which a learned model is to be applied.

It is important to assess whether there is sufficient data to realistically obtain the desired machine learning outcomes.

Data quality should be investigated. Much data is of low quality. Those responsible for manual data collection may have little commitment to assuring data accuracy and may take shortcuts in data entry. For example, when default values are provided by a system, these tend to be substantially overrepresented in the collected data. Automated data collection processes might be faulty, resulting in inaccurate or incorrect data. The precision of a measuring instrument may be lower than desirable. Data may be out of date and no longer correct.

Where the data contain ►noise, it may be desirable to identify and remove outliers and other suspect data points or take other remedial action.

Existing data may be augmented through data enrichment. This commonly involves sourcing of

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

additional information about the data points on which data are already held. For example, customer data might be enriched by purchasing socioeconomic data about individual customers.

Transforming Representation

It may be necessary to frequently transform data from one representation to another. Reasons for doing so include highlighting relevant distinctions and formatting data to satisfy the requirements of a specific learner.

►Discretization is a process whereby quantitative data are transformed into qualitative.

Some systems cannot process multi-valued categorical variables. This limitation can be circumvented by converting a multi-valued categorical variable into multiple binary variables, one new variable to represent the presence or absence of each value of the original variable. Conversely, multiple mutually exclusive binary variables might be converted into a single multi-valued categorical variable.

Some systems require the input to be numeric. Categorical variables must be converted into numeric form. Multi-valued categorical variables should usually be converted into multiple binary variables before conversion to numbers, as projecting unordered values onto a linear scale can greatly distort analytic outcomes.

It is important to select appropriate levels of granularity for analysis. For example, when distinguishing products, should a gallon of low fat milk be described as a diary product, and hence not distinguished from any other type of dairy product, be described as low fat milk, and hence not distinguished from other brands and quantities, or uniquely distinguished from all other products. Analysis at the lowest level of granularity makes possible identification of potentially valuable fine-detail regularities in the data, but may make it more difficult to identify high-level relationships.

It is often desirable to create derived values. For example, the available data might contain fields for purchase price, costs, and sale price. The relevant quantity for analysis might be profit, which must be computed from the raw data. The creation of new features is called *feature construction*.

As many learning systems have difficulty with high dimension data, it may be desirable to project the data onto a lower dimensional space. Popular approaches to doing so include ►Principal Components Analysis and ►Kernel Methods.

Another approach to reducing dimensionality is to select only a subset of the available features (see ►Feature Selection).

It is important to determine whether the data have ►Missing Values and, if so, to ensure that appropriate measures are taken to allow the learning system to handle this situation.

►Propositionalization. Some data sets contain information expressed in a relational form, i.e., describing relationships between objects in the world. While some learning systems can accept relations directly, most operate only on attribute-value representations. Therefore, a relational representation must be reexpressed in attribute-value form. In other words, a representation equivalent to first-order logic must be converted to a representation equivalent only to propositional logic.

Cross References

►Data Set
►Discretization
►Entity Resolution
►Evolutionary Feature Selection and Construction
►Feature Construction in Text Mining
►Feature Selection
►Feature Selection in Text Mining
►Kernel Methods
►Measurement Scales
►Missing Values
►Noise
►Principal Component Analysis
►Propositionalization

Recommended Reading

Pyle, D. (1999). *Data preparation for data mining*. San Francisco, Morgan Kaufmann.

Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco, Morgan Kaufmann.

Data Preprocessing

►Data Preparation

Data Set

A data set is a collection of data used for some specific machine learning purpose. A ►training set is a data set that is used as input to a ►learning system, which analyzes it to learn a ►model. A ►test set or ►evaluation set is a data set containing data that are used to ►evaluate the model learned by a learning system. A training set may be divided further into a ►growing set and a ►pruning set. Where the training set and the test set contain disjoint sets of data, the test set is known as a ►holdout set.

DBN

Dynamic Bayesian Network. See ►Learning Graphical Models

Decision Epoch

In a ►Markov decision process, *decision epochs* are sequences of times at which the decision-maker is required to make a decision. In a discrete time Markov decision process, decision epochs occur at regular, fixed intervals, whereas in a continuous time Markov decision process (or semi-Markov decision process), they may occur at randomly distributed intervals.

Decision List

Johannes Fürnkranz
Fachbereich Informatik, Darmstadt, Germany

Synonyms

Ordered rule set

Definition

A decision list (also called an ordered rule set) is a collection of individual Classification Rules that collectively form a Classifier. In contrast to an unordered Rule Set, decision lists have an inherent order, which makes classification quite straightforward. For classifying a new instance, the rules are tried in order, and the class of the first rule that covers the instance is predicted. If no induced rule fires, a *default rule* is invoked, which typically predicts the majority class.

Typically, decision lists are learned with a ►Covering Algorithm, which learns one rule at a time, appends it to the list, and removes all covered examples before learning the next one. Decision lists are popular in ►Inductive Logic Programming, because PROLOG programs may be considered to be simple decision lists, where all rules predict the same concept.

A formal definition of decision lists, a comparison of their expressiveness to decision trees and rule sets in disjunctive and conjunctive normal form, as well as theoretical results on the learnability of decision lists can be found in Rivest (1987).

Cross References

►Classification Rule
►Disjunctive Normal Form
►Rule Learning

Recommended Reading

Rivest, R.L. (1987). Learning decision lists. *Machine Learning, 2*, 229–246.

Decision Lists and Decision Trees

Johannes Fürnkranz
Fachbereich Informatik, Darmstadt, Germany

Definition

►Decision Trees and ►Decision Lists are two popular ►Hypothesis Languages, which share quite a few similarities. The key difference is that decision trees may be viewed as unordered Rule Sets, where each leaf of the tree corresponds to a single rule with a condition part consisting of the conjunction of all edge labels on the path from the root to this leaf. The hierarchical structure of the tree ensures that the rules in the set are nonoverlapping, that is, each example can only be covered by a single rule. This additional constraint makes

classification easier (no conflicts from multiple rules), but may result in more complex rules. For example, it has been shown that decision lists (ordered rule sets) with at most k conditions per rule are strictly *more expressive* than decision trees of depth k (Rivest, 1987). A similar result has been proved in Boström (1995).

Moreover, the restriction of decision tree learning algorithms to nonoverlapping rules imposes strong constraints on learnable rules. One problem resulting from this constraint is the *replicated subtree problem* (Pagallo and Haussler 1990); it often happens that identical subtrees have to be learned at various places in a decision tree, because of the fragmentation of the example space imposed by the restriction to nonoverlapping rules. Rule learners do not make such a restriction and are thus less susceptible to this problem. An extreme example for this problem has been provided by Cendrowska (1987), who showed that the minimal decision tree for the concept `x` defined as

```
IF A = 3 AND B = 3 THEN Class = x
IF C = 3 AND D = 3 THEN Class = x
```

has 10 interior nodes and 21 leafs assuming that each attribute `A` ... `D` can be instantiated with three different values.

On the other hand, a key advantage of decision tree learning is that not only a single rule is optimized, but that conditions are selected in a way that simultaneously optimizes the example distribution in all successors of a node. Attempts to adopt this property for rule learning have given rise to several hybrid systems, the best known being PART (Frank & Witten, 1998), which learns a decision list of rules, each one being the single best rule of a separate decision tree. This rule can be efficiently found without learning the full tree, by repeated expansion of its most promising branch. Similarly, pruning algorithms can be used to convert decision trees into sets of nonoverlapping rules (Quinlan, 1987).

Cross References

►Covering Algorithm
►Decision Trees
►Divide-and-Conquer Learning
►Rule Learning

Recommended Reading

Boström, H. (1995). Covering vs. divide-and-conquer for top-down induction of logic programs. In *Proceedings of the 14th international joint conference on artificial intelligence (IJCAI-95)*, (pp. 1194–1200). Montreal, Canada.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies, 27*, 349–370.

Frank, E., & Witten, I. H. (1998). Generating accurate rule sets without global optimization. In J. Shavlik (Ed.), *Proceedings of the 15th international conference on machine learning (ICML-98), Madison, Wisconsin* (pp. 144–151). San Francisco, CA. Morgan Kaufmann.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning, 5*, 71–99.

Quinlan, J. R. (1987). Generating production rules from decision trees. In *Proceedings of the tenth international joint conference on artificial intelligence (IJCAI-87)*, (pp. 304–307). Morgan Kaufmann, Milan, Italy.

Rivest, R. L. (1987). Learning decision lists. *Machine Learning, 2*, 229–246.

Decision Rule

A decision rule is an element (piece) of knowledge, usually in the form of a "if-then statement":

if < Condition > then < Action >

If its Condition is satisfied (i.e., matches a fact in the corresponding database of a given problem) then its Action (e.g., classification or decision making) is performed. See also ►Markovian Decision Rule.

Decision Stump

Definition

A *decision stump* is a ►Decision Tree, which uses only a single attribute for splitting. For discrete attributes, this typically means that the tree consists only of a single interior node (i.e., the root has only leaves as successor nodes). If the attribute is numerical, the tree may be more complex.

Decision stumps perform surprisingly well on some commonly used benchmark datasets from the ►UCI repository (Holte, 1993), which illustrates that learners with a high ►Bias and low ►Variance may perform well because they are less prone to ►Overfitting. Decision stumps are also often used as weak learners

in ▶Ensemble Methods such as boosting (Freund & Schapire, 1996).

Cross References

▶Bias and Variance
▶Decision Tree
▶Overfitting

Recommended Reading

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In L. Saitta (Ed.), *Proceedings of the 13th international conference on machine learning; Bari, Italy* (pp. 148–156). San Francisco: Morgan Kaufmann.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning, 11*, 63–91.

Decision Threshold

The decision threshold of a binary classifier that outputs scores, such as ▶decision trees or ▶naive Bayes, is the value above which scores are interpreted as positive classifications. Decision thresholds can be either fixed if the classifier outputs calibrated scores on a known scale (e.g., 0.5 for a probabilistic classifier), or learned from data if the scores are uncalibrated. See ▶ROC Analysis.

Decision Tree

Johannes Fürnkranz
Fachbereich Informatik
Darmstadt
Germany

Synonyms

C4.5; CART; Classification tree

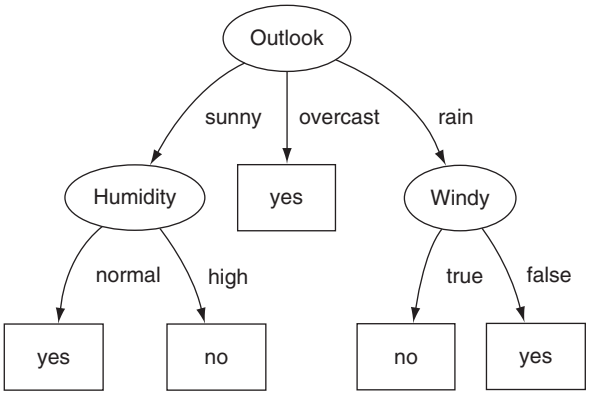
Definition

A *decision tree* is a tree-structured ▶classification ▶model, which is easy to understand, even by nonexpert users, and can be efficiently induced from data. The induction of decision trees is one of the oldest and most popular techniques for learning discriminatory models, which has been developed independently in the statistical (Breiman, Friedman, Olshen, & Stone, 1984; Kass, 1980) and machine learning (Hunt, Marin, & Stone, 1966; Quinlan, 1983, 1986) communities. An extensive survey of decision tree learning can be found in Murthy (1998).

Representation

Figure 1 shows a sample decision tree for a well-known sample dataset, in which examples are descriptions of weather conditions (*Outlook, Humidity, Windy, Temperature*), and the target concept is whether these conditions are suitable for playing golf or not (Quinlan, 1986). Classification of a new example starts at the top node—the *root*—and the value of the attribute that corresponds to this node is considered (*Outlook* in the example). The example is then moved down the branch that corresponds to a particular value of this attribute, arriving at a new node with a new attribute. This process is repeated until one arrives at a terminal node—a so-called *leaf*—which is not labeled with an attribute but with a value of the target attribute (*PlayGolf?*). For all examples that arrive at the same leaf, the same target value will be predicted. Figure 1 shows leaves as rectangular boxes.

Note that some of the attributes may not occur at all in the tree. For example, the tree in Fig. 1 does not contain a test on *Temperature* because the training data can be classified without making a reference to this variable. More generally, one can say that the attributes in



Decision Tree. Figure 1. A decision tree describing the Golf dataset (Quinlan, 1986)

the upper parts of the tree (near the root) tend to have a stronger influence on the value of the target variable than the nodes in the lower parts of the tree (e.g., *Outlook* will always be tested, whereas *Humidity* and *Windy* will only be tested under certain conditions).

Learning Algorithm

Decision trees are learned in a top-down fashion, with an algorithm known as *Top-Down Induction of Decision Trees (TDIDT)*, *recursive partitioning*, or *divide-and-conquer* learning. The algorithm selects the best attribute for the root of the tree, splits the set of examples into disjoint sets, and adds corresponding nodes and branches to the tree. The simplest splitting criterion is for discrete attributes, where each test has the form $t \leftarrow (A = v)$ where v is one possible value of the chosen attribute A. The corresponding set S_t contains all training examples for which the attribute A has the value v. This can be easily adapted to numerical attributes, where one typically uses binary splits of the form $t \leftarrow (A < v_t)$, which indicate whether the attribute's value is above or below a certain threshold value v_t. Alternatively, one can transform the data before-hand using a ▶Discretization algorithm.

---

function TDIDT(S)

Input: S, a set of labeled examples.

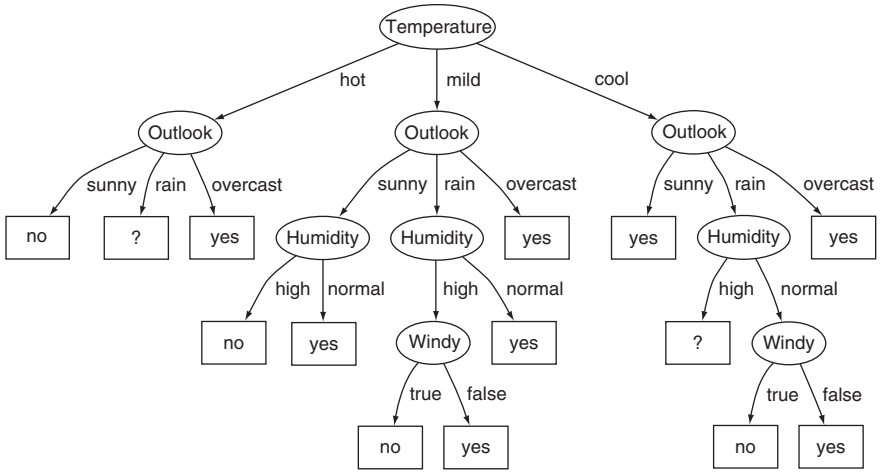
Tree = new empty node
if all examples have the same class c
  or no further splitting is possible
then // new leaf
  LABEL(*Tree*) = c
else // new decision node
  (A, T) = FINDBESTSPLIT(S)
  for each test $t \in T$ **do**
    S_t = all examples that satisfy t
    $Node_t$ = TDIDT(S_t)
    ADDEDGE($Tree \xrightarrow{t} Node_t$)
  endfor
endif
return *Tree*

After splitting the dataset according to the selected attribute, the procedure is recursively applied to each of the resulting datasets. If a set contains only examples from the same class, or if no further splitting is possible (e.g., because all possible splits have already been exhausted or all remaining splits will have the same outcome for all examples), the corresponding node is turned into a leaf node and labeled with the respective class. For all other sets, an interior node is added and associated with the best splitting attribute for the corresponding set as described above. Hence, the dataset is successively partitioned into nonoverlapping, smaller datasets until each set only contains examples of the same class (a so-called *pure* node). Eventually, a pure node can always be found via successive partitions unless the training data contains two identical but contradictory examples, that is, examples with the same feature values but different class values.

Attribute Selection

The crucial step in decision tree induction is the choice of an adequate attribute. In the sample tree of Fig. 2, which has been generated from the same 14 training examples as the tree of Fig. 1, most leaves contain only a single training example, that is, with the selected splitting criteria, the termination criterion (all examples of a node have to be of the same class) could, in many cases, only trivially be satisfied (only one example remained in the node). Although both trees classify the training data correctly, the former appears to be more trustworthy, and in practice, one can often observe that simpler trees are more accurate than more complex trees. A possible explanation could be that labels that are based on a higher number of training examples tend to be more reliable. However, this preference for simple models is a heuristic criterion known as ▶Occam's Razor, which appears to work fairly well in practice, but is still the subject of ardent debates within the machine learning community.

Typical attribute selection criteria use a function that measures the *impurity* of a node, that is, the degree to which the node contains only examples of a single class. Two well-known impurity measures are the information-theoretic entropy (Quinlan, 1986), and the



Decision Tree. Figure 2. A needlessly complex decision tree describing the same dataset

Gini index (Breiman et al., 1984) which are defined as

$$\text{Entropy}(S) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \cdot \log_2 \left(\frac{|S_i|}{|S|} \right)$$

$$\text{Gini}(S) = 1 - \sum_{i=1}^{c} \left(\frac{|S_i|}{|S|} \right)^2$$

where S is a set of training examples, and S_i is the set of training examples that belong to class c_i. Both functions have their maximum at the point where the classes are equally distributed (i.e., where all S_i have the same size, maximum impurity), and their minimum at the point where one S_i contains all examples ($S_i = S$) and all other $S_j, j \neq i$ are empty (minimum impurity).

A good attribute divides the dataset into subsets that are as pure as possible, ideally into sets so that each one only contains examples from the same class. Thus, one wants to select the attribute that provides the highest decrease in average impurity, the so-called *gain*:

$$Gain(S, A) = Impurity(S) - \sum_{t} \frac{|S_t|}{|S|} \cdot Impurity(S_t)$$

where t is one of the tests on attribute A which partitions the set S is into nonoverlapping disjoint subsets S_t, and *Impurity* can be any impurity measure. As the first term, $Impurity(S)$, is constant for all attributes, one can also omit it and directly minimize the average impurity (which is typically done when *Gini* is used as an impurity measure).

A common problem is that attributes with many values have a higher chance of resulting in pure successor nodes and are, therefore, often preferred over attributes with fewer values. To counter this, the so-called *gain ratio* normalizes the gained entropy with the intrinsic entropy of the split:

$$GainRatio(S, A) = \frac{Gain(S, A)}{\sum_t \frac{|S_t|}{|S|} \cdot \log_2 \left(\frac{|S_t|}{|S|} \right)}$$

A similar phenomenon can be observed for numerical attributes, where the number of possible threshold values determines the number of possible binary splits for this attribute. Numerical attributes with many possible binary splits are often preferred over numerical attributes with fewer splits because they have a higher chance that one of their possible splits fit the data. A discussion of this problem and a proposal for a solution can be found in Quinlan (1996).

Other attribute selection measures, which do not conform to the gain framework laid out above, are also possible, such as CHAID's evaluation with a χ^2 test statistic (Kass, 1980). Experimental comparison of different measures can be found in Buntine and Niblett (1992) and Mingers (1989a).

Thus, the final tree is constructed by a sequence of local choices that each consider only those examples that end up at the node that is currently split. Of course, such a procedure can only find local optima for each node, but cannot guarantee convergence to

a global optimum (the smallest tree). One of the key advantages of this divide-and-conquer approach is its efficiency, which results from the exponential decrease in the quantity of data to be processed at successive depths in the tree.

Overfitting Avoidance

In principle, a decision tree model can be fit to any training set that does not contain contradictions (i.e., there are no examples with identical attributes but different class values). This may lead to ►Overfitting in the form of overly complex trees.

For this reason, state-of-the-art decision tree induction techniques employ various ►Pruning techniques for restricting the complexity of the found trees. For example, C4.5 has a ►pre-pruning parameter m that is used to prevent further splitting unless at least two successor nodes have at least m examples. The *cost-complexity pruning* method used in CART may be viewed as a simple ►Regularization method, where a good choice for the regularization parameter, which trades off the fit of the data with the complexity of the tree, is determined via ►Cross-validation.

More typically, ►post-pruning is used for removing branches and nodes from the learned tree. More precisely, this procedure replaces some of the interior nodes of the tree with a new leaf, thereby removing the subtree that was rooted at this node. An empirical comparison of different decision-tree pruning techniques can be found in Mingers (1989b).

It is important to note that the leaf nodes of the new tree are no longer pure nodes, that is, they no longer need to contain training examples that all belong to the same class. Typically, this is simply resolved by predicting the most frequent class at a leaf. The class distribution of the training examples within the leaf may be used as a reliability criterion for this prediction.

Well-known Decision Tree Learning Algorithms

The probably best-known decision tree learning algorithm is C4.5 (Quinlan, 1993) which is based upon ID3 (Quinlan, 1983), which, in turn, has been derived from an earlier concept learning system (Hunt et al., 1966). ID3 realized the basic recursive partitioning algorithm for an arbitrary number of classes and for discrete attribute values. C4.5 (Quinlan, 1993) incorporates several key improvements that were necessary for tackling real-world problems, including handling of numeric and ►missing attribute values, ►overfitting avoidance, and improved scalability. A C-implementation of C4.5 is freely available from its author. A re-implementation is available under the name J4.8 in the Weka data mining library. C5.0 is a commercial successor of C4.5, distributed by RuleQuest Research. CART (Breiman et al., 1984) is the best-known system in the statistical learning community. It is integrated into various statistical software packages, such as R or S.

Decision trees are also often used as components in ►Ensemble Methods such as random forests (Breiman, 2001) or AdaBoost (Freund & Schapire, 1996). They can also be modified for predicting numerical target variables, in which case they are known as ►Regression Trees. One can also put more complex prediction models into the leaves of a tree, resulting in ►Model Trees.

Cross References

►Decision List
►Decision Lists and Decision Trees
►Decision Stump
►Divide-and-Conquer Learning
►Model Tree
►Pruning
►Regression Tree
►Rule Learning

Recommended Reading

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32.

Breiman, L., Friedman, J. H., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Pacific Grove: Wadsworth & Brooks.

Buntine, W., & Niblett, T. (1992). A further comparison of splitting rules for decision-tree induction. *Machine Learning, 8*, 75–85.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In L. Saitta (Ed), *Proceedings of the 13th international conference on machine learning, Bari, Italy* (pp. 148–156). San Francisco: Morgan Kaufmann.

Hunt, E. B., Marin, J., & Stone, P. J. (1966). *Experiments in induction*. New York: Academic.
Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Applied Statistics, 29*, 119–127.
Mingers, J. (1989a). An empirical comparison of selection measures for decision-tree induction. *Machine Learning, 3*, 319–342.
Mingers, J. (1989b). An empirical comparison of pruning methods for decision tree induction. *Machine Learning, 4*, 227–243.
Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery, 2*(4), 345–389.
Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning. An artificial intelligence approach* (pp. 463–482). Palo Alto: Tioga.
Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.
Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo: Morgan Kaufmann.
Quinlan, J. R. (1996). Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research, 4*, 77–90.

Decision Trees For Regression

►Regression Trees

Deductive Learning

Synonyms

Analytical learning; Explanation-based learning

Definition

Deductive learning is a subclass of machine learning that studies algorithms for learning provably correct knowledge. Typically such methods are used to speedup problem solvers by adding knowledge to them that is deductively entailed by existing knowledge, but that may result in faster solutions.

Deduplication

►Entity Resolution

Deep Belief Nets

Geoffrey Hinton
University of Toronto, Toronto, Canada

Synonyms

Deep belief networks

Definition

Deep belief nets are probabilistic generative models that are composed of multiple layers of stochastic latent variables (also called "feature detectors" or "hidden units"). The top two layers have undirected, symmetric connections between them and form an associative memory. The lower layers receive top-down, directed connections from the layer above. Deep belief nets have two important computational properties. First, there is an efficient procedure for learning the top-down, generative weights that specify how the variables in one layer determine the probabilities of variables in the layer below. This procedure learns one layer of latent variables at a time. Second, after learning multiple layers, the values of the latent variables in every layer can be inferred by a single, bottom-up pass that starts with an observed data vector in the bottom layer and uses the generative weights in the reverse direction.

Motivation and Background

The perceptual systems of humans and other animals show that high-quality pattern recognition can be achieved by using multiple layers of adaptive nonlinear features, and researchers have been trying to understand how this type of perceptual system could be learned, since the 1950s (Selfridge, 1958). Perceptrons (Rosenblatt, 1962) were an early attempt to learn a biologically inspired perceptual system, but they did not have an efficient learning procedure for multiple layers of features. Backpropagation (Rumelhart, Hinton, & Williams, 1986; Werbos, 1974) is a supervised learning procedure that became popular in the 1980s because it provided a fairly efficient way of learning multiple layers of nonlinear features by propagating derivatives of the error in the output backward through the multilayer network. Unfortunately, backpropagation has difficulty

optimizing the weights in deep networks that contain many layers of hidden units and it requires labeled training data, which is often expensive to obtain. Deep belief nets overcome the limitations of backpropagation by using *unsupervised* learning to create layers of feature detectors that model the statistical structure of the input data without using any information about the required output. High-level feature detectors that capture complicated higher-order statistical structure in the input data can then be used to predict the labels.

Structure of the Learning System

Deep belief nets are learned one layer at a time by treating the values of the latent variables in one layer, when they are being inferred from data, as the data for training the next layer. This efficient, greedy learning can be followed by, or combined with, other learning procedures that fine-tune all of the weights to improve the generative or discriminative performance of the whole network. Discriminative fine-tuning can be performed by adding a final layer of variables that represent the desired outputs and backpropagating error derivatives. When networks with many hidden layers are applied in domains that contain highly structured input vectors, backpropagation learning works much better if the feature detectors in the hidden layers are initialized by learning a deep belief net that models the structure in the input data (Hinton & Salakhutdinov, 2006). Matlab code for learning and fine-tuning deep belief nets can be found at http://cs.toronto.edu/~hinton.

Composing Simple Learning Modules

Early deep belief networks could be viewed as a composition of simple learning modules, each of which is a "restricted Boltzmann machine." Restricted Boltzmann machines contain a layer of "visible units" that represent the data and a layer of "hidden units" that learn to represent features that capture higher-order correlations in the data. The two layers are connected by a matrix of symmetrically weighted connections, W, and there are no connections within a layer. Given a vector of activities $\mathbf{v}$ for the visible units, the hidden units are all conditionally independent so it is easy to sample a vector, $\mathbf{h}$, from the posterior distribution over hidden vectors, $p(\mathbf{h}|\mathbf{v},\mathbf{W})$. It is also easy to sample from $p(\mathbf{v}|\mathbf{h},\mathbf{W})$. By starting with an observed data vector on the visible units and alternating several times between sampling from $p(\mathbf{h}|\mathbf{v},\mathbf{W})$ and $p(\mathbf{v}|\mathbf{h},\mathbf{W})$, it is easy to get a learning signal which is simply the difference between the pairwise correlations of the visible and hidden units at the beginning and end of the sampling (see Chapter Boltzmann Machines for details).

The Theoretical Justification of the Learning Procedure

The key idea behind deep belief nets is that the weights, $\mathbf{W}$, learned by a restricted Boltzmann machine define both $p(\mathbf{v}|\mathbf{h},\mathbf{W})$ and the prior distribution over hidden vectors, $p(\mathbf{h}|\mathbf{W})$, so the probability of generating a visible vector, $\mathbf{v}$, can be written as:

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{W})p(\mathbf{v}|\mathbf{h},\mathbf{W}) \quad (1)$$

After learning $\mathbf{W}$, we keep $p(\mathbf{v}|\mathbf{h},\mathbf{W})$ but we replace $p(\mathbf{h}|\mathbf{W})$ by a better model of the *aggregated* posterior distribution over hidden vectors – i.e., the nonfactorial distribution produced by averaging the factorial posterior distributions produced by the individual data vectors. The better model is learned by treating the hidden activity vectors produced from the training data as the training data for the next learning module. Hinton, Osindero, and Teh (2006) show that this replacement improves a variational lower bound on the probability of the training data under the composite model.

Deep Belief Nets with Other Types of Variable

Deep belief nets typically use the logistic function $y = 1/(1+\exp(-x))$ of the weighted input, x, received from above or below to determine the probability that a binary latent variable has a value of 1 during top-down generation or bottom-up inference. Other types of variable within the exponential family, such as Gaussian, Poisson, or multinomial can also be used (Movellan & Marks, 2001; Welling, Rosen-Zvi, & Hinton, 2005) and the variational bound still applies. However, networks with multiple layers of Gaussian or Poisson units are difficult to train and can become unstable. To avoid these problems, the function $\log(1+\exp(x))$ can be used as a smooth approximation to a rectified linear unit. Units of this type often learn features that are easier to interpret than those learned by logistic units. $\log(1+\exp(x))$

is not in the exponential family, but it can be approximated very accurately as a sum of a set of logistic units that all share the same weight vector and adaptive bias term, but differ by having offsets to the shared bias of −0.5, −1.5, −2.5, ...

Using Autoencoders as the Learning Module

A closely related approach that is also called a "deep belief net" uses the same type of greedy, layer-by-layer learning with a different kind of learning module – an "autoencoder" that simply tries to reproduce each data vector from the feature activations that it causes (Bengio, Lamblin, Popovici, & Larochelle, 2007; Hinton, 1989; LeCun & Bengio, 2007). However, the variational bound no longer applies, and an autoencoder module is less good at ignoring random noise in its training data (Larochelle, Erhan, Courville, Bergstra, & Bengio, 2007).

Applications of Deep Belief Nets

Deep belief nets have been used for generating and recognizing images (Bengio, et al., 2007; Hinton et al., 2006; Ranzato, Huang, Boureau, & LeCun, 2007), video sequences (Sutskever & Hinton, 2007), and motion-capture data (Taylor, Hinton, & Roweis, 2007). If the number of units in the highest layer is small, deep belief nets perform nonlinear dimensionality reduction (Hinton & Salakhutdinov, 2006), and by pretraining each layer separately it is possible to learn very deep autoencoders that can then be fine-tuned with back-propagation (Hinton & Salakhutdinov, 2006). Such networks cannot be learned in reasonable time using back-propagation alone. Deep autoencoders learn compact representations of their input vectors that are much better than those found by linear methods such as Principal Components Analysis, and if the highest level code is forced to be binary, they allow extremely fast retrieval of documents or images (Salakhutdinov & Hinton, 2007; Torralba, Fergus, & Weiss, 2008).

Recommended Reading

Bengio, Y., Lamblin, P., Popovici, P., & Larochelle, H. (2007). Greedy layer-wise training of deep networks, In *Advances in neural information processing systems* (Vol. 19). Cambridge, MA: MIT Press.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence, 40*(1–3), 185–234.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*, 1527–1554.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*, 504–507.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on machine learning*. New York: ACM.

LeCun, Y., & Bengio, Y. (2007). Scaling learning algorithms towards AI. In L. Bottou et al. (Eds.), *Large-scale kernel machines*. MA: MIT Press.

Movellan, J. R., & Marks, T. K. (2001). Diffusion networks, product of experts, and factor analysis.

Ranzato, M., Huang, F. J., Boureau, Y., & LeCun, Y. (2007) Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proceedings of computer vision and pattern recognition conference (CVPR 2007)*. Minneapolis, MN.

Rosenblatt, F. (1962). *Principles of neurodynamics*. Washington, DC: Spartan Books.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533-536.

Salakhutdinov, R. R., & Hinton, G. E. (2007). Semantic hashing. In *Proceedings of the SIGIR workshop on information retrieval and applications of graphical models*. Amsterdam, the Netherlands.

Selfridge, O. G. (1958) Pandemonium: A paradigm for learning. In *Mechanisation of though processes: Proceedings of a symposium held at the National Physical Laboratory*. London: HMSO.

Sutskever, I., & Hinton, G. E. (2007). Learning multilevel distributed representations for high-dimensional sequences. In *Proceedings of the eleventh international conference on artificial intelligence and statistics, San Juan,* Puerto Rico.

Taylor, G. W., Hinton, G. E., & Roweis, S. (2007). Modeling human motion using binary latent variables. In *Advances in neural information processing systems* (Vol. 19). Cambridge, MA: MIT Press.

Torralba, A., Fergus, R., & Weiss, Y. (2008). Small codes and large image databases for recognition. In *IEEE conference on computer vision and pattern recognition* (pp. 1–8). Anchorage, AK.

Welling, M., Rosen-Zvi, M., & Hinton, G. E. (2005). Exponential family harmoniums with an application to information retrieval. In *Advances in neural information processing systems* (Vol. 17, pp. 1481–1488). Cambridge, MA: MIT Press.

Werbos, P. (1974). *Beyond Regression: new tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, Cambridge, MA.

Deep Belief Networks

►Deep Belief Nets

Density Estimation

Claude Sammut
University of New South Wales, Sydney, Australia

Synonyms

Kernel density estimation

Definition

Given a set of observations, $x_1, \ldots, x_N$, which is a random sample from a probability density function $f_X(x)$, density estimation attempts to approximate $f_X(x)$ by $\widehat{f}_X(x_0)$.

A simple way of estimating a probability density function is to plot a histogram from a random sample drawn from the population. Usually, the range of data values is subdivided into equally sized intervals or *bins*. How well the histogram estimates the function depends on the bin width and the placement of the boundaries of the bins. The latter can be somewhat improved by modifying the histogram so that fixed boundaries are not used for the estimate. That is, the estimate of the probability density function at a point uses that point as the centre of a neighborhood. Following Hastie, Tibshirani and Friedman (2009), the estimate can be expressed as:

$$\widehat{f}_X(x_0) = \frac{\#x_i \in N(x_0)}{N\lambda} \quad (1)$$

where $x_1, \ldots, x_N$ is a random sample drawn from a probability density function $f_X(x)$ and $\widehat{f}_X(x_0)$ is the estimate of f_X at point x_0. $N(x_0)$ is a neighborhood of width λ, around x_0. That is, the estimate is the normalized count of the number of values that fall within the neighborhood of x_0.

The estimate above is still *bumpy*, like the histogram. A smoother approximation can be obtained by using a *kernel function*. Each x_i in the sample is associated with a kernel function, usually Gaussian. The count in formula (1) above is replaced by the sum of the kernel function applied to the points in the neighborhood of x_0:

$$\widehat{f}_X(x_0) = \frac{1}{N\lambda}\sum_{i=1}^{N} K_\lambda(x_0, x_i) \quad (2)$$

where K is the kernel function associated with sample x_i near x_0. This is called the *Parzen* estimate (Parzen, 1962). The *bandwidth*, λ, affects the roughness or smoothness of the kernel histogram. The kernel density estimate is said to be under-smoothed if the bandwidth is too small. The estimate is over-smoothed if the bandwidth is too large.

Density estimation is most often used in association with memory-based classification methods, which can be thought of as weighted ▶nearest neighbor classifiers.

▶Mixture models and ▶Locally weighted regression are forms of kernel density estimation.

Cross References

▶Kernel Methods
▶Locally Weighted Regression for Control
▶Mixture Models
▶Nearest Neighbor
▶Support Vector Machine

Recommended Reading

Kernel Density estimation is well covered in texts including Hastie, Tibshirani and Friedman (2009), Duda, Hart and Stork (2001) and Ripley (Ripley, 1996).

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference and perception* (2nd ed.). New York: Springer.

Parzen, E. (1962). On the estimation of a probability density function and the mode. *Annals of Mathematics and Statistics*, *33*, 1065–1076.

Ripley, B. D. (1996). *Pattern recognition and neural networks*. Cambridge: Cambridge University Press.

Density-Based Clustering

Joerg Sander
University of Alberta
Edmonton, AB, Canada

Synonyms

Estimation of density level sets; Mode analysis; Nonparametric cluster analysis

Definition

Density-Based Clustering refers to ▶unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in a data space is a contiguous region of high point density, separated from other such clusters by contiguous regions

of low point density. The data points in the separating regions of low point density are typically considered noise/outliers.

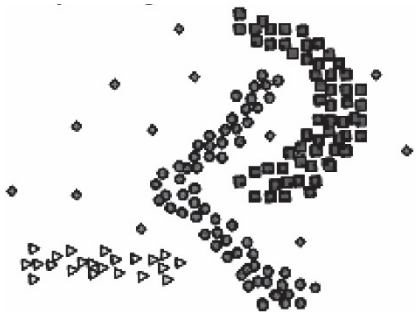
Motivation and Background

Clustering in general is an unsupervised learning task that aims at finding distinct groups in data, called clusters. The minimum requirements for this task are that the data is given as some set of objects O for which a dissimilarity-distance function $d : O \times O \to R^+$ is given. Often, O is a set of d-dimensional real valued points, $O \subset R^d$, which can be viewed as a sample from some unknown probability density $p(x)$, with d as the Euclidean or some other form of distance.

There are different approaches to classifying what characterizes distinct groups in the data.

From a procedural point of view, many clustering methods try to find a partition of the data into k groups, so that within-cluster dissimilarities are minimized while the between-cluster dissimilarities are maximized. The notions of within-cluster dissimilarity and between-cluster dissimilarity are defined using the given distance function d. From a statistical point of view, such methods correspond to a parametric approach, where the unknown density $p(x)$ of the data is assumed to be a mixture of k densities $p_i(x)$, each corresponding to one of the k groups in the data; the $p_i(x)$ are assumed to come from some parametric family (e.g., Gaussian distributions) with unknown parameters, which are then estimated from the data.

In contrast, *density-based* clustering is a nonparametric approach, where the groups in the data are considered to be the high density areas of the density $p(x)$. Density-based clustering methods do not require the number of clusters as input parameters, nor do they make assumptions about the underlying density $p(x)$ or the variance within the groups that may exist in the data. Consequently, density-based clusters are not necessarily groups of points with high within-cluster similarity as measured by the distance function d, but can have an "arbitrary shape" in the feature space; they are sometimes also referred to as "natural clusters." This property makes density-based clustering particularly suitable for applications where clusters cannot be well described as distinct groups of low within-cluster dissimilarity, as, for instance, in spatial data, where clusters of points in the space may form along natural structures such as rivers, roads, and seismic faults. Figure 1 illustrates density-based clusters using a two-dimensional example, where the assumed dissimilarity function between the points is the Euclidean distance: there are three clusters indicated by triangles, points, and rectangles, as well as some noise points, indicated by diamond shapes. Note that the distance between some points within the clusters is much larger than the distance between some points from different clusters, yet the regions containing the clusters clearly have a higher point density than the region between them, and they can easily be separated.



Density-Based Clustering. Figure 1. Illustration of a density-based clustering, showing three distinguishable groups

Density-based clustering is one of the prominent paradigms for clustering large data sets in the data mining community. It has been extensively studied and successfully used in many applications.

Structure of Learning System

Assuming that the data set $O \subset R^d$ is a sample from some unknown probability density $p(x)$, there are different ways of determining high density areas of the density $p(x)$. Commonly, the notion of a high density area is (implicitly or explicitly) based on a local density estimate at each point (typically, some kernel or nearest neighbor density estimate), and a notion of connection between objects (typically, points are connected if they are within a certain distance ε from each other); clusters are essentially constructed as maximal sets of objects that are directly or transitively connected to objects whose density exceeds some threshold λ. The set $\{x | p(x) > \lambda\}$ of all high density objects is called the *density level set* of p at λ. Objects that are not part of such clusters are called *noise* or *outliers*.

Different proposed density-based methods distinguish themselves mainly by how the density $p(x)$ is estimated, how the notion of connectivity is defined, and how the algorithm for finding connected components of the induced graph is implemented and supported by suitable data structures to achieve scalability for large data sets. Some methods include in a cluster only objects whose density exceeds the threshold λ, while others also include objects with lower density if they are connected to an object with density above the threshold λ.

Density-based clustering was probably introduced for the first time by Wishart (1969). His algorithm for *one level mode analysis* consists of six steps: "(1) Select a distance threshold r, and a frequency (or density) threshold k, (2) Compute the triangular similarity matrix of all inter-point distances, (3) Evaluate the frequency k_i of each data point, defined as the number of points which lie within a distance r of point i (...), (4) Remove the "noise" or non-dense points, those for which $k_i < k$, (5) Cluster the remaining dense points ($k_i > k$) by single linkage, forming the mode nuclei, (6) Reallocate each non-dense point to a suitable cluster according to some criterion (...) (Wishart, 1969).

Hartigan (1975) suggested a more general definition of a density-based cluster, a *density contour cluster* at level λ, as a maximally connected set of points x for which $p(x) > \lambda$, given a density $p(x)$ at each point x, a density threshold λ, and links specified for some pairs of objects. For instance, given a particular distance function, points can be defined as linked if the distance between them is no greater than some threshold r, or, if only direct links are available, one can define a "distance" for pairs of objects x and y in the following way:

$$d(x,y) = \begin{cases} -\min[p(x), p(y)], & x \text{ and } y \text{ are linked,} \\ 0 & \text{otherwise.} \end{cases}$$

To compute the density-contour clusters, Hartigan, like Wishart, suggest a version of single linkage clustering, which will construct the maximal connected sets of objects of density greater than the given threshold λ.

The DBSCAN algorithm (Ester et al., 1996) introduced density-based clustering independently to the Computer Science Community, also proposing the use of spatial index structures to achieve a scalable clustering algorithm. Assuming a distance threshold r and a density threshold k, DBSCAN, like Wishart's method, estimates the density for each point x_i as the number k_i of points that lie inside a radius r around x. *Core points* are defined as data points for which $k_i > k$. Points are considered directly connected if the distance between them is no greater than r. Density-based clusters are defined as maximally connected components of the set of points that lie within distance r from some core object (i.e., a cluster may contain points x_i with $k_i < k$, called *border objects*, if they are within distance r of a core object of that cluster). Objects not part of a cluster are considered *noise*. The algorithm DBSCAN constructs clusters iteratively, starting a new cluster C with a non-assigned core object x, and assigning all points to C that are directly or transitively connected to x. To determine directly and transitively connected points for a given point, a spatial index structure is used to perform range queries with radius r for each object that is newly added to a current cluster, resulting in an efficient runtime complexity for moderately dimensional data of $O(N \log N)$, where N is the total number of points in the data set, and a worst case runtime of $O(N^2)$, e.g., for high-dimensional data when the performance of spatial index structures deteriorates.

DENCLUE (Hinneburg and Keim, 1998) proposed a notion of density-based clusters using a kernel density estimation. Each data point x is associated with ("attracted by") a local maximum ("density attractor") of the overall density function that lies in the direction of maximum increase in density from x. Density-based clusters are defined as connected components of density attractors with their associated points, whose density estimate is above a given threshold λ. In this formulation, DBSCAN and Wishart's method can be seen as special cases of DENCLUE, using a uniform spherical kernel and, for Wishart's method, not including attracted points whose density is below λ. DENCLUE essentially uses a Gaussian kernel for the implementation, which is based on a clever data structure to speed up local density estimation. The data space is partitioned into d-dimensional cells. Non-empty cells are mapped to one-dimensional keys, which are stored, together with some sufficient statistics about the cell (number of points, pointers to points, and linear sum of the points belonging to the cell), in a search tree for

efficient retrieval of neighboring cells, and local density estimation (Hinneburg and Keim (1998) reports that in an experimental comparison on 11-dimensional data sets of different sizes, DENCLUE runs up to 45 times faster than DBSCAN).

A large number of related methods and extensions have been proposed, particularly in computer science and application-oriented domains, some motivated by algorithmic considerations that could improve efficiency of the computation of density-based clusters, others motivated by special applications, proposing essentially density based clustering algorithms using specific density measures and notions of connectivity. An algorithmic framework, called GDBSCAN, which generalizes the topological properties of density-based clusters, can be found in Sander et al. (1998). GDBSCAN generalizes the notion of a density-based clustering to that of a *density-connected decomposition*, assuming only a reflexive and symmetric *neighborhood* relation for pairs of objects (direct links between some objects), and an arbitrary predicate, called "*MinWeight*," that evaluates to *true* for some neighborhood sets of objects and *false* on others, so that a core object can be defined as an object whose neighborhood satisfies the *MinWeight* predicate. Then, a density-connected decomposition consists of the maximally connected components of the set of objects that are in the neighborhood of some core object, and they can be computed with the same algorithmic scheme as density-based clusters by DBSCAN.

One of the principal problems of finding the density-based clusters of a density level set for a single level λ is how to determine a suitable level λ. The result of a density-based clustering method depends critically on the choice of λ, which may be difficult to determine even in situations when a meaningful level exists, depending on how well the clusters are separated in the given sample. In other situations, it may not even be possible to characterize the cluster structure appropriately using a single density threshold, when modes exist in different regions of the data space that have very different local densities, or clusters are nested within clusters. The problem of selecting suitable density threshold parameters has been already observed by Wishart (1969) who also proposed a hierarchical algorithm to represent the clusters at different density levels. Hartigan (1975) also observed that density-based clusters at different density levels have a hierarchical structure, a *density contour tree*, based on the fact that two clusters (i.e., connected components) of different density levels are either disjoint, or the cluster of higher density is completely contained in the cluster of lower density. Recent proposals for hierarchical clustering methods based on a density estimate and a notion of linkage are, e.g., Ankerst et al. (1999) and Stuetzle (2003). These hierarchical methods are closely related, and are essentially processing and rendering a Minimum Spanning Tree of the data (with pairwise distances or reachability distances as defined in Stuetzle (2003) as edge weights), and are thus also closely related to single linkage clustering. Hierarchical methods do not, in a strict sense, compute a partition of the data, but compute a representation of the overall hierarchical density structure of the data from which particular density-based clusters at different density levels or a global density threshold (a "cut level") could be determined.

Cross References

►Clustering
►Density Estimation

Recommended Reading

Ankerst, M., Breunig, M. M., Kriegel, H.-P., Sander, J. (1999). OPTICS: Ordering Points to Identify the Clustering Structure. In A. Delis, C. Faloutsos, S. Ghandeharizadeh (Eds.), *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. Philadelphia: ACM.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In E. Simoudis, J. Han, & U.M. Fayyad (Eds.), *Proceedings of the second International Conference on Knowledge Discovery and Data Mining*. Portland: AAAI Press.

Hartigan, J. A. (1975). *Clustering Algorithms*. New York: Wiley.

Hinneburg, A., Keim, D. A. (1998). En Efficient Approach to Clustering in Large Multimedia Databases with Noise. In R. Agrawal, & P. Stolorz (Eds.), *Proceedings of the fourth International Conference on Knowledge Discovery and Data Mining*. New York City: AAAI Press.

Sander, J., Ester, M., Kriegel, H.-P., Xu, X. (1998). Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery, 2*(2), 169–194.

Stuetzle, W. (2003). Estimating the Cluster Tree of a Density by Analyzing the Minimal Spanning Tree of a Sample. *Journal of Classification, 20*(1), 025–047.

Wishart, D. (1969). Mode analysis: A generalization of nearest neighbor which reduces chaining effects. In A. J. Cole (Ed.), *Proceedings of the Colloquium in Numerical Taxonomy*. Scotland: St. Andrews.

Dependency Directed Backtracking

►Intelligent Backtracking

Detail

In ►Minimum Message Length, *detail* is the code or language shared between sender and receiver that is used to describe the data conditional on the asserted model.

Deterministic Decision Rule

►Decision Rule

Digraphs

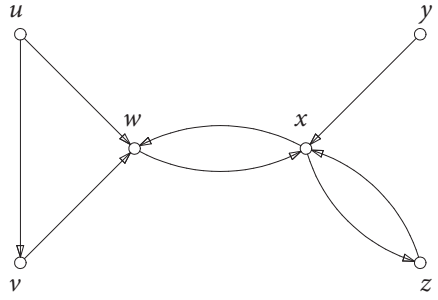
Synonyms

Directed graphs

Definition

A *digraph* D consists of a (finite) set of *vertices* $V(D)$ and a set $A(D)$ of ordered pairs, called *arcs*, of distinct vertices. An arc (u, v) has *tail* u and *head* v, and it is said to leave u and enter v.

Figure 1 shows a digraph D with vertex set $V(D) = \{u, v, w, x, y, z\}$ and arc set $A(D) = \{(u, v), (u, w), (v, w), (w, x), (x, w), (x, z), (y, x), (z, x)\}$. Digraphs can be viewed as generalizations of ►graphs.



Digraphs. Figure 1. A digraph

Dimensionality Reduction

Michail Vlachos
IBM Zürich Research Laboratory
Rüschlikon
Switzerland

Synonyms

Feature extraction

Definition

Every data object in a computer is represented and stored as a set of features, for example, color, price, dimensions, and so on. Instead of the term *features* one can use interchangeably the term *dimensions*, because an object with n features can also be represented as a multidimensional point in an n-dimensional space. Therefore, dimensionality reduction refers to the process of mapping an n-dimensional point, into a lower k-dimensional space. This operation reduces the size for representing and storing an object or a dataset generally; hence, dimensionality reduction can be seen as a method for data compression. Additionally, this process promotes data visualization, particularly when objects are mapped onto two or three dimensions. Finally, in the context of classification, dimensionality reduction can be a useful tool for the following: (a) making tractable classification schemes that are super-linear with respect to dimensionality, (b) reducing the variance of classifiers that are plagued by large variance in higher dimensionalities, and (c) removing the noise that may be present, thus boosting classification accuracy.

Motivation and Background

There are many techniques for dimensionality reduction. The objective of dimensionality reduction techniques is to appropriately select the k dimensions (and also the number k) that would retain the important characteristics of the original object. For example, when performing dimensionality reduction on an image, using a wavelet technique, then the desirable outcome is for the difference between the original and final images to be almost imperceptible.

When performing dimensionality reduction not on a single object, but on a dataset, an additional requirement is for the method to preserve the relationship

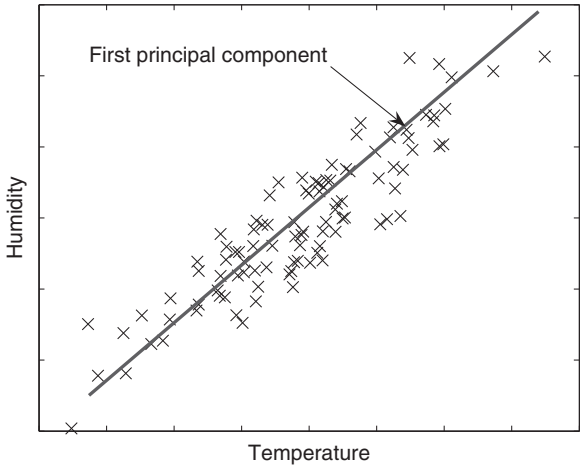
between the objects in the original space. This is particularly important for reasons of classification and visualization in the new space.

There exist two important categories of dimensionality reduction techniques:

- *Feature selection* techniques, where only the most important or descriptive features/dimensions are retained and the remaining are discarded. More details on such techniques can be found under the entry ►Feature Selection.
- *Feature projection* methodologies, which project the existing features onto different dimensions or axes. The aim here is again, to find these new data axes that retain the dataset structure and its variance as closely as possible.

Feature projection techniques typically exploit the correlations between the various data dimensions, with the goal of creating dimensions/axes that are uncorrelated and sufficiently describe the data.

One of the most popular dimensionality reduction techniques is *Principal Components Analysis* or PCA. It attempts to discover those axes (or components) onto which the data can be projected, while maintaining the original correlation between the dimensions. Consider, for example, a dataset that contains records of environmental measurements over a period of time, such as humidity and temperature. The two attributes can be highly correlated, as shown in Fig. 1. By deploying PCA this trend will be discovered and the original two-dimensional points can be reduced to one-dimensional, by projecting the original points on the first *principal component*. In that way the derived dataset can be stored in less space.



Dimensionality Reduction. Figure 1. Principal components analysis (PCA)

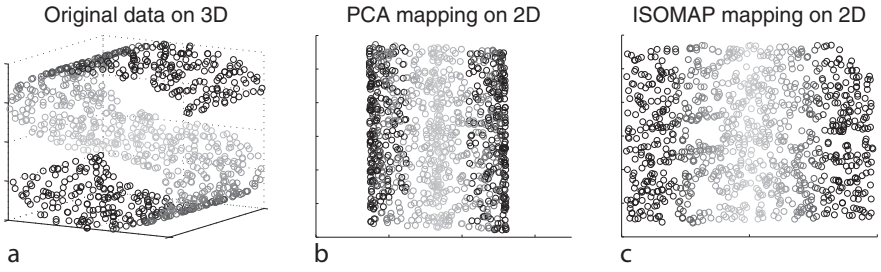
PCA uses the Euclidean distance as the measure of dissimilarity among the objects. The first principal component (or axis) indicates the direction of maximum variance in the original dimensions. The second component shows the direction of next highest variance (and is uncorrelated to the first component), etc.

Other dimensionality reduction techniques optimize or preserve different criteria than PCA. Manifold inspired methods like ISOMAP (Tenenbaum et al., 2000) preserve the geodesic distances between objects. The notion here is to approximate the distance between objects "through" the remaining ones. The result of such dimensionality reduction techniques, is that when the data lie on a manifold, the projected dimensions effectively 'unfold' the underlying high-dimensional manifold. An example of this mapping is portrayed in Fig. 2, where it is also compared with the respective PCA mapping.

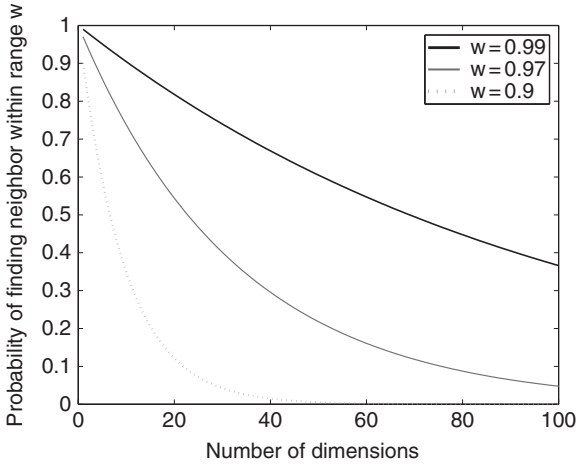
Other recent dimensionality reduction techniques include locally linear embeddings (LLE) (Roweis and Saul, 2000) and Laplacian Eigenmaps (Belkin and Niyogi, 2002). We also refer the interested practitioners to (van der Maaten et al., 2009), for a detailed comparison of various techniques and also for Matlab implementations on a variety of dimensionality reduction algorithms.

In general, dimensionality reduction is a commonly practiced and useful operation in database and machine learning systems because it generally offers the following desirable properties:

- Data compression: the dataset objects are represented in fewer dimensions, hence saving important disk storage space and offering faster loading of the compressed data from the disk.
- Better data visualization: the relationships between the original high-dimensional objects can be visualized in two- or three-dimensional projections.
- Improved classification accuracy: this can be attributed to both variance reduction and noise removal from the original high-dimensional dataset.



Dimensionality Reduction. Figure 2. Nonlinear dimensionality reduction techniques produce a better low-dimensional data mapping, when the original data lie on a high-dimensional manifold



Dimensionality Reduction. Figure 3. Probability $P_w(d)$ against dimensionality d. The data becomes sparse in higher dimensions

- More efficient data retrieval: dimensionality reduction techniques can also assist in making faster and more efficient the retrieval of the original uncompressed data, by offering very fast prefiltering with the help of the compressed data representation.
- Boosting index performance: more effective use of indexing structures can be achieved by utilizing the compressed data, since indexing techniques only work efficiently with lower-dimensional data (e.g., from 1 to 30 dimensions, depending on the type of the index).

The fact that indexing structures do not perform efficiently for higher-dimensional data is also known as ►"curse of dimensionality." Suppose that we are interested in performing search operations on a set of high-dimensional data. For simplicity let us assume that the data lie in a unit hypercube $C = [0,1]^d$, where d is the data dimensionality. Given a query point, the probability P_w that a match (neighbor) exists within radius w in the data space of dimensionality d is given by $P_w(d) = w^d$.

Figure 3 illustrates this probability for various values of w. Evidently, at higher dimensionalities the data becomes very sparse and even at large radii, only a small portion of the entire space is covered. This fact is coined under the term 'curse of dimensionality,' which in simple terms translates into the following fact: for large dimensionalities existing indexing structures outperform sequential scan only when the dataset size (number of objects) grows exponentially with respect to dimensionality.

Dimensionality Reduction for Time-Series Data

In this section we provide more detailed examples on dimensionality reduction techniques for ►time-series data. We chose time-series in order to convey more visually the effect of dimensionality reduction particularly for high-dimensional data such as time-series.

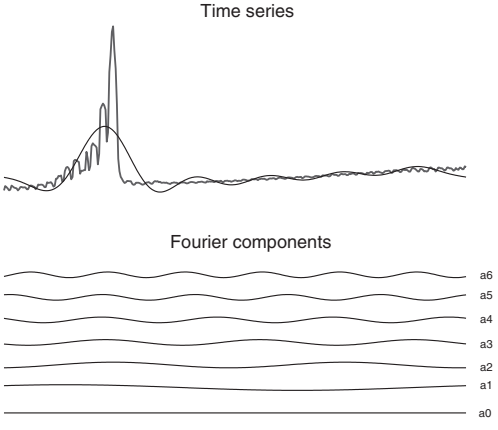
Later, we also show how dimensionality reduction on large datasets can help speed up the search operations over the original uncompressed data.

Dimensionality reduction for one- and two-dimensional signals is commonly accomplished using the Fourier decomposition. Fourier decomposition was first presented in the beginning of the nineteenth century by Jean Baptiste Fourier (1768–1830), in his seminal work *On the Propagation of Heat in Solid Bodies*. Fourier reached the conclusion that every function could be expressed as a sum of trigonometrical series (i.e., sines and cosines). This original work was initially faced

with doubt (even by famous mathematicians such as Lagrange and Laplace), because of its unexpected result and because the solution was considered impractical due of the complex integration functions.

However, in the twentieth century no one can deny the importance of Fourier's findings. With the introduction of fast ways to compute the Fourier decomposition in the 1960s (Fast Fourier Transform or FFT), the barrier of the high computational complexity has been lifted. What the Fourier transform attempts to achieve is, represent the original signal as a linear combination of sinusoids. Therefore, each Fourier coefficient is a complex number that essentially encodes the amplitude and phase of each of these sinusoids, after the original signal is projected on them.

For most signals, utilizing just few of the coefficients we can reconstruct with high accuracy the original sequence. This is where the great power of the Fourier transformation lies; by neglecting the majority of the coefficients, we can essentially compress the signal or describe it with fewer numbers. For stock market data or other time-series that follow the pattern of a random walk, the first few coefficients, which capture the low frequencies of the signal, are adequate to describe accurately the signal (or capture most of its energy). Figure 4 depicts a signal of 1,024 points and its reconstruction using seven Fourier coefficients (i.e., using $7 \times 2 = 14$ numbers).



Dimensionality Reduction. Figure 4. Decomposition of a signal into the first seven Fourier coefficients. We can see that using only of few of the Fourier coefficients we can achieve a good reconstruction of the original signal

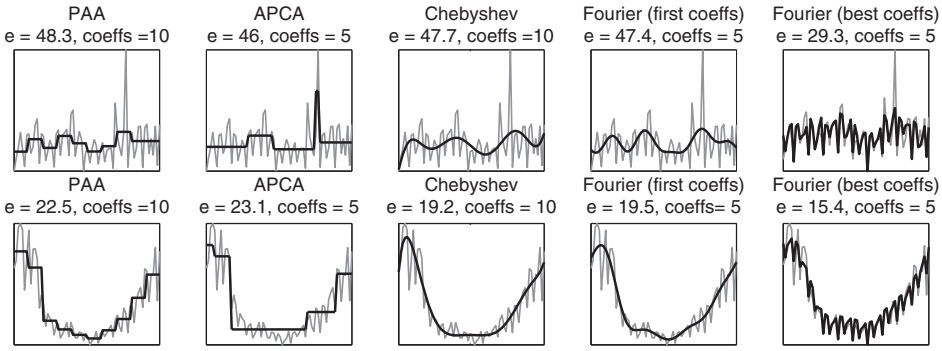
Other popular dimensionality reduction techniques for time-series data are the various wavelet transforms, piecewise linear approximations, piecewise aggregate approximation (PAA), which can be regarded as a projection in time of the wavelet coefficients, adaptive piecewise constant approximation (APCA (Keogh et al., 2001)), that utilizes the highest energy wavelet coefficients, Chebyshev Polynomial Approximation and Symbolic Approximation of time-series (such as the SAX representation (Lin et al., 2003)).

No dimensionality reduction technique is universally better than all the rest. According to the dataset characteristics, one method may provide better approximation of a dataset compared to other techniques. Therefore, the key is to carefully pick the representation that better suits the specific application or the task at hand. In Fig. 5 we demonstrate various dimensionality reduction techniques and the quality of the time-series approximation. For all of the methods, the same storage space is allocated for the compressed sequences. The time-series reconstruction is shown in darker color, and the approximation error to the original sequence is also reported. In general, we can notice that dimensionality reduction techniques based on the selection of the highest energy coefficients can consistently provide a high quality sequence approximation.

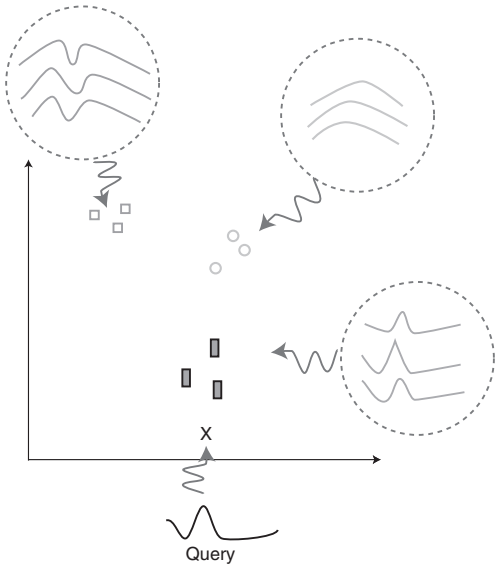
Dimensionality Reduction and Lower-Bounding

Dimensionality reduction can be a useful tool for speeding up search operations. Figure 6 elucidates dimensionality reduction for high-dimensional time-series data. After dimensionality reduction, each object is represented using fewer dimensions (attributes), so it is represented in a lower-dimensional space. Suppose that a user poses another high-dimensional object as a query and wishes to find all the objects closest to this query.

In order to avoid the search on the original high-dimensional space, the query is also transformed into a point in the low-dimensional space and its closest matches can be discovered in the vicinity of the projected query point. However, when searching using the compressed objects, one needs to provide an estimate of the distance between the original objects. Typically, it is preferable that the distance in the new space underestimates (or lower bounds) the distance in the original high-dimensional space. The reason for this is explained as follows.



Dimensionality Reduction. Figure 5. Comparison of various dimensionality reduction techniques for time-series data. The *darker series* indicates the approximation using the indicated number of coefficients. Each figure also reports the error *e* introduced by the dimensionality reduction technique. Lower errors indicate better low-dimensional approximation of the original object



Dimensionality Reduction. Figure 6. Search and dimensionality reduction. Every object (time-series in this case) is tranformed into a lower-dimensional point. User queries are also projected into the new space. Similarity search consists in finding the closest points to the query projection

Suppose that we are seeking for the 1-NN (►Nearest-Neighbor) of a query Q in a database $\mathcal{D}$. By examining all the objects (linear scan) one can guarantee that the best match will be found. Can one provide the same guarantee (i.e., that the same best match will be returned) when examining the compressed objects (after dimensionality reduction)?

The answer is positive, as long as the distance on the compressed data *underestimates* or *lower bounds* the distance on the raw data. In other words, the dimensionality reduction (dR) that is performed on the raw data must have the following property:

$$\text{Having} \quad A \subset \mathcal{D} \xrightarrow{dR} a \quad \text{and} \quad Q \xrightarrow{dR} q$$

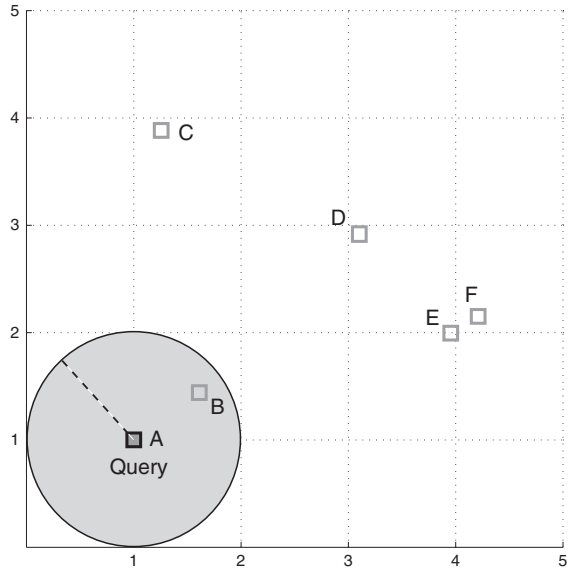
$$\text{then}$$

$$\Delta(q, a) \leq \Delta(Q, A)$$

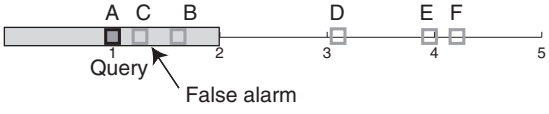
Since the computed distance Δ between any two compressed objects is underestimated, *false alarms* may arise. Suppose, for example, that our database consists of six two-dimensional point (Fig. 7). If the user query is "Find everything that lies within a radius of 1 around A," then B is the only result.

Let us assume for a minute that the dimensionality reduction that is performed on the data is simply a projection on the x-axis (Fig. 8). In this new space, seeking for points within a range of 1 from A, would also retrieve point C, which is called a *false alarm*. This does not constitute a problem, because in a post-processing phase, the calculation of the exact distance will eliminate any false alarms. Suppose now, that another dimensionality reduction results in the projection of Fig. 9. Here, we have a case of a *false dismissal*, since object B lies outside the range of search.

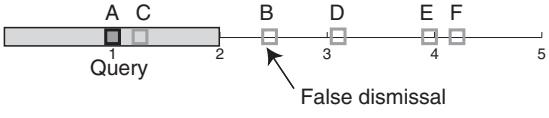
This generic framework for similarity search using dimensionality reduction and lower-bounding distance



Dimensionality Reduction. Figure 7. Range search in the original space, returns only object *B*



Dimensionality Reduction. Figure 8. Because of the dimensionality reduction, false alarms may arise



Dimensionality Reduction. Figure 9. False dismissals may happen when the lower bounding lemma is not obeyed

functions was proposed in (Agrawal et al., 1993) and is called GEMINI (**GE**neric **M**ultimedia **IN**dex**I**ng). One can show that orthonormal dimensionality reduction techniques (PCA, Fourier, Wavelets) satisfy the lower bounding lemma when the distance used is the Euclidean distance.

In conclusion, for search operations, by using dimensionality reduction one can examine first the compressed objects and eliminate many of the uncompressed objects from examination using a lower-bounding approximation of the distance function. This initial search will return a superset of the correct answers (no false dismissals). False alarms can be filtered out by computing the original distance between the remaining uncompressed objects and the query. Therefore, a significant speedup is achieved by examining only a small subset of the original raw data.

Cross References

►Curse of Dimensionality
►Feature Selection

Recommended Reading

Agrawal, R., Faloutsos, C., & Swami, A. (1993). Efficient similarity search in sequence databases. *Proceedings of Foundations of Data Organization and Algorithms*, Chicago, Illinois. (pp. 69–84).

Belkin, M., & Niyogi, P. (2002). Laplacian Eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems*. (Vol. 14, pp. 585–591). Canada: Vancouver.

Jolliffee, I. T. (2002). *Principal component analysis* (2nd ed.). New York: Springer.

Keogh, E., Chakrabarti, K., Pazzani, M., & Mehrotra, S. (2001). Locally adaptive dimensionality reduction for indexing large time series databases. *Proceedings of ACM SIGMOD*, (pp. 151–162). Santa Barbara, CA.

Lin, J., Keogh, E., Lonardi, S., & Chiu, B. (2003). A symbolic representation of time series, with implications for streaming algorithms. *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. San Diego, CA.

Tenenbaum, J. B., de Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science, 290*(5500), 2319–2323.

Roweis, S., & Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science, 290*(5500), 2323–2326.

van der Maaten, L. J. P., Postma, E. O., and van den Herik, H. J. (2009). *Dimensionality reduction: a comparative review* (TiCC-TR 2009-005), Tilburg University Technical Report.

Dimensionality Reduction on Text via Feature Selection

►Feature Selection in Text Mining

Directed Graphs

►Digraphs

Dirichlet Process

Yee Whye Teh
University College London,
London, UK

Definition

The Dirichlet process (DP) is a stochastic process used in ►Bayesian nonparametric models of data, particularly in Dirichlet process mixture models (also known as infinite mixture models). It is a distribution over distributions, that is, each draw from a Dirichlet process is itself a distribution. It is called a Dirichlet process because it has Dirichlet distributed finite dimensional marginal distributions, just as the ►Gaussian process, another popular stochastic process used for Bayesian nonparametric regression, has Gaussian distributed finite dimensional marginal distributions. Distributions drawn from a Dirichlet process are discrete, but cannot be described using a finite number of parameters, thus the classification as a nonparametric model.

Motivation and Background

Probabilistic models are used throughout machine learning to model distributions over observed data. Traditional parametric models using a fixed and finite number of parameters can suffer from over- or underfitting of data when there is a misfit between the complexity of the model (often expressed in terms of the number of parameters) and the amount of data available. As a result, model selection, or the choice of a model with the right complexity, is often an important issue in parametric modeling. Unfortunately, model selection is an operation that is fraught with difficulties, whether we use ►cross validation or marginal probabilities as the basis for selection. The Bayesian nonparametric approach is an alternative to parametric modeling and selection. By using a model with an unbounded complexity, underfitting is mitigated, while the Bayesian approach of computing or approximating the full posterior over parameters mitigates overfitting. For a general overview of Bayesian nonparametrics, see ►Bayesian Nonparametrics.

Nonparametric models are also motivated philosophically by Bayesian modeling. Typically we assume that we have an underlying and unknown distribution which we wish to infer given some observed data. Say we observe $x_1, \ldots, x_n$, with $x_i \sim F$ independent and identical draws from the unknown distribution F. A Bayesian would approach this problem by placing a prior over F then computing the posterior over F given data. Traditionally, this prior over distributions is given by a parametric family. But constraining distributions to lie within parametric families limits the scope and type of inferences that can be made. The nonparametric approach instead uses a prior over distributions with wide support, typically the support being the space of all distributions. Given such a large space over which we make our inferences, it is important that posterior computations are tractable.

The Dirichlet process is currently one of the most popular Bayesian nonparametric models. It was first formalized in Ferguson (1973) for general Bayesian statistical modeling, as a prior over distributions with wide support yet tractable posteriors. (Note however that related models in population genetics date back to Ewens (1972)). Unfortunately the Dirichlet process is limited by the fact that draws from it are discrete distributions, and generalizations to more general priors did not have tractable posterior inference until the development of MCMC (►Markov chain Monte Carlo) techniques (Escobar & West, 1995; Neal, 2000). Since then there has been significant developments in terms of inference algorithms, extensions, theory and applications. In the machine learning, community work on Dirichlet processes date back to Neal (1992) and Rasmussen (2000).

Theory

The Dirichlet process (DP) is a stochastic process whose sample paths are probability measures with probability one. Stochastic processes are distributions over function spaces, with sample paths being random functions drawn from the distribution. In the case of the DP, it is a distribution over probability measures, which are functions with certain special properties, which allow them to be interpreted as distributions over some probability space Θ. Thus draws from a DP can be interpreted as

random distributions. For a distribution over probability measures to be a DP, its marginal distributions have to take on a specific form which we shall give below. We assume that the user is familiar with a modicum of measure theory and Dirichlet distributions.

Before we proceed to the formal definition, we will first give an intuitive explanation of the DP as an infinite dimensional generalization of Dirichlet distributions. Consider a Bayesian mixture model consisting of K components:

$$\begin{aligned} \pi|\alpha &\sim Dir\left(\tfrac{\alpha}{K},\ldots,\tfrac{\alpha}{K}\right) & \theta_k^*|H &\sim H \\ z_i|\pi &\sim Mult\left(\pi\right) & x_i|z_i,\{\theta_k^*\} &\sim F\left(\theta_{z_i}^*\right) \end{aligned} \quad (1)$$

where π is the mixing proportion, α is the pseudo-count hyperparameter of the Dirichlet prior, H is the prior distribution over component parameters θ_k^*, and $F(\theta)$ is the component distribution parametrized by θ. It can be shown that for large K, because of the particular way we parametrized the Dirichlet prior over π, the number of components typically used to model n data items becomes independent of K and is approximately $O(\alpha \log n)$. This implies that the mixture model stays well defined as $K \to \infty$, leading to what is known as an infinite mixture model (Neal, 1992; Rasmussen, 2000). This model was first proposed as a way to sidestep the difficult problem of determining the number of components in a mixture, and as a nonparametric alternative to finite mixtures whose size can grow naturally with the number of data items. The more modern definition of this model uses a DP and with the resulting model called a DP mixture model. The DP itself appears as the $K \to \infty$ limit of the random discrete probability measure $\sum_{k=1}^{K} \pi_k \delta_{\theta_k^*}$, where δ_θ is a point mass centered at θ. We will return to the DP mixture toward the end of this entry.

Dirichlet Process

For a random distribution G to be distributed according to a DP, its marginal distributions have to be Dirichlet distributed (Ferguson, 1973). Specifically, let H be a distribution over Θ and α be a positive real number. Then for any finite measurable partition $A_1,\ldots,A_r$ of Θ the vector $(G(A_1),\ldots,G(A_r))$ is random since G is random. We say G is Dirichlet process distributed with base distribution H and concentration parameter α, written $G \sim DP(\alpha, H)$, if

$$(G(A_1),\ldots,G(A_r)) \sim Dir(\alpha H(A_1),\ldots,\alpha H(A_r)) \quad (2)$$

for every finite measurable partition $A_1,\ldots,A_r$ of Θ.

The parameters H and α play intuitive roles in the definition of the DP. The base distribution is basically the mean of the DP: for any measurable set $A \subset \Theta$, we have $E[G(A)] = H(A)$. On the other hand, the concentration parameter can be understood as an inverse variance: $V[G(A)] = H(A)(1-H(A))/(\alpha+1)$. The larger α is, the smaller the variance, and the DP will concentrate more of its mass around the mean. The concentration parameter is also called the strength parameter, referring to the strength of the prior when using the DP as a nonparametric prior over distributions in a Bayesian nonparametric model, and the mass parameter, as this prior strength can be measured in units of sample size (or mass) of observations. Also, notice that α and H only appear as their product in the definition (3) of the DP. Some authors thus treat $\widetilde{H} = \alpha H$, as the single (positive measure) parameter of the DP, writing $DP(\widetilde{H})$ instead of $DP(\alpha, H)$. This parametrization can be notationally convenient, but loses the distinct roles α and H play in describing the DP.

Since α describes the concentration of mass around the mean of the DP, as $\alpha \to \infty$, we will have $G(A) \to H(A)$ for any measurable A, that is $G \to H$ weakly or pointwise. However this not equivalent to saying that $G \to H$. As we shall see later, draws from a DP will be discrete distributions with probability one, even if H is smooth. Thus G and H need not even be absolutely continuous with respect to each other. This has not stopped some authors from using the DP as a nonparametric relaxation of a parametric model given by H. However, if smoothness is a concern, it is possible to extend the DP by convolving G with kernels so that the resulting random distribution has a density.

A related issue to the above is the coverage of the DP within the class of all distributions over Θ. We already noted that samples from the DP are discrete, thus the set of distributions with positive probability under the DP is small. However it turns out that this set is also large in a different sense: if the topological support of H (the smallest closed set S in Θ with $H(S) = 1$) is all of Θ, then any distribution over Θ can be approximated

arbitrarily accurately in the weak or pointwise sense by a sequence of draws from $DP(\alpha, H)$. This property has consequence in the consistency of DPs discussed later.

For all but the simplest probability spaces, the number of measurable partitions in the definition (3) of the DP can be uncountably large. The natural question to ask here is whether objects satisfying such a large number of conditions as (3) can exist. There are a number of approaches to establish existence. Ferguson (1973) noted that the conditions (3) are consistent with each other, and made use of Kolmogorov's consistency theorem to show that a distribution over functions from the measurable subsets of Θ to $[0,1]$ exists satisfying (3) for all finite measurable partitions of Θ. However it turns out that this construction does not necessarily guarantee a distribution over probability measures. Ferguson (1973) also provided a construction of the DP by normalizing a gamma process. In a later section we will see that the predictive distributions of the DP are related to the Blackwell–MacQueen urn scheme. Blackwell and MacQueen (1973) made use of this, along with de Finetti's theorem on exchangeable sequences, to prove existence of the DP. All the above methods made use of powerful and general mathematical machinery to establish existence, and often require regularity assumptions on H and Θ to apply these machinery. In a later section, we describe a stick-breaking construction of the DP due to Sethuraman (1994), which is a direct and elegant construction of the DP, which need not impose such regularity assumptions.

Posterior Distribution

Let $G \sim DP(\alpha, H)$. Since G is a (random) distribution, we can in turn draw samples from G itself. Let $\theta_1, \ldots, \theta_n$ be a sequence of independent draws from G. Note that the θ_i's take values in Θ since G is a distribution over Θ. We are interested in the posterior distribution of G given observed values of $\theta_1, \ldots, \theta_n$. Let $A_1, \ldots, A_r$ be a finite measurable partition of Θ, and let $n_k = \#\{i : \theta_i \in A_k\}$ be the number of observed values in A_k. By (3) and the conjugacy between the Dirichlet and the multinomial distributions, we have

$$\begin{aligned}&(G(A_1), \ldots, G(A_r))|\theta_1, \ldots, \theta_n \\ &\sim \mathrm{Dir}(\alpha H(A_1) + n_1, \ldots, \alpha H(A_r) + n_r) \end{aligned} \tag{3}$$

Since the above is true for all finite measurable partitions, the posterior distribution over G must be a DP as well. A little algebra shows that the posterior DP has updated concentration parameter $\alpha+n$ and base distribution $\frac{\alpha H + \sum_{i=1}^n \delta_{\theta_i}}{\alpha+n}$, where δ_i is a point mass located at θ_i and $n_k = \sum_{i=1}^n \delta_i(A_k)$. In other words, the DP provides a conjugate family of priors over distributions that is closed under posterior updates given observations. Rewriting the posterior DP, we have

$$G|\theta_1, \ldots, \theta_n \sim DP\left(\alpha + n, \frac{\alpha}{\alpha+n} H + \frac{n}{\alpha+n} \frac{\sum_{i=1}^n \delta_{\theta_i}}{n}\right) \tag{4}$$

Notice that the posterior base distribution is a weighted average between the prior base distribution H and the empirical distribution $\frac{\sum_{i=1}^n \delta_{\theta_i}}{n}$. The weight associated with the prior base distribution is proportional to α, while the empirical distribution has weight proportional to the number of observations n. Thus we can interpret α as the strength or mass associated with the prior. In the next section we will see that the posterior base distribution is also the predictive distribution of θ_{n+1} given $\theta_1, \ldots, \theta_n$. Taking $\alpha \to 0$, the prior becomes non-informative in the sense that the predictive distribution is just given by the empirical distribution. On the other hand, as the amount of observations grows large, $n \gg \alpha$, the posterior is simply dominated by the empirical distribution, which is in turn a close approximation of the true underlying distribution. This gives a consistency property of the DP: the posterior DP approaches the true underlying distribution.

Predictive Distribution and the Blackwell–MacQueen Urn Scheme

Consider again drawing $G \sim DP(\alpha, H)$, and drawing an i.i.d. (independently and identically distributed) sequence $\theta_1, \theta_2, \ldots \sim G$. Consider the predictive distribution for θ_{n+1}, conditioned on $\theta_1, \ldots, \theta_n$ and with G marginalized out. Since $\theta_{n+1}|G, \theta_1, \ldots, \theta_n \sim G$, for a measurable $A \subset \Theta$, we have

$$\begin{aligned}P(\theta_{n+1} \in A|\theta_1, \ldots, \theta_n) &= E[G(A)|\theta_1, \ldots, \theta_n] \\ &= \frac{1}{\alpha+n}\left(\alpha H(A) + \sum_{i=1}^n \delta_{\theta_i}(A)\right)\end{aligned} \tag{5}$$

where the last step follows from the posterior base distribution of G given the first n observations. Thus with G marginalized out:

$$\theta_{n+1}|\theta_1, \ldots, \theta_n \sim \frac{1}{\alpha+n}\left(\alpha H + \sum_{i=1}^n \delta_{\theta_i}\right) \tag{6}$$

Therefore the posterior base distribution given $\theta_1, \ldots, \theta_n$ is also the predictive distribution of θ_{n+1}.

The sequence of predictive distributions (6) for $\theta_1, \theta_2, \ldots$ is called the Blackwell–MacQueen urn scheme (Blackwell & MacQueen, 1973). The name stems from a metaphor useful in interpreting (6). Specifically, each value in Θ is a unique color, and draws $\theta \sim G$ are balls with the drawn value being the color of the ball. In addition we have an urn containing previously seen balls. In the beginning there are no balls in the urn, and we pick a color drawn from H, that is, draw $\theta_1 \sim H$, paint a ball with that color, and drop it into the urn. In subsequent steps, say the $n+1$st, we will either, with probability $\frac{\alpha}{\alpha+n}$, pick a new color (draw $\theta_{n+1} \sim H$), paint a ball with that color and drop the ball into the urn, or, with probability $\frac{n}{\alpha+n}$, reach into the urn to pick a random ball out (draw θ_{n+1} from the empirical distribution), paint a new ball with the same color, and drop both balls back into the urn.

The Blackwell–MacQueen urn scheme has been used to show the existence of the DP (Blackwell & MacQueen, 1973). Starting from (6), which are perfectly well defined conditional distributions regardless of the question of the existence of DPs, we can construct a distribution over sequences $\theta_1, \theta_2, \ldots$ by iteratively drawing each θ_i given $\theta_1, \ldots, \theta_{i-1}$. For $n \geq 1$ let

$$P(\theta_1, \ldots, \theta_n) = \prod_{i=1}^{n} P(\theta_i | \theta_1, \ldots, \theta_{i-1}) \tag{7}$$

be the joint distribution over the first n observations, where the conditional distributions are given by (6). It is straightforward to verify that this random sequence is infinitely exchangeable. That is, for every n, the probability of generating $\theta_1, \ldots, \theta_n$ using (6), in that order, is equal to the probability of drawing them in any alternative order. More precisely, given any permutation σ on $1, \ldots, n$, we have

$$P(\theta_1, \ldots, \theta_n) = P(\theta_{\sigma(1)}, \ldots, \theta_{\sigma(n)}) \tag{8}$$

Now de Finetti's theorem states that for any infinitely exchangeable sequence $\theta_1, \theta_2, \ldots$ there is a random distribution G such that the sequence is composed of i.i.d. draws from it:

$$P(\theta_1, \ldots, \theta_n) = \int \prod_{i=1}^{n} G(\theta_i)\, dP(G) \tag{9}$$

In our setting, the prior over the random distribution $P(G)$ is precisely the Dirichlet process $DP(\alpha, H)$, thus establishing existence.

A salient property of the predictive distribution (6) is that it has point masses located at the previous draws $\theta_1, \ldots, \theta_n$. A first observation is that with positive probability draws from G will take on the same value, regardless of smoothness of H. This implies that the distribution G itself has point masses. A further observation is that for a long enough sequence of draws from G, the value of any draw will be repeated by another draw, implying that G is composed only of a weighted sum of point masses, that is, it is a discrete distribution. We will see two sections below that this is indeed the case, and give a simple construction for G called the stick-breaking construction. Before that, we shall investigate the clustering property of the DP.

Clustering, Partitions, and the Chinese Restaurant Process

In addition to the discreteness property of draws from a DP, (6) also implies a ▶clustering property. The discreteness and clustering properties of the DP play crucial roles in the use of DPs for clustering via DP mixture models, described in the application section. For now we assume that H is smooth, so that all repeated values are due to the discreteness property of the DP and not due to H itself. (Similar conclusions can be drawn when H has atoms, there is just more bookkeeping.) Since the values of draws are repeated, let $\theta_1^*, \ldots, \theta_m^*$ be the unique values among $\theta_1, \ldots, \theta_n$, and n_k be the number of repeats of θ_k^*. The predictive distribution can be equivalently written as

$$\theta_{n+1} \Big| \theta_1, \ldots, \theta_n \sim \frac{1}{\alpha + n} \left(\alpha H + \sum_{k=1}^{m} n_k \delta_{\theta_k^*} \right) \tag{10}$$

Notice that value θ_k^* will be repeated by θ_{n+1} with probability proportional to n_k, the number of times it has already been observed. The larger n_k is, the higher the probability that it will grow. This is a rich-gets-richer phenomenon, where large clusters (a set of θ_i's with identical values θ_k^* being considered a cluster) grow larger faster.

We can delve further into the clustering property of the DP by looking at partitions induced by the clustering. The unique values of $\theta_1, \ldots, \theta_n$ induce a partitioning of the set $[n] = \{1, \ldots, n\}$ into clusters such that within each cluster, say cluster k, the θ_i's take on the same value θ_k^*. Given that $\theta_1, \ldots, \theta_n$ are random, this induces a random partition of $[n]$. This random partition in fact encapsulates all the properties of the DP, and is a very well-studied mathematical object in its own right, predating even the DP itself (Aldous, 1985; Ewens, 1972; Pitman, 2002). To see how it encapsulates the DP, we simply invert the generative process. Starting from the distribution over random partitions, we can reconstruct the joint distribution (7) over $\theta_1, \ldots, \theta_n$, by first drawing a random partition on $[n]$, then for each cluster k in the partition draw a $\theta_k^* \sim H$, and finally assign $\theta_i = \theta_k^*$ for each i in cluster k. From the joint distribution (7) we can obtain the DP by appealing to de Finetti's theorem.

The distribution over partitions is called the Chinese restaurant process (CRP) due to a different metaphor. (The name was coined by Lester Dubins and Jim Pitman in the early 1980s (Aldous, 1985)) In this metaphor we have a Chinese restaurant with an infinite number of tables, each of which can seat an infinite number of customers. The first customer enters the restaurant and sits at the first table. The second customer enters and decides either to sit with the first customer, or by herself at a new table. In general, the $n+1$st customer either joins an already occupied table k with probability proportional to the number n_k of customers already sitting there, or sits at a new table with probability proportional to α. Identifying customers with integers $1, 2, \ldots$ and tables as clusters, after n customers have sat down the tables define a partition of $[n]$ with the distribution over partitions being the same as the one above. The fact that most Chinese restaurants have round tables is an important aspect of the CRP. This is because it does not just define a distribution over partitions of $[n]$, it also defines a distribution over permutations of $[n]$, with each table corresponding to a cycle of the permutation. We do not need to explore this aspect further and refer the interested reader to Aldous (1985) and Pitman (2002).

This distribution over partitions first appeared in population genetics, where it was found to be a robust distribution over alleles (clusters) among gametes (observations) under simplifying assumptions on the population, and is known under the name of Ewens sampling formula (Ewens, 1972). Before moving on we shall consider just one illuminating aspect, specifically the distribution of the number of clusters among n observations. Notice that for $i \geq 1$, the observation θ_i takes on a new value (thus incrementing m by one) with probability $\frac{\alpha}{\alpha+i-1}$ independently of the number of clusters among previous θ's. Thus the number of cluster m has mean and variance:

$$
\begin{aligned}
E[m|n] &= \sum_{i=1}^{n} \frac{\alpha}{\alpha + i - 1} = \alpha(\psi(\alpha + n) - \psi(\alpha)) \\
&\simeq \alpha \log\left(1 + \frac{n}{\alpha}\right) \qquad \text{for } N, \alpha \gg 0, \qquad (11) \\
V[m|n] &= \alpha(\psi(\alpha + n) - \psi(\alpha)) \\
&\quad + \alpha^2(\psi'(\alpha + n) - \psi'(\alpha)) \\
&\simeq \alpha \log\left(1 + \frac{n}{\alpha}\right) \qquad \text{for } n > \alpha \gg 0, \qquad (12)
\end{aligned}
$$

where $\psi(\cdot)$ is the digamma function. Note that the number of clusters grows only logarithmically in the number of observations. This slow growth of the number of clusters makes sense because of the rich-gets-richer phenomenon: we expect there to be large clusters thus the number of clusters m has to be smaller than the number of observations n. Notice that α controls the number of clusters in a direct manner, with larger α implying a larger number of clusters a priori. This intuition will help in the application of DPs to mixture models.

Stick-Breaking Construction

We have already intuited that draws from a DP are composed of a weighted sum of point masses. Sethuraman (1994) made this precise by providing a constructive definition of the DP as such, called the stick-breaking construction. This construction is also significantly more straightforward and general than previous proofs of the existence of DPs. It is simply given as follows:

$$
\begin{aligned}
\beta_k &\sim Beta(1, \alpha) & \theta_k^* &\sim H \\
\pi_k &= \beta_k \prod_{l=1}^{k-1} (1 - \beta_k) & G &= \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*} \qquad (13)
\end{aligned}
$$

Then $G \sim DP(\alpha, H)$. The construction of π can be understood metaphorically as follows. Starting with a stick of length 1, we break it at β_1, assigning π_1 to be the

length of stick we just broke off. Now recursively break the other portion to obtain π_2, π_3, and so forth. The stick-breaking distribution over π is sometimes written $\pi \sim GEM(\alpha)$, where the letters stand for Griffiths, Engen, and McCloskey (Pitman, 2002). Because of its simplicity, the stick-breaking construction has lead to a variety of extensions as well as novel inference techniques for the Dirichlet process (Ishwaran & James, 2001).

Applications

Because of its simplicity, DPs are used across a wide variety of applications of Bayesian analysis in both statistics and machine learning. The simplest and most prevalent applications include Bayesian model validation, density estimation, and clustering via mixture models. We shall briefly describe the first two classes before detailing DP mixture models.

How does one validate that a model gives a good fit to some observed data? The Bayesian approach would usually involve computing the marginal probability of the observed data under the model, and comparing this marginal probability to that for other models. If the marginal probability of the model of interest is highest we may conclude that we have a good fit. The choice of models to compare against is an issue in this approach, since it is desirable to compare against as large a class of models as possible. The Bayesian nonparametric approach gives an answer to this question: use the space of all possible distributions as our comparison class, with a prior over distributions. The DP is a popular choice for this prior, due to its simplicity, wide coverage of the class of all distributions, and recent advances in computationally efficient inference in DP models. The approach is usually to use the given parametric model as the base distribution of the DP, with the DP serving as a nonparametric relaxation around this parametric model. If the parametric model performs as well or better than the DP relaxed model, we have convincing evidence of the validity of the model.

Another application of DPs is in ▶density estimation (Escobar & West, 1995; Lo, 1984; Neal, 1992; Rasmussen, 2000). Here we are interested in modeling the density from which a given set of observations is drawn. To avoid limiting ourselves to any parametric class, we may again use a nonparametric prior over all densities. Here again DPs are a popular. However note that distributions drawn from a DP are discrete, thus do not have densities. The solution is to smooth out draws from the DP with a kernel. Let $G \sim DP(\alpha, H)$ and let $f(x|\theta)$ be a family of densities (kernels) indexed by θ. We use the following as our nonparametric density of x:

$$p(x) = \int f(x|\theta) G(\theta)\, d\theta \tag{14}$$

Similarly, smoothing out DPs in this way is also useful in the nonparametric relaxation setting above. As we see below, this way of smoothing out DPs is equivalent to DP mixture models, if the data distributions $F(\theta)$ below are smooth with densities given by $f(x|\theta)$.

Dirichlet Process Mixture Models

The most common application of the Dirichlet process is in clustering data using mixture models (Escobar & West, 1995; Lo, 1984; Neal, 1992; Rasmussen, 2000). Here the nonparametric nature of the Dirichlet process translates to mixture models with a countably infinite number of components. We model a set of observations $\{x_1, \ldots, x_n\}$ using a set of latent parameters $\{\theta_1, \ldots, \theta_n\}$. Each θ_i is drawn independently and identically from G, while each x_i has distribution $F(\theta_i)$ parametrized by θ_i:

$$\begin{aligned} x_i|\theta_i &\sim F(\theta_i) \\ \theta_i|G &\sim G \\ G|\alpha, H &\sim DP(\alpha, H) \end{aligned} \tag{15}$$

Because G is discrete, multiple θ_i's can take on the same value simultaneously, and the above model can be seen as a mixture model, where x_i's with the same value of θ_i belong to the same cluster. The mixture perspective can be made more in agreement with the usual representation of mixture models using the stick-breaking construction (13). Let z_i be a cluster assignment variable, which takes on value k with probability π_k. Then (15) can be equivalently expressed as

$$\begin{aligned} \pi|\alpha &\sim GEM(\alpha) & \theta_k^*|H &\sim H \\ z_i|\pi &\sim Mult(\pi) & x_i|z_i, \{\theta_k^*\} &\sim F\left(\theta_{z_i}^*\right) \end{aligned} \tag{16}$$

with $G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*}$ and $\theta_i = \theta_{z_i}^*$. In mixture modeling terminology, π is the mixing proportion, θ_k^* are the

cluster parameters, $F(\theta_k^*)$ is the distribution over data in cluster k, and H the prior over cluster parameters.

The DP mixture model is an *infinite* mixture model – a mixture model with a countably infinite number of clusters. However, because the π_k's decrease exponentially quickly, only a small number of clusters will be used to model the data a priori (in fact, as we saw previously, the expected number of components used a priori is logarithmic in the number of observations). This is different than a finite mixture model, which uses a fixed number of clusters to model the data. In the DP mixture model, the actual number of clusters used to model data is not fixed, and can be automatically inferred from data using the usual Bayesian posterior inference framework (see Neal (2000) for a survey of MCMC inference procedures for DP mixture models). The equivalent operation for finite mixture models would be model averaging or model selection for the appropriate number of components, an approach that is fraught with difficulties. Thus infinite mixture models as exemplified by DP mixture models provide a compelling alternative to the traditional finite mixture model paradigm.

Generalizations and Extensions

The DP is the canonical distribution over probability measures and a wide range of generalizations have been proposed in the literature. First and foremost is the *Pitman–Yor process* (Ishwaran & James, 2001; Pitman & Yor, 1997), which has recently seen successful applications modeling data exhibiting power-law properties (Goldwater, Griffiths, & Johnson, 2006; Teh, 2006). The Pitman–Yor process includes a third parameter $d \in [0,1)$, with $d=0$ reducing to the DP. The various representations of the DP, including the Chinese restaurant process and the stick-breaking construction, have analogues for the Pitman–Yor process. Other generalizations of the DP are obtained by generalizing one of its representations. These include Pólya trees, normalized random measure, Poisson–Kingman models, species sampling models and stick-breaking priors.

The DP has also been used in more complex models involving more than one random probability measure. For example, in nonparametric regression we might have one probability measure for each value of a covariate, and in multitask settings each task might be associated with a probability measure with dependence across tasks implemented using a hierarchical Bayesian model. In the first situation, the class of models is typically called dependent Dirichlet processes (MacEachern, 1999), while in the second the appropriate model is a hierarchical Dirichlet process (Teh, Jordan, Beal, & Blei, 2006).

Future Directions

The Dirichlet process, and Bayesian nonparametrics in general, is an active area of research within both machine learning and statistics. Current research trends span a number of directions. Firstly, there is the issue of efficient inference in DP models. Reference Neal (2000) is an excellent survey of the state-of-the-art in 2000, with all algorithms based on Gibbs sampling or small-step Metropolis–Hastings MCMC sampling. Since then there has been much work, including split-and-merge and large-step auxiliary variable MCMC sampling, sequential Monte Carlo, expectation propagation, and variational methods. Secondly, there has been interest in extending the DP, both in terms of new random distributions, as well as novel classes of nonparametric objects inspired by the DP. Thirdly, theoretical issues of convergence and consistency are being explored to provide frequentist guarantees for Bayesian nonparametric models. Finally, there are applications of such models, to clustering, transfer learning, relational learning, models of cognition, sequence learning, and regression and classification among others. We believe DPs and Bayesian nonparametrics will prove to be rich and fertile grounds for research for years to come.

Cross References

- ▶Bayesian Methods
- ▶Bayesian Nonparametrics
- ▶Clustering
- ▶Density Estimation
- ▶Gaussian Process
- ▶Prior Probabilities

Further Reading

In addition to the references embedded in the text above, we recommend the book (Hjort, Holmes, Müller, & Walker, 2010) on Bayesian nonparametrics.

Recommended Reading

Aldous, D. (1985). Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII-1983* (pp. 1–198). Berlin: Springer.

Antoniak, C. E. (1974). Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *Annals of Statistics, 2*(6), 1152–1174.

Blackwell, D., & MacQueen, J. B. (1973). Ferguson distributions via Pólya urn schemes. *Annals of Statistics, 1*, 353–355.

Escobar, M. D., & West, M. (1995). Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association, 90*, 577–588.

Ewens, W. J. (1972). The sampling theory of selectively neutral alleles. *Theoretical Population Biology, 3*, 87–112.

Ferguson, T. S. (1973). A Bayesian analysis of some nonparametric problems. *Annals of Statistics, 1*(2), 209–230.

Goldwater, S., Griffiths, T. L., & Johnson, M. (2006). Interpolating between types and tokens by estimating power-law generators. In *Advances in neural information processing systems* (Vol. 18).

Hjort, N., Holmes, C., Müller, P., & Walker, S. (Eds.). (2010). *Bayesian nonparametrics. Cambridge series in statistical and probabilistic mathematics* (Vol. 28). Cambridge University Press.

Ishwaran, H., & James, L. F. (2001). Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association, 96*(453), 161–173.

Lo, A. Y. (1984). On a class of Bayesian nonparametric estimates: I. Density estimates. *Annals of Statistics, 12*(1), 351–357.

MacEachern, S. (1999). Dependent nonparametric processes. In *Proceedings of the section on Bayesian statistical science*. American Statistical Association. Alexandria, VA, USA.

Neal, R. M. (1992). Bayesian mixture modeling. In *Proceedings of the workshop on maximum entropy and Bayesian methods of statistical analysis* (Vol. 11, pp. 197–211). Neal (1992): Kluwer Academic Publishers, The Netherlands.

Neal, R. M. (2000). Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics, 9*, 249–265.

Pitman, J. (2002). *Combinatorial stochastic processes* (Tech. Rep. 621). Department of Statistics, University of California at Berkeley. Lecture notes for St. Flour Summer School.

Pitman, J., & Yor, M. (1997). The two-parameter Poisson–Dirichlet distribution derived from a stable subordinator. *Annals of Probability, 25*, 855–900.

Rasmussen, C. E. (2000). The infinite Gaussian mixture model. In *Advances in neural information processing systems* (Vol. 12).

Sethuraman, J. (1994). A constructive definition of Dirichlet priors. *Statistica Sinica, 4*, 639–650.

Teh, Y. W. (2006). A hierarchical Bayesian language model based on Pitman–Yor processes. In *Proceedings of the 21st international conference on computational linguistics and 44th annual meeting of the association for computational linguistics* (pp. 985–992).

Teh, Y. W., Jordan, M. I., Beal, M. J., & Blei, D. M. (2006). Hierarchical Dirichlet processes. *Journal of the American Statistical Association, 101*(476), 1566–1581.

Discrete Attribute

A **discrete attribute** assumes values that can be counted. The attribute cannot assume all values on the number line within its value range. See ►Attribute and ►Measurement Scales.

Discretization

Ying Yang
Australian Taxation Office, Australia

Synonyms

Binning

Definition

Discretization is a process that transforms a ►numeric attribute into a ►categorical attribute. Under discretization, a new categorical attribute X' is formed from and replaces an existing numeric attribute X. Each value x' of X' corresponds to an interval $(a,b]$ of X. Any original numeric value x of X that belongs to $(a,b]$ is replaced by x'. The boundary values of formed intervals are often called "cut points."

Motivation and Background

Many learning systems require categorical data, while many data are numeric. Discretization allows numeric data to be transformed into categorical form suited to processing by such systems. Further, in some cases effective discretization can improve either computational or prediction performance relative to learning from the original numeric data.

Taxonomy

The following taxonomy identifies many key dimensions along which alternative discretization techniques can be distinguished.

►**Supervised** vs. ►**Unsupervised** (Dougherty, Kohavi, & Sahami, 1995). Supervised methods use the class information of the training instances to select discretization cut points. Methods that do not use the class information are unsupervised.

Global vs. **Local** (Dougherty et al., 1995). Global methods discretize with respect to the whole training data space. They perform discretization only once, using a single set of intervals throughout a single classification task. Local methods allow different sets of intervals to be formed for a single attribute, each set being applied in a different classification context. For example, different discretizations of a single attribute might be applied at different nodes of a decision tree (Quinlan, 1993).

Eager vs. **Lazy** (Hsu, Huang, & Wong, 2000). Eager methods perform discretization prior to classification time. Lazy methods perform discretization during the process of classification.

Disjoint vs. **Nondisjoint** (Yang & Webb, 2002). Disjoint methods discretize the value range of a numeric attribute into disjoint intervals. No intervals overlap. Nondisjoint methods discretize the value range into intervals that can overlap.

Parameterized vs. **Unparameterized**. Parameterized discretization requires input from the user, such as the maximum number of discretized intervals. Unparameterized discretization uses information only from data and does not need input from the user, for instance, the entropy minimization discretization (Fayyad & Irani, 1993).

Univariate vs. **Multivariate** (Bay, 2000). Methods that discretize each attribute in isolation are univariate. Methods that take into consideration relationships among attributes during discretization are multivariate.

Split vs. **Merge** (Kerber, 1992) vs. **Single-scan** (Yang & Webb, 2001). Split discretization initially has the whole value range as an interval and then continues splitting it into subintervals until some threshold is met. Merge discretization initially puts each value into an interval and then continues merging adjacent intervals until some threshold is met. Single-scan discretization uses neither split nor merge process. Instead, it scans the ordered values only once, sequentially forming the intervals.

Recommended Reading

Bay, S. D. (2000). Multivariate discretization of continuous variables for set mining. In *Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 315–319).

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Proceedings of the twelfth international conference on machine learning* (pp. 194–202).

Fayyad, U. M. & Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the thirteenth international joint conference on artificial intelligence* (pp. 1022–1027).

Hsu, C. N., Huang, H. J., & Wong, T. T. (2000). Why discretization works for naïve Bayesian classifiers. In *Proceedings of the seventeenth international conference on machine learning* (pp. 309–406).

Kerber, R. (1992). ChiMerge: Discretization for numeric attributes. In *AAAI national conference on artificial intelligence* (pp. 123–128).

Kononenko, I. (1992). Naive Bayesian classifier and continuous Attributes. *Informatica*, *16*(1), 1–8.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco: Morgan Kaufmann Publishers.

Yang, Y., & Webb, G. (2001). Proportional k-interval discretization for naive-Bayes classifiers. In *Proceedings of the twelfth european conference on machine learning* (pp. 564–575).

Yang, Y. & Webb, G. (2002). Non-disjoint discretization for naive-Bayes classifiers. In *Proceedings of the nineteenth international conference on machine learning* (pp. 666–673).

Discriminative Learning

Definition

Discriminative learning refers to any ►classification learning process that classifies by using a ►model or estimate of the probability $P(y|\mathbf{x})$ without reference to an explicit estimate of any of $P(\mathbf{x})$, $P(y, \mathbf{x})$, or $P(\mathbf{x}|y)$, where y is a class and $\mathbf{x}$ is a description of an object to be classified. Discriminative learning contrasts to ►generative learning which classifies by using an estimate of the joint probability $P(y, \mathbf{x})$ or of the prior probability $P(y)$ and the conditional probability $P(\mathbf{x}|y)$.

It is also common to categorize as discriminative any approaches that are directly based on a decision risk function (such as ►Support Vector Machines, ►Artificial Neural Networks, and ►Decision Trees), where the decision risk is minimized without estimation of $P(\mathbf{x})$, $P(y, \mathbf{x})$, or $P(\mathbf{x}|y)$.

Cross References

►Generative and Discriminative Learning

Disjunctive Normal Form

Bernhard Pfahringer
University of Waikato, Hamilton, New Zealand

Disjunctive normal form is an important normal form for propositional logic. A logic formula is in disjunctive normal form if it is a single disjunction of conjunctions of (possibly negated) literals. No more nesting and no other negations are allowed. Examples are:

a
$\neg b$
$a \vee b$
$(a \wedge \neg b) \vee (c \wedge d)$
$\neg a \vee (b \wedge \neg c \wedge d) \vee (a \wedge \neg d)$

Any arbitrary formula in propositional logic can be transformed into disjunctive normal form by application of the laws of distribution, De Morgan's laws, and by removing double negations. It is important to note that this process can lead to exponentially larger formulas which implies that the process in the worst case runs in exponential time. An example for this behavior is the following formula given in ►conjunctive normal form (CNF), which is linear in the number of propositional variables in this form. When transformed into disjunctive normal form (DNF), its size is exponentially larger.

CNF: $(a_0 \vee a_1) \wedge (a_2 \vee a_3) \wedge \cdots \wedge (a_{2n} \vee a_{2n+1})$

DNF: $(a_0 \wedge a_2 \wedge \cdots \wedge a_{2n}) \vee (a_1 \wedge a_2 \wedge \cdots \wedge a_{2n})$
$\vee \cdots \vee (a_1 \wedge a_3 \wedge \cdots \wedge a_{2n+1})$

Recommended Reading

Mendelson, E. (1997). *Introduction to mathematical logic* (4th ed.) (p.30). Chapma & Hall.

Distance

►Similarity Measures

Distance Functions

►Similarity Measures

Distance Measures

►Similarity Measures

Distance Metrics

►Similarity Measures

Distribution-Free Learning

►PAC Learning

Divide-and-Conquer Learning

Synonyms

Recursive partitioning; TDIDT strategy

Definition

The *divide-and-conquer* strategy is a learning algorithm for inducing ►Decision Trees. Its name reflects its key idea, which is to successively partition the dataset into smaller sets (the *divide* part), and recursively call itself on each subset (the *conquer* part). It should not be confused with the *separate-and-conquer* strategy which is used in the ►Covering Algorithm for rule learning.

Cross References

►Covering Algorithm
►Decision Tree

Document Classification

Dunja Mladeni, Janez Brank, Marko Grobelnik
Jožef Stefan Institute, Ljubljana, Slovenia

Synonyms

Document categorization; Supervised learning on text data

Definition

Document classification refers to a process of assigning one or more ►labels for a document from a predefined

set of labels. The main issues in document classification are connected to classification of free text giving document content. For instance, classifying Web documents as being about arts, education, science, etc. or classifying news articles by their topic. In general, one can consider different properties of a document in document classification and combine them, such as document type, authors, links to other documents, content, etc. Machine learning methods applied to document classification are based on general classification methods adjusted to handle some specifics of text data.

Motivation and Background

Documents and text data provide for valuable sources of information and their growing availability in electronic form naturally led to application of different analytic methods. One of the common ways is to take a whole vocabulary of the natural language in which the text is written as a feature set, resulting in several tens of thousands of features. In a simple setting, each feature gives a count of the word occurrences in a document. In this way, text of a document is represented as a vector of numbers. The representation of a particular document contains many zeros, as most of the words from the vocabulary do not occur in a particular document. In addition to the already mentioned two common specifics of text data, having a large number of features and a sparse data representation, it was observed that frequency of words in text generally follows Zipf's law – a small subset of words occur very frequently in texts while a large number of words occur only rarely. Document classification takes these and some other data specifics into account when developing the appropriate classification methods.

Structure of Learning System

Document classification is usually performed by representing documents as word-vectors, usually referred to as the "bag-of-words" or "vector space model" representation, and using documents that have been manually classified to generate a model for document classification (Cohen & Singer, 1996, Mladenić & Grobelnik, 2003; Sebastiani, 2002; Yang, 1997).

Data Representation

In the word-vector representation of a document, a vector of word weights is formed taking all the words occurring in all the documents. Most researchers have used single words when representing text, but there is also research that proposes using additional information to improve classification results. For instance, the feature set might be extended with various multi-word features, e.g., n-grams (sequences of n adjacent words), loose phrases (n-grams in which word order is ignored), or phrases based on grammatical analysis (noun phrases, verb phrases, etc.). Information external to the documents might also be used if it is available; for example, when dealing with Web pages, their graph organization can be a source of additional features (e.g., features corresponding to the adjacency matrix; features based on graph vertex statistics such as degree or PageRank; or features taken from the documents that are adjacent to the current document in the Web graph).

The commonly used approach to weighting words is based on TF–IDF weights where the number of occurrences of the word in the document, referred to as term frequency (TF), is multiplied by the importance of the word with regards to the whole corpus (IDF – inverse document frequency). The IDF weight for the ith word is defined as $\mathrm{IDF}_i = \log(N/\mathrm{DF}_i)$, where N is total number of documents and DF_i is the document frequency of the ith word (the number of documents from the whole corpus in which the ith word appears). The IDF weight decreases the influence of common words (which are not as likely to be useful for discriminating between classes of documents) and favors the less common words. However, the least frequently occurring words are often deleted from the documents as a preprocessing step, based on the notion that if a word that does not occur often enough in the training set cannot be useful for learning and generalization, and would effectively be perceived as noise by the learning algorithm. A stopword list is also often used to delete some of the most common and low-content words (such as "the," "of," "in," etc.) during preprocessing. For many purposes, the vectors used to represent documents should be normalized to unit length so that the vector reflects the contents and themes of the document but not its length (which is typically not relevant for the purposes of document categorization).

Even in a corpus of just a few thousand documents, this approach to document representation can easily lead to a feature space of thousands, possibly tens of thousands, of features. Therefore, feature selection

is sometimes used to reduce the feature set before training. Such questions as whether feature selection is needed and/or beneficial, and which feature selection method should be used, depend considerably on the learning algorithm used; the number of features to be retained depends both on the learning algorithm and on the feature selection method used. For example, naive Bayes tends to benefit, indeed require, heavy feature selection while support vector machines (SVMs) tend to benefit little or nothing from it. Similarly, odds ratio tends to value (some) rare features highly and therefore requires a lot of features to be kept, while information gain tends to score some of the more frequent features highly and thus often works better if a smaller number of features is kept (see also ►Feature Selection in Text Mining).

Due to the large number of features in the original data representation, some of the more computationally expensive feature selection methods from traditional machine learning cannot be used with textual data. Typically, simple feature scoring measures, such as information gain, odds ratio, and chi-squared are used to rank the features and the features whose score falls below a certain threshold are discarded. A better, but computationally more expensive feature scoring method is to train a linear classifier on the full feature set first (e.g., using linear SVM, see below) and rank the features by the absolute value of their weights in the resulting linear model (see also ►Feature Construction in Text Mining).

Classification

Different classification algorithms have been adjusted and applied on text data. A few more popular are described here.

►Naive Bayes based on the multinomial model, where the predicted class for document d is the one that maximizes the posterior probability $P(c\,|\,d) \propto P(c)\Pi_t P(t\,|\,c)$ TF(t,d), where $P(c)$ is the prior probability that a document belongs to class c, $P(t\,|\,c)$ is the probability that a word chosen randomly in a document from class c equals t, and TF(t, d) is the "term frequency," or the number of occurrences of word t in a document d. Where there are only two classes, say c_+ and c_-, maximizing $P(c\,|\,d)$ is equivalent to taking the sign of $\ln P(c_+\,|\,d)/P(c_t\,|\,d)$, which is a linear combination of TF(w, d). Thus, the naive Bayes classifier can be seen as a linear classifier as well. The training consists simply of estimating the probabilities $P(t\,|\,c)$ and $P(c)$ from the training documents.

►Perceptron trains a linear classifier in an incremental way as a neural unit using an additive update rule. The prediction for a document represented by the vector $\mathbf{x}$ is sgn($\mathbf{w}^T\mathbf{x}$), where $\mathbf{w}$ is a vector of weights obtained during training. Computation starts with $\mathbf{w} = 0$, then considers each training example $\mathbf{x}_i$ in turn. If the present $\mathbf{w}$ classifies document $\mathbf{x}_i$ correctly it is left unchanged, otherwise it is updated according to the additive rule: $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$, where y_i is the correct class label of the document $\mathbf{x}_i$, namely $y_i = +1$ for a positive document, $y_i = 1$ for a negative one.

►*SVM* trains a linear classifier of the form sgn ($\mathbf{w}^T\mathbf{x} + b$). Learning is posed as an optimization problem with the goal of maximizing the *margin*, i.e., the distance between the separating hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ and the nearest training vectors. An extension of this formulation, known as the *soft margin*, also allows for a wider margin at the cost of misclassifying some of the training examples. The dual form of this optimization task is a quadratic programing problem and can be solved numerically.

Results of numerous experiments reported in research papers suggest that among the classification algorithms that have been adjusted to text data SVM, Naive Bayes and k-Nearest Neighbor are among the best performing (Lewis, Schapire, Callan, & Ron Papka, 1996). Moreover, experimental evaluation on some standard Reuters news datasets shows that SVM tends to outperform other classifiers including Naive Bayes and Perceptron (Mladenic, Brank, Grobelnik, & Milic-Frayling, 2004).

Evaluation Measures

A characteristic property of machine learning problems arising in document classification is a very unbalanced class distribution. In a typical dataset there may be tens (or sometimes hundreds or thousands) of categories, most of which are very small. When we train a binary (two-class) classification model for a particular category, documents belonging to that category are treated as the positive class while all other documents are treated as the negative class. Thus, the negative class is typically vastly larger as the positive one. These circumstances are not well suited to some traditional machine

learning evaluation measures, such as ►accuracy (if almost all documents are negative, then a useless classifier that always predicts the negative class will have very high accuracy). Instead, evaluation measures from information retrieval are more commonly used, such as ►precision, ►recall, the F_1-measure, the breakeven point (BEP), and the area under the receiver operating characteristic (ROC) curve (see also ►ROC Analysis).

The evaluation of a binary classifier for a given category c on a given test set can be conveniently summarized in a contingency table. We can divide documents into four groups depending on whether they belong to c and whether our classifier predicted them as positive (i.e., supposedly belonging to c) or not:

	Belongs to c	Not in c
Predicted negative	TP (true positives)	FP (false positives)
Predicted negative	FN (false negatives)	TN (true negatives)

Given the number of documents in each of the four groups (TP, FP, TN, and FN), we can compute various evaluation measures as follows:

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP})$$
$$\text{Recall} = \text{TP}_{\text{rate}} = \text{TP}/(\text{TP} + \text{FN})$$
$$\text{FP}_{\text{rate}} = \text{FP}/(\text{TN} + \text{FP})$$
$$F_1 = 2 \bullet \textit{precision} \bullet \textit{recall}/(\textit{precision} + \textit{recall})$$

Thus, precision is the proportion of documents predicted positive that are really positive, while recall is the proportion of positive documents that have been correctly predicted as positive. The F_1 is the harmonic mean of precision and recall; thus, it lies between precision and recall, but is closer to the lower of these two values. This means that a classifier with high F_1 has both good precision and good recall. In practice, there is usually a tradeoff between precision and recall; by making the classifier more liberal (i.e., more likely to predict positive), we can increase recall at the expense of precision, while by making it more conservative (less likely to predict positive) we can usually increase precision at the expense of recall. Often the classification model involves a threshold which can be varied at will to obtain various $\langle \textit{precision}, \textit{recall} \rangle$ pairs. These can be plotted on a chart, resulting in the *precision–recall curve*. As we decrease the threshold (thus making the classifier more liberal), precision decreases and recall increases until at some point precision and recall are equal; this value is known as the (*precision–recall*) *BEP* (Lewis, 1991). Instead of $\langle \textit{precision}, \textit{recall} \rangle$ pairs, one can measure $\langle \text{TP}_{\text{rate}}, \text{FP}_{\text{rate}} \rangle$ pairs, resulting in a *ROC curve* (see ROC analysis). The *area under the ROC curve* is another valuable measure of the classifier quality.

Document classification problems are typically multi-class, multi-label problems, which are treated by regarding each category as a separate two-class classification problem. After training a two-class classifier for each category and evaluating it, the question arises how to combine these evaluation measures into an overall evaluation measure. One way is *macroaveraging*, which means that the values of precision, recall, F_1, or whatever other measure we are interested in are simply averaged over all the categories. Since small categories tend to be much more numerous than large ones, macroaveraging tends to emphasize the performance of our learning algorithm on small categories. An alternative approach is *microaveraging*, in which the contingency tables for individual two-class classifiers are summed up and measures such as precision, recall, and F_1 computed from the resulting aggregated table. This approach emphasizes the performance of our learning algorithm on larger categories.

Cross References

►Classification
►Document Clustering
►Feature Selection
►Perceptron
►Semi-Supervised Text Processing
►Support Vector Machine
►Text Visualization

Recommended Reading

Cohen, W. W., & Singer, Y. (1996). Context sensitive learning methods for text categorization. In *Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 307–315). Zurich: ACM.

Lewis, D. D. (1991). *Representation and learning in information retrieval*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, MA.

Lewis, D. D., Schapire, R. E., Callan, J. P., & Ron Papka, R. (1996) Training algorithms for linear text classifiers. In *Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval SIGIR-1996* (pp. 298–306). New York: ACM.

Mladenic, D., Brank, J., Grobelnik, M., & Milic-Frayling, N. (2004). Feature selection using linear classifier weights: Interaction with classification models. In *Proceedings of the twenty-seventh annual international ACM SIGIR conference on research and development in information retrieval SIGIR-2004* (pp. 234–241). New York: ACM.

Mladenić, D., & Grobelnik, M. (2003). Feature selection on hierarchy of web documents. *Journal of Decision Support Systems, 35*, 45–87.

Sebastiani, F. (2002). Machine learning for automated text categorization. *ACM Computing Surveys, 34*(1), 1–47.

Yang, Y. (1997). An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval, 1*, 67–88.

Document Clustering

Ying Zhao[1], George Karypis[2]
[1]Tsinghua University, Beijing, China
[2]University of Minnesota, Minneapolis, USA

Synonyms

High-dimensional clustering; Text clustering; Unsupervised learning on document datasets

Definition

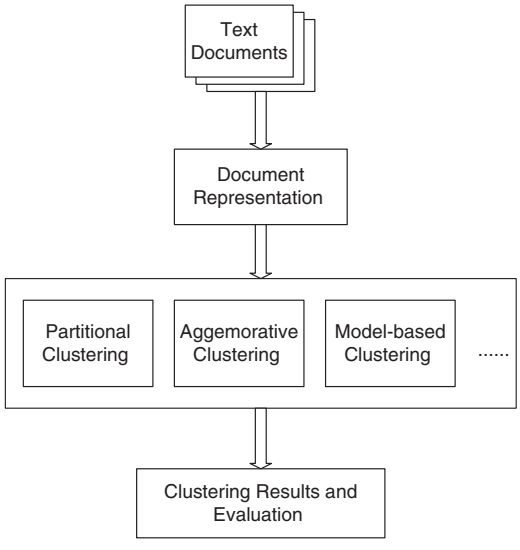
At a high-level, the problem of document clustering is defined as follows. Given a set S of n documents, we would like to partition them into a predetermined number of k subsets $S_1, S_2, \ldots, S_k$, such that the documents assigned to each subset are more similar to each other than the documents assigned to different subsets. Document clustering is an essential part of text mining and has many applications in information retrieval and knowledge management. Document clustering faces two big challenges: the dimensionality of the feature space tends to be high (i.e., a document collection often consists of thousands or tens of thousands unique words) and the size of a document collection tends to be large.

Motivation and Background

►Clustering is an essential component of data mining and a fundamental means of knowledge discovery in data exploration. Fast and high-quality document clustering algorithms play an important role in providing intuitive navigation and browsing mechanisms as well as in facilitating knowledge management. In recent years, we have witnessed a tremendous growth in the volume of text documents available on the Internet, digital libraries, news sources, and company-wide intranets. This has led to an increased interest in developing methods that can help users effectively navigate, summarize, and organize this information with the ultimate goal of helping them find what they are looking for. Fast and high-quality document clustering algorithms play an important role toward this goal as they have been shown to provide both an intuitive navigation/browsing mechanism by organizing large amounts of information into a small number of meaningful clusters as well as to greatly improve the retrieval performance either via cluster-driven dimensionality reduction, term-weighting, or query expansion.

Structure of Learning System

Figure 1 shows the three procedures of transferring a document collection to clustering results that are valuable to users. Original documents are often plain text files, html files, xml files, or a mixture of them. However, most clustering algorithms cannot operate



Document Clustering. Figure 1. Structure of document clustering learning system

on such textual files directly. Hence, *document representation* is needed to prepare original documents into the data model on which clustering algorithms can operate. The actual clustering process can choose clustering algorithms of various kinds: partitional clustering, agglomerative clustering, model-based clustering, etc., depending on the characteristics of the dataset and requirements of the application. We focus on two kinds of clustering algorithms that have been widely used in document clustering: partitional clustering and agglomerative clustering. Finally, clustering results need to be presented with proper quality evaluation to users.

Structure of Document Clustering

In the section, we describe document representation, partitional document clustering, agglomerative document clustering, and clustering evaluation in details.

Document Representation

We introduce here the most widely used document model for clustering and information retrieval: *term frequency-inverse document frequency* (tf-idf) vector-space model (Salton, 1989). In this model, each document d is considered to be a vector in the term-space and is represented by the vector

$$d_{\text{tfidf}} = (tf_1 \log(n/\text{d}f_1), tf_2 \log(n/\text{d}f_2), \ldots, \times tf_m \log(n/\text{d}f_m)),$$

where tf_i is the frequency of the ith term (i.e., term frequency), n is the total number of documents, and df_i is the number of documents that contain the ith term (i.e., document frequency). To account for documents of different lengths, the length of each document vector is normalized so that it is of unit length.

Similarity Measures

We need to define similarity between two documents under tf-idf model, which is essential to a clustering algorithm. Two prominent ways have been proposed to compute the similarity between two documents d_i and d_j. The first method is based on the commonly-used (Salton, 1989) cosine function

$$\cos(d_i, d_j) = d_i^t d_j / (\|d_i\| \, \|d_j\|),$$

and since the document vectors are of unit length, it simplifies to $d_i^t d_j$. The second method computes the similarity between the documents using the Euclidean distance $\text{dis}(d_i, d_j) = \|d_i - d_j\|$. Note that besides the fact that one measures similarity and the other measures distance, these measures are quite similar to each other because the document vectors are of unit length.

Partitional Document Clustering

Partitional algorithms, such as K-means (MacQueen, 1967), K-medoids (Jain & Dubes, 1988), probabilistic (Dempster, Laird, & Rubin, 1977), graph-partitioning-based (Zahn, 1971), or spectral-based (Boley, 1998), find the clusters by partitioning the entire dataset into either a predetermined or an automatically derived number of clusters. A key characteristic of many partitional clustering algorithms is that they use a global criterion function whose optimization drives the entire clustering process. For some of these algorithms the criterion function is implicit (e.g., PDDP, Boley, 1998), whereas for other algorithms (e.g., K-means, MacQueen, 1967) the criterion function is explicit and can be easily stated. This latter class of algorithms can be thought of as consisting of two key components. First is the criterion function that the clustering solution optimizes, and second is the actual algorithm that achieves this optimization.

Criterion Function Criterion functions used in the partitional clustering reflect the underlying definition of the "goodness" of clusters. The partitional clustering can be considered as an optimization procedure that tries to create high quality clusters according to a particular criterion function. Many criterion functions have been proposed and analyzed (Duda, Hart, & Stork, 2001; Jain & Dubes, 1988; Zhao & Karypis, 2004). We list in Table 1 a total of seven different clustering criterion functions. These functions optimize various aspects of intra-cluster similarity, inter-cluster dissimilarity, and their combinations, and represent some of the most widely used criterion functions for document clustering. These criterion functions utilize different views of the underlying collection, by modeling either the objects as vectors in a high-dimensional space, or the collection as a graph.

The $\mathcal{I}_1$ criterion function (1) maximizes the sum of the average pairwise similarities (as measured by the cosine function) between the documents assigned to each cluster weighted according to the size of each

Document Clustering. Table 1 The mathematical definition of various clustering criterion functions

Criterion function	Optimization function	
$\mathcal{I}_1$	maximize $\sum_{i=1}^{k} \frac{1}{n_i} \left(\sum_{v,u \in S_i} \text{sim}(v,u) \right)$	(1)
$\mathcal{I}_2$	maximize $\sum_{i=1}^{k} \sqrt{\sum_{v,u \in S_i} \text{sim}(v,u)}$	(2)
$\mathcal{E}_1$	minimize $\sum_{i=1}^{k} n_i \frac{\sum_{v \in S_i, u \in S} \text{sim}(v,u)}{\sqrt{\sum_{v,u \in S_i} \text{sim}(v,u)}}$	(3)
$\mathcal{G}_1$	minimize $\sum_{i=1}^{k} \frac{\sum_{v \in S_i, u \in S} \text{sim}(v,u)}{\sum_{v,u \in S_i} \text{sim}(v,u)}$	(4)
$\mathcal{G}_2$	minimize $\sum_{r=1}^{k} \frac{\text{cut}(V_r, V - V_r)}{W(V_r)}$	(5)
$\mathcal{H}_1$	maximize $\frac{\mathcal{I}_1}{\mathcal{E}_1}$	(6)
$\mathcal{H}_2$	maximize $\frac{\mathcal{I}_2}{\mathcal{E}_1}$	(7)

The notation in these equations are as follows: k is the total number of clusters, S is the total objects to be clustered, S_i is the set of objects assigned to the ith cluster, n_i is the number of objects in the ith cluster, v and u represent two objects, and $\text{sim}(v,u)$ is the similarity between two objects

cluster. The $\mathcal{I}_2$ criterion function (2) is used by the popular vector-space variant of the K-means algorithm (Cutting, Pedersen, Karger, & Tukey, 1992). In this algorithm each cluster is represented by its centroid vector and the goal is to find the solution that maximizes the similarity between each document and the centroid of the cluster that is assigned to. Comparing $\mathcal{I}_1$ and $\mathcal{I}_2$, we see that the essential difference between them is that $\mathcal{I}_2$ scales the within-cluster similarity by the $\|D_r\|$ term as opposed to the n_r term used by $\mathcal{I}_1$. $\|D_r\|$ is the square-root of the pairwise similarity between all the document in S_r and will tend to emphasize clusters whose documents have smaller pairwise similarities compared to clusters with higher pairwise similarities.

The $\mathcal{E}_1$ criterion function (3) computes the clustering by finding a solution that separates the documents of each cluster from the entire collection. Specifically, it tries to minimize the cosine between the centroid vector of each cluster and the centroid vector of the entire collection. The contribution of each cluster is weighted proportionally to its size so that larger clusters will be weighted higher in the overall clustering solution. $\mathcal{E}_1$ was motivated by multiple discriminant analysis and is similar to minimizing the trace of the between-cluster scatter matrix (Duda et al., 2001).

The $\mathcal{H}_1$ and $\mathcal{H}_2$ criterion functions (6) and (7) are obtained by combining criterion $\mathcal{I}_1$ with $\mathcal{E}_1$, and $\mathcal{I}_2$ with $\mathcal{E}_1$, respectively. Since $\mathcal{E}_1$ is minimized, both $\mathcal{H}_1$ and $\mathcal{H}_2$ need to be maximized as they are inversely related to $\mathcal{E}_1$.

The criterion functions that we described so far view each document as a multidimensional vector. An alternate way of modeling the relations between documents is to use graphs. Two types of graphs are commonly used in the context of clustering. The first corresponds to the document-to-document similarity graph G_s and the second to the document-to-term bipartite graph G_b (Dhillon, 2001; Zha, He, Ding, Simon, & Gu, 2001). G_s is obtained by treating the pairwise similarity matrix of the dataset as the adjacency matrix of G_s, whereas G_b is obtained by viewing the documents and the terms as the two sets of vertices (V_d and V_t) of a bipartite graph. In this bipartite graph, if the ith document contains the jth term, then there is an edge connecting the corresponding ith vertex of V_d to the jth vertex of V_t. The weights of these edges are set using the *tf-idf* model.

Viewing the documents in this fashion, edge-cut-based criterion functions can be used to cluster document datasets. $\mathcal{G}_1$ and $\mathcal{G}_2$ ((4) and (5)) are two such criterion functions that are defined on the similarity and bipartite graphs, respectively. The $\mathcal{G}_1$ function (Ding, He, Zha, Gu, & Simon, 2001) views the clustering process as that of partitioning the documents into groups that minimize the edge-cut of each partition. However, because this edge-cut-based criterion function may have trivial solutions the edge-cut of each cluster is scaled by the sum of the cluster's internal edges (Ding et al., 2001). Note that, $\text{cut}(S_r, S - S_r)$ in (4) is the edge-cut between the vertices in S_r and the rest of the vertices $S - S_r$ and can be re-written as $D_r^t(D - D_r)$ because the similarity between documents is measured using the cosine function. The $\mathcal{G}_2$ criterion function (Dhillon, 2001; Zha et al., 2001) views the clustering problem as a simultaneous partitioning of the documents and the terms so that

it minimizes the normalized edge-cut of the partitioning. Note that, V_r is the set of vertices assigned to the rth cluster and $W(V_r)$ is the sum of the weights of the adjacency lists of the vertices assigned to the rth cluster.

Optimization Method There are many techniques that can be used to optimize the criterion functions described above. They include relatively simple greedy schemes, iterative schemes with varying degree of hill-climbing capabilities, and powerful but computationally expensive spectral-based optimizers (Boley, 1998; Dhillon, 2001; Fisher, 1996; MacQueen, 1967; Zha et al., 2001). We introduce here a simple yet very powerful greedy strategy that has been shown to produce comparable results to those produced by more sophisticated optimization algorithms. In this greedy straggly, a k-way clustering of a set of documents can be computed either directly or via a sequence of repeated bisections. A direct k-way clustering is computed as follows. Initially, a set of k objects is selected from the datasets to act as the *seeds* of the k clusters. Then, for each object, its similarity to these k seeds is computed, and it is assigned to the cluster corresponding to its most similar seed. This forms the initial k-way clustering. This clustering is then repeatedly refined so that it optimizes a desired clustering criterion function. A k-way partitioning via repeated bisections is obtained by recursively applying the above algorithm to compute 2-way clustering (i.e., bisections). Initially, the objects are partitioned into two clusters, then one of these clusters is selected and is further bisected, and so on. This process continues $k-1$ times, leading to k clusters. Each of these bisections is performed so that the resulting two-way clustering solution optimizes a particular criterion function.

Agglomerative Document Clustering

Hierarchical agglomerative algorithms find the clusters by initially assigning each object to its own cluster and then repeatedly merging pairs of clusters until a certain stopping criterion is met. Consider an n-object dataset and the clustering solution that has been computed after performing l merging steps. This solution will contain exactly $n-l$ clusters as each merging step reduces the number of clusters by one. Now, given this $(n-l)$-way clustering solution, the pair of clusters that is selected to be merged next is the one that leads to an $(n-l-1)$-way solution that optimizes a particular criterion function. That is, each one of the $(n-l) \times (n-l-1)/2$ pairs of possible merges is evaluated, and the one that leads to a clustering solution that has the maximum (or minimum) value of the particular criterion function is selected. Thus, the criterion function is *locally* optimized within each particular stage of agglomerative algorithms. Depending on the desired solution, this process continues until either there are only k clusters left or when the entire agglomerative tree has been obtained.

The three basic criteria to determine which pair of clusters to be merged next are single-link (Sneath & Sokal, 1973), complete-link (King, 1967), and group average (i.e., unweighed pair group method with arithmetic mean, UPGMA) (Jain & Dubes, 1988). The single-link criterion function measures the similarity of two clusters by the maximum similarity between any pair of objects from each cluster, whereas the complete-link uses the minimum similarity. In general, both the single- and the complete-link approaches do not work very well because they either base their decisions to a limited amount of information (single-link), or assume that all the objects in the cluster are very similar to each other (complete-link). On the other hand, the group average approach measures the similarity of two clusters by the average of the pairwise similarity of the objects from each cluster and does not suffer from the problems arising with single- and complete-link.

Evaluation of Document Clustering

Clustering results are hard to be evaluated, especially for high dimensional data and without a priori knowledge of the objects' distribution, which is quite common in practical cases. However, assessing the quality of the resulting clusters is as important as generating the clusters. Given the same dataset, different clustering algorithms with various parameters or initial conditions will give very different clusters. It is essential to know whether the resulting clusters are valid and how to compare the quality of the clustering results, so that the right clustering algorithm can be chosen and the best clustering results can be used for further analysis.

In general, there are two types of metrics for assessing clustering results: metrics that only utilize the information provided to the clustering algorithms (i.e., *internal metrics*) and metrics that utilize a priori knowledge of the classification information of the dataset (i.e., *external metrics*).

The basic idea behind internal quality measures is rooted from the definition of clusters. A meaningful clustering solution should group objects into various clusters, so that the objects within each cluster are more similar to each other than the objects from different clusters. Therefore, most of the internal quality measures evaluate the clustering solution by looking at how similar the objects are within each cluster and how well the objects of different clusters are separated. In particular, the *internal similarity* measure, *ISim*, is defined as the average similarity between the objects of each cluster, and the *external similarity* measure, *ESim*, is defined as the average similarity of the objects of each cluster and the rest of the objects in the data set. The ratio between the internal and external similarity measure is also a good indicator of the quality of the resultant clusters. The higher the ratio values, the better the clustering solution is. One of the limitations of the internal quality measures is that they often use the same information both in discovering and in evaluating the clusters.

The approaches based on external quality measures require a priori knowledge of the natural clusters that exist in the dataset, and validate a clustering result by measuring the agreement between the discovered clusters and the known information. For instance, when clustering document datasets, the known categorization of the documents can be treated as the natural clusters, and the resulting clustering solution will be considered correct if it leads to clusters that preserve this categorization. A key aspect of the external quality measures is that they utilize information other than that used by the clustering algorithms. The *entropy* measure is one such metric that looks how the various classes of documents are distributed within each cluster.

Given a particular cluster, S_r, of size n_r, the entropy of this cluster is defined to be

$$E(S_r) = -\frac{1}{\log q}\sum_{i=1}^{q}\frac{n_r^i}{n_r}\log\frac{n_r^i}{n_r}, \qquad (8)$$

where q is the number of classes in the data set, and n_r^i is the number of documents of the ith class that were assigned to the rth cluster. The entropy of the entire clustering solution is then defined to be the sum of the individual cluster entropies weighted according to the cluster size. That is,

$$\text{Entropy} = \sum_{r=1}^{k}\frac{n_r}{n}E(S_r). \qquad (9)$$

A perfect clustering solution will be the one that leads to clusters that contain documents from only a single class, in which case the entropy will be zero. In general, the smaller the entropy values, the better the clustering solution is.

Programs and Data

An illustrative example of a software package for clustering low- and high-dimensional datasets and for analyzing the characteristics of the various clusters is Cluto(Karypis, 2002). Cluto has implementations of the various clustering algorithms and evaluation metrics described in previous sections. It was designed by the University of Minnesota's data mining's group and is available at www.cs.umn.edu/~karypis/cluto. Cluto has been developed as a general purpose clustering toolkit. Cluto's distribution consists of both stand-alone programs (vcluster and scluster) for clustering and analyzing these clusters, as well as a library through which an application program can access directly the various clustering and analysis algorithms implemented in Cluto. Utility tools for preprocessing documents into vector matrices and some sample document datasets are also available at www.cs.umn.edu/~karypis/cluto.

Cross References

►Clustering
►Information Retrieval
►Text Mining
►Unsupervised Learning

Recommended Reading

Boley, D. (1998). Principal direction divisive partitioning. *Data Mining and Knowledge Discovery, 2*(4), 325–344.

Cutting, D. R., Pedersen, J. O., Karger, D. R., & Tukey, J. W. (1992). Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the ACM SIGIR* (pp. 318–329). Copenhagen, Denmark.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, 39*(1), 1–38.

Dhillon, I. S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. In *Knowledge discovery and data mining* (pp. 269–274). San Francisco: Morgan Kaufmann.

Ding, C., He, X., Zha, H., Gu, M., & Simon, H. (2001). *Spectral min-max cut for graph partitioning and data clustering.* Technical report TR-2001-XX, Lawrence Berkeley National Laboratory, University of California, Berkeley, CA.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification.* New York: Wiley.

Fisher, D. (1996). Iterative optimization and simplification of hierarchical clusterings. *Journal of Artificial Intelligence Research, 4*, 147–180.
Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Englewood Cliffs, NJ: Prentice-Hall.
Karypis, G. (2002). CLUTO: *A clustering toolkit*. Technical report 02-017, Department of Computer Science, University of Minnesota. Available at http://www.cs.umn.edu/~cluto.
King, B. (1967). Step-wise clustering procedures. *Journal of the American Statistical Association, 69*, 86–101.
MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th symposium on mathematical statistics and probability* (pp. 281–297). Berkeley, CA: University of California Press.
Salton, G. (1989). *Automatic text processing: The transformation, analysis, and retrieval of information by computer*. Reading, MA: Addison-Wesley.
Sneath, P. H., & Sokal, R. R. (1973). *Numerical taxonomy*. San Francisco: Freeman.
Zahn, K. (1971). Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers, (C-20)*, 68–86.
Zha H., He X., Ding C., Simon H., and Gu M. Bipartite graph partitioning and data clustering. In *Proceedings of the International Conference on Information and Knowledge Management*, 2001.
Zhao, Y., & Karypis, G. (2004). Criterion functions for document clustering: Experiments and analysis. *Machine Learning, 55*, 311–331.

Dual Control

►Bayesian Reinforcement Learning
►Partially Observable Markov Decision Process

Duplicate Detection

►Entity Resolution

Dynamic Bayesian Network

►Learning Graphical Models

Dynamic Decision Networks

►Partially Observable Markov Decision Processes

Dynamic Memory Model

SUSAN CRAW
The Robert Gordon University, Scotland, UK

Synonyms

Dynamic memory model; Memory organization packets

Definition

Schank's dynamic memory model (Schank, 1982) was designed to capture knowledge of specific experiences. Schank's memory organization packets (MOPs) and Kolodner's E-MOPs (episodic MOPS) (Kolodner, 1983) provide templates about typical scenes. For a restaurant scene these might identify "being seated," "ordering," and "paying."

Cross References

►Case-Based Reasoning

Recommended Reading

Kolodner, J. (1983). Reconstructive memory. A computer model. *Cognitive Science, 7*(4), 281–328.
Schank, R. S. (1982). *Dynamic Memory : A theory of reminding and learning in computers and people*. New York: Cambridge University Press.

Dynamic Programming

MARTIN L. PUTERMAN[1], JONATHAN PATRICK[2]
[1]University of British Columbia, Vancouver, Canada
[2]University of Ottawa, Ottawa, Canada

Definition

Dynamic programming is a method for modeling a sequential decision process in which past decisions impact future possibilities. Decisions can be made at fixed discrete time intervals or at random time intervals triggered by some change in the system. The decision process can last for a finite period of time or run indefinitely – depending on the application. Each time a decision needs to be made, the decision-maker (referred

to as "he" in this chapter with no sexist connotation intended) views the current ▶state of the system and chooses from a known set of possible ▶actions. As a result of the state of the system and the action chosen, the decision-maker receives a ▶reward (or pays a ▶cost) and the system evolves to a new state based on known probabilities. The challenge faced by the decision-maker is to choose a sequence of actions that will lead to the greatest reward over the length of the decision-making horizon. To do this, he needs to consider not only the current reward (or cost) for taking a given action but the impact such an action might have on future rewards. A ▶policy is a complete sequence of decisions that dictates what action to take in any given state and at any given time. Dynamic programming finds the optimal policy by developing mathematical recursions that decompose the multi-decision problem into a series of single-decision problems that are analytically or computationally more tractable.

Background and Motivation

The earliest concepts that later developed into dynamic programming can be traced back to the calculus of variations problems in the seventeenth century. However, the modern investigation of stochastic sequential decision problems arguably dates back to work by Wald in 1947 on sequential statistical analysis. At much the same time, Pierre Masse was analyzing similar problems applied to water resource management in France. However, the major name associated with dynamic programming is that of Richard Bellman who established the optimality equations that form the basis of dynamic programming.

It is not hard to demonstrate the potential scope of dynamic programming. Table 1 gives a sense of the breadth of application as well as highlighting the stochastic nature of most instances.

Structure of the Learning System

A dynamic program is a general representation of a sequential decision problem under uncertainty about the future and is one of the main methods for solving Markov Decision Problems (see ▶Markov Decision Process). Like a decision tree, it models a process where the decision we make "today" impacts where we end up tomorrow and therefore what decisions are available to us tomorrow. It has distinct advantages over a decision tree in that:

- It is a more compact representation of a decision process
- It enables efficient calculation
- It allows exploration of the structural properties of optimal decisions
- It can analyze and solve problems with infinite or indefinite time horizons

The Finite Horizon Setting

A finite horizon MDP, is a decision process with a known end date. Thus, the decision-maker is faced with the task of making a finite sequence of decisions at fixed intervals. The MDP model is based on five elements:

▶Decision epochs: Sequences of decision times $n = 1, \ldots, N$ (in the infinite horizon we set $N = \infty$). In a discrete time MDP, these decision times happen at regular, fixed intervals while in a continuous time model they occur at random times triggered by a change in the system. The time between decision epochs is called a period.

▶State space: States represent the possible system configurations facing the decision-maker at each decision epoch. They contain all information available to the decision-maker at each decision epoch. The state space, S, is the set of all such states (often assumed to be finite). In choosing the state space, it is important to include all the information that may be relevant in determining a decision and that may change from decision epoch to decision epoch.

▶Actions: Actions are the available choices for the decision-maker at any given decision epoch, in any given state. $A(s)$ is the set of all actions available in state s (usually assumed to be finite for all s). No action is taken in the final decision epoch N.

▶Transition probabilities: The probability of being in state s' at time $t+1$, given you take action a from state s at time t, is written as $p_t(s'|s, a)$. It clearly makes sense to allow the transition probabilities to be conditional upon the current state and the action taken.

▶Rewards/costs: In most MDP applications, the decision-maker receives a reward each period. This reward can depend on the current state, the action taken, and the next state and is denoted by $r_t(s, a, s')$. Since a decision must be made before knowing the next state, s', the

Dynamic Programming. Table 1 Dynamic programming applications

Application	System state	Actions	Rewards	Stochastic aspect
Capacity	Size of plant	Maintain or add capacity	Costs of expansion and production at current capacity	Demand for a product
Cash mgt	Cash available	Borrow or invest	Transaction costs and less interest	External demand for cash
Catalog mailing	Customer purchase record	Type of catalog to send, if any	Purchases in current period less mailing costs	Customer purchase amount
Clinical trials	Number of successes with each treatment	Stop or continue the trial	Costs of treatment and incorrect decisions	Response of a subject to treatment
Economic growth	State of the economy	Investment or consumption	Utility of consumption	Effect of investment
Fisheries mgt	Fish stock in each age class	Number of fish to to harvest	Value of the catch	Population size
Forest mgt	Size and condition of stand	Harvesting and reforestation activities	Revenues and less harvesting costs	Stand growth and price fluctuation
Gambling	Current wealth	Stop or continue playing	Cost of playing	Outcome of the game
Inventory control	Stock on hand	Order additional stock	Revenue per item sold and less ordering, holding, and penalty costs	Demand for items
Project selection	Status of each project	Project to invest in at present	Return from investing in project	Change in project status
Queueing control	Number in the queue	Accept/reject new customers or control service rate	Revenue from serving customers and less delay costs	Interarrival times and service times
Reliability	Age or status of equipment	Inspect and repair or replace if necessary	Inspection, repair, and failure costs	Failure and deterioration
Reservations	Number of confirmed reservations	Accept, wait-list, or reject new reservation	Profit from satisfied reservations and less overbooking penalties	Number of arrivals and the demand for reservations
Scheduling	Activities completed	Next activity to schedule	Cost of activity	Length of time to complete activity
Selling an asset	Current offer	Accept or reject the offer	The offer is less than the cost of holding the asset for one period	Size of the offer
Water resource management	Level of water in each reservoir	Quantity of water to release	Value of power generated	Rainfall and run-off

MDP formulation deals with the expected reward:

$$r_t(s,a) = \sum_{s' \in S} r_t(s,a,s') p_t(s'|s,a).$$

We also define the terminal rewards as $r_N(s)$ for being in state s at the final decision epoch. These are independent of the action since no action is taken at that point.

The objective in the finite horizon model is to maximize total expected reward:

$$\max \left\{ E\left[\sum_{t=1}^{N} r_t(s_t, a_t, s_{t+1}) + r_N(s_N) | s_1 = s \right] \right\}. \quad (1)$$

At any given time t, the decision-maker has observed the history up to time t, represented by $h_t = (s_1, a_1, s_2, a_2, \ldots, a_{t-1}, s_t)$, and needs to choose a_t in such a way as to maximize (1). A ▶decision rule, d_t, determines what action to take, based on the history to date at a given decision epoch and for any possible state. It is ▶deterministic if it selects a single member of $A(s)$ with probability 1 for each $s \in S$ and for a given h_t, and it is ▶randomized (▶randomized decision rule) if it selects a member of $A(s)$ at random with probability $q_{d_t(h_t)}(a)$. It is Markovian (▶Markovian decision rule) if it depends on h_t only through s_t. That is, $d_t(h_t) = d_t(s_t)$.

A ▶policy, $\pi = (d_1, \ldots, d_{N-1})$, denotes a complete sequence of decision rules over the whole horizon. It can be viewed as a "contingency plan" that determines the action for each possible state at each decision epoch. One of the major results in MDP theory is that, under reasonable conditions, it is possible to prove that there exists a Markovian, deterministic policy that attains the maximum total expected reward. Thus, for the purposes of this chapter we will concentrate on this subset of all policies.

If we define, $v_t(s)$ as the expected total reward from time t to the end of the planning horizon, given that at time t the system occupies state s, then a recursion formula can be built that represents v_t in terms of v_{t+1}. Specifically,

$$v_t(s) = \max_{a \in A(s)} \left\{ r_t(s,a) + \sum_{s' \in S} p(s'|s,a) v_{t+1}(s') \right\} \quad (2)$$

This is often referred to as the ▶Bellman equation, named after Richard Bellman who was responsible for the seminal work in this area. It breaks the total reward at time t, into the immediate reward $r_t(s,a)$ and the total future expected reward, $\sum_{s' \in S} p(s'|s,a) v_{t+1}(s')$. Define $A^*_{s,t}$ as the set of actions that attain the maximum in (2) for a given state s and decision epoch t. Then the finite horizon discrete time MDP can be solved through the following backward induction algorithm.

Backward Induction Algorithm

- Set $t = N$ and $v_t(s) = r_N(s) \quad \forall s \in S$ (since there is no decision at epoch N and no future epochs, it follows that the optimal reward-to-go function is just the terminal reward).
- Let $t = t - 1$ and compute for each $s \in S_t$

$$v_t(s) = \max_{a \in A(s)} \left\{ r_t(s,a) + \sum_{s' \in S} p(s'|s,a) v_{t+1}(s') \right\}.$$

- For each $s \in S_t$, compute $A^*_{s,t}$ by solving

$$\text{argmax}_{a \in A(s)} \left\{ r_t(s,a) + \sum_{s' \in S} p(s'|s,a) v_{t+1}(s') \right\}.$$

- If $t = 1$ then stop else return to step 2.

The function $v_1(s)$ is the maximum expected reward over the entire planning horizon given the system starts in state s. The optimal policy is constructed by choosing a member of $A^*_{s,t}$ for each $s \in S$ and $t \in \{1, \ldots, N\}$. In essence, the algorithm solves a complex N-period decision problem by solving N simple 1-period decision problems.

Example – inventory control: Periodically (daily, weekly, or monthly), an inventory manager must determine how much of a product to stock in order to satisfy random external demand for the product. If too little is in stock, potential sales are lost. Conversely, if too much is on hand, a cost for carrying inventory is incurred. The objective is to choose an ordering rule that maximizes expected total profit (sales minus holding and ordering costs) over the planning horizon. To formulate an MDP model of this system requires precise assumptions such as:

- The decision regarding the quantity to order is made at the beginning of each period and delivery occurs instantaneously.

- Demand for the product arrives throughout the period, but all orders are filled on the last day of the period.
- If demand exceeds the stock on hand, potential sales are lost.
- The revenues, costs and demand distribution are the same each period.
- The product can only be sold in whole units.
- The warehouse has a capacity for M units.

(These assumptions are not strictly necessary but removing them leads to a different formulation.) Decisions epochs correspond to the start of a period. The state, $s_t \in \{0, \ldots, M\}$, represents the inventory on hand at the start of period t and the action, $a_t \in \{0, 1, 2, \ldots, M - s\}$, is the number of units to order that period; the action 0 corresponds to not placing an order. Let D_t represent the random demand throughout period t and assume that the distribution of demand is given by $p_t(d) = P(D_t = d), d = 0, 1, 2, \ldots$. The cost of ordering u units is $O(u) = K + c(u)$ (a fixed cost plus variable cost) and the cost of storing u units is $h(u)$, where $c(u)$ and $h(u)$ are increasing functions in u. We will assume that left-over inventory at the end of the planning horizon has value $g(u)$ and that the sale of u units yields a revenue of $f(u)$. Thus, if there are u units on hand at decision epoch t, the expected revenue is

$$F_t(u) = \sum_{j=0}^{u-1} f(j) p_t(j) + f(u) P(D_t \geq u).$$

The expected reward is therefore

$$r_t(s, a) = F(s + a) - O(a) - h(s + a)$$

and the terminal rewards are $r_N(s, a) = g(s)$. Finally, the transition probabilities depend on whether or not there is enough stock on hand, $s + a$, to meet the demand for that month, D_t. Specifically,

$$p_t(j|s, a) = \begin{cases} 0 & \text{if } j > s + a, \\ p_t(j) & \text{if } j = s + a - D_t, s + a \leq M, \\ & \quad s + a > D_t, \\ \sum_{d=s+a}^{\infty} p_t(d) & \text{if } j = 0, s + a \leq M, s + a \leq D_t. \end{cases}$$

Solving the finite horizon version of this problem through backward induction reveals a simple form to the optimal policy referred to as an (s, S) policy. Specifically, if at time t, the inventory is below some number s^t then it is optimal to order a quantity that raises the inventory level to S^t. It has been shown that a structured policy of this type is optimal for several variants of the inventory management problem with a fixed ordering cost. Many variants of this problem have been studied; these models underly the field of supply chain management.

The Infinite Horizon Setting

In the infinite (or indefinite) horizon setting, the backward induction algorithm described above no longer suffices as there are no terminal rewards with which to begin the process.

In most finite horizon problems, the optimal policy begins to look the same at each decision epoch as the horizon is pushed further and further into the future. For instance, in the inventory example above, $s^t = s^{t+1}$ and $S^t = S^{t+1}$ if t is sufficiently removed from the end of the horizon. The form of the optimal policy only changes as the end of the time horizon approaches. Thus, if there is no fixed time horizon, we should expect the optimal policy to be *stationary* in most cases. We call a policy *stationary* if the same decision rule is applied at each decision epoch (i.e., $d_t = d \; \forall \; t$). One necessary assumption for this to be true is that the rewards and transition probabilities are independent of time (i.e., $r_t(s, a) = r(s, a)$ and $p_t(s'|s, a) = p(s'|s, a) \; \forall \; s, s' \in S$ and $a \in A(s)$). For the infinite horizon MDP, the theory again proves that under mild assumptions there exists an optimal policy that is *stationary, deterministic*, and *Markovian*. This fact greatly simplifies the process of finding the optimal policy as we can concentrate on a small subset of all potential policies.

The set up for the infinite horizon MDP is entirely analogous to the finite horizon setting with the same ▶decision epochs, ▶states, ▶actions, ▶rewards, and ▶transition probabilities (with the last two assumed to be independent of time).

The most obvious objective is to extend the finite horizon objective to infinity and seek to find the policy, π, that maximizes the total expected reward:

$$v^{\pi}(s) = \lim_{N \to \infty} \left\{ E_s^{\pi} \left[\sum_{t=1}^{N} r(s_t, a_t) \right] \right\}. \qquad (3)$$

This, however, is problematic since

1. The sum may be infinite for some or all policies
2. The sum may not even exist, or
3. Even if the sum exists, there may be no maximizing policy

In the first case, just because all (or a subset of all) policies lead to infinite reward in the long run does not mean that they are all equally beneficial. For instance, one may give a reward of $100 each epoch and the other $1 per epoch. Alternatively, one may give large rewards earlier on while another gives large rewards only much later. Generally speaking, the first is more appealing but the above objective function will not differentiate between them. Secondly, the limit may not exist if, for instance, the reward each decision epoch oscillates between 1 and −1. Thirdly, there may be no maximizing policy simply because there is an infinite number of policies and thus there may be an infinite sequence of policies that converges to a maximum limit but never reaches it. Thus, instead we look to maximize either the *total expected discounted reward* or the *expected long run average reward* depending on the application.

Let $\lambda \in (0,1)$ be a discount factor. Assuming the rewards are bounded (i.e., there exists an M such that $|r(s,a)| < M \quad \forall (s,a) \in S \times A(s)$), the *total expected discounted reward* for a given policy π is defined as

$$v_\lambda^\pi(s) = \lim_{N\to\infty} E_s^\pi \left\{ \sum_{t=1}^{N} \lambda^{t-1} r(s_t, d_t(s_t)) \right\} = E_s^\pi \left\{ \sum_{t=1}^{\infty} \lambda^{t-1} r(s_t, d_t(s_t)) \right\}.$$

Since, $\lambda < 1$ and the rewards are bounded, this limit always exists. The second objective is the *expected average reward* which, for a given policy π, is defined as

$$g^\pi(s) = \lim_{N\to\infty} \frac{1}{N} E_s^\pi \left\{ \sum_{t=1}^{N} r(s_t, d_t(s_t)) \right\}.$$

Once again, we are dealing with a limit that may or may not exist. As we will see later, whether the above limit exists depends on the structure of the Markov chain induced by the policy.

Let us, at this point, formalize what we mean by an optimal policy. Clearly, that will depend on which objective function we choose to use. We say that

- π^* is *total reward optimal* if $v^{\pi^*}(s) \geq v^\pi(s) \; \forall s \in S$ and $\forall \pi$.
- π^* is *discount optimal* if $v_\lambda^{\pi^*}(s) \geq v_\lambda^\pi(s) \; \forall s \in S$ and $\forall \pi$.
- π^* is *average optimal* if $g^{\pi^*}(s) \geq g^\pi(s) \; \forall s \in S$ and $\forall \pi$.

For simplicity, we introduce matrix and vector notation. Let $r_d(s) = r(s, d(s))$ and $p_d(j|s) = p(j|s, d(s))$. Thus r_d is the vector of rewards for each state under decision rule d, and P_d is the transition matrix of states under decision rule d. We will now take a more in-depth look at the infinite horizon model with the total expected discounted reward as the optimality criterion.

Solving the Infinite Horizon Discounted MDP

Given a Markovian, deterministic policy $\pi = (d_1, d_2, d_3, \ldots)$ and defining $\pi_k = (d_k, d_{k+1}, \ldots)$ we can compute

$$\begin{aligned} v_\lambda^\pi(s) &= E_s^{\pi_1}\left[\sum_{t=1}^{\infty} \lambda^{t-1} r(s_t, d_t(s_t)) \right] \\ &= E_s^{\pi_1}\left[r(s, d_1(s)) + \lambda \sum_{t=2}^{\infty} \lambda^{t-2} r(s_t, d_t(s_t)) \right] \\ &= r(s, d_1(s)) + \lambda \sum_{j\in S} p_{d_1}(j|s) E_j^{\pi_2}\left[\sum_{t=1}^{\infty} \lambda^{t-1} r(s_t, d_t(s_t)) \right] \\ &= r(s, d_1(s)) + \lambda \sum_{j\in S} p_{d_1}(j|s) v_\lambda^{\pi_2}(j). \end{aligned}$$

In matrix notation,

$$v_\lambda^{\pi_1} = r_{d_1} + \lambda P_{d_1} v_\lambda^{\pi_2}.$$

If we follow our supposition that we need to only consider *stationary* policies (so that the same decision rule is applied to every decision epoch), $\pi = d^\infty = (d, d, \ldots)$, then this results in

$$v_\lambda^{d^\infty} = r_d + \lambda P_d v_\lambda^{d^\infty}.$$

This implies that the value function generated by a stationary policy satisfies the equation:

$$\begin{aligned} v &= r_d + \lambda P_d v \\ \Rightarrow v &= (I - \lambda P_d)^{-1} r_d. \end{aligned}$$

The inverse above always exists since P_d is a probability matrix (so that its spectral radius is less than or equal to 1) and $\lambda \in (0,1)$. Moving to the maximization problem of finding the optimal policy, we get the recursion formula

$$v(s) = \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{j \in S} p(s|s,a) v(j) \right\}. \quad (4)$$

Note that the right hand side can be viewed as a function of a vector v (given r, p, λ). We define a vector-valued function

$$Lv = \max_{d \in D^{MD}} \left\{ r_d + \lambda P_d v \right\},$$

where D^{MD} is the set of all Markovian, ▶deterministic decision rules. There are three methods for solving the above optimization problem in order to determine the optimal policy. The first method, called *value iteration*, creates a sequence of approximations to the value function that eventually converges to the value function associated with the optimal policy.

Value Iteration

1. Start with an arbitrary $|S|$-vector v^0. Let $n = 0$ and choose $\epsilon > 0$ to be small.
2. For every $s \in S$, compute $v^{n+1}(s)$ as

$$v^{n+1}(s) = \max_{a \in A(s)} \left\{ r(s,a) + \sum_{j \in S} \lambda p(j|s,a) v^n(j) \right\}.$$

3. If $\max_{s \in S} |v^{n+1}(s) - v^n(s)| \geq \epsilon(1-\lambda)/2\lambda$ let $n \to n+1$ and return to step 2.
4. For each $s \in S$, choose

$$d_\epsilon(s) \in \operatorname{argmax}_{a \in A(s)} \left\{ r(s,a) + \sum_{j \in S} \lambda p(j|s,a) v^{n+1}(j) \right\}.$$

It has been shown that value iteration identifies a policy with expected total discounted reward within ϵ of optimality in a finite number of iterations. Many variants of value iteration are available such as using different stopping criteria to accelerate convergence or combining value iteration with the policy iteration algorithm described below.

A second algorithm, called *policy iteration*, iterates through a sequence of policies eventually converging to the optimal policy.

Policy Iteration

1. Set $d_0 \in D$ to be an arbitrary policy. Let $n = 0$.
2. (Policy evaluation) Obtain v^n by solving

$$v^n = (I - \lambda P_{d_n})^{-1} r_{d_n}.$$

3. (Policy improvement) Choose d_{n+1} to satisfy

$$d_{n+1} \in \operatorname{argmax}_{d \in D} \{ r_d + \lambda P_d v^n \}$$

componentwise. If d_n is in this set, then choose $d_{n+1} = d_n$.
4. If $d_{n+1} = d_n$, set $d^* = d_n$ and stop. Otherwise, let $n \to n+1$ and return to (2).

Note that value iteration and policy iteration have different conceptual underpinnings. Value iteration seeks a *fixed point* of the operator L using successive approximations while policy iteration can be viewed as using Newton's Method to solve $Lv - v = 0$.

Finally, a third method for solving the discounted infinite horizon MDP takes advantage of the fact that, because L is monotone, if $Lv \leq v$ then $L^2 v \leq Lv$ and more generally, $L^k v \leq v$. Thus, induction implies that the value function of the optimal policy, v_λ^* is less than or equal to v for any v, where $Lv \leq v$. We define the set $U := \{v \in V | Lv \leq v\}$. Then, not only is v_λ^* in the set U, it is also the smallest element of U. Therefore, we can solve for v_λ^* by solving the following linear program:

$$\min_v \sum_{s \in S} \alpha(s) v(s)$$

subject to

$$v(s) \geq r(s,a) + \lambda \sum_{j \in S} p(j|s,a) v(j) \quad \forall s \in S, \ \ a \in A_s.$$

(Note that the above set of constraints is equivalent to $Lv \leq v$.) We call this the primal LP. The coefficients $\alpha(s)$ are arbitrarily chosen. The surprising fact is that the solution to the above LP will be v_λ^* for any strictly positive α.

We can construct the dual to the above primal to get

$$\max_X \sum_{s \in S} \sum_{a \in A_s} r(s,a) X(s,a)$$

subject to

$$\sum_{a\in A_j} X(j,a) - \sum_{s\in S}\sum_{a\in A_s} \lambda p(j|s,a)X(s,a) = \alpha(j) \quad \forall j \in S$$
$$X(s,a) \geq 0 \quad \forall s \in S, \quad a \in A_s.$$

Let $(X(s,a) : s \in S, a \in A_s)$ be a feasible solution for the dual (i.e., satisfies the constraints but not necessarily optimal). Every such feasible solution corresponds to a randomized Markov policy d^∞ and vice versa. Furthermore, for a given feasible solution, X, and the corresponding policy d^∞, $X(s,a)$ represents the expected total number of times you will be in state s and take action a following policy d^∞ before stopping in the indefinite horizon problem. Thus, the objective in the dual can be interpreted as the total expected reward over the length of the indefinite horizon. The strong law of duality states that at the optimal solution the objective functions in the primal and dual will be equal. But we already know that at the optimal, the primal objective will correspond to a weighted sum of $v^*_\lambda(s), s \in S$, which is the total expected discounted reward over the infinite (or indefinite) horizon given you start in state s. Thus our interpretations for the primal and dual variables coincide.

Solving the Infinite Horizon Average Reward MDP

Recall that in the average reward model, the objective is to find the policy that has the maximum average reward, often called the *gain*. The gain of a policy can be written as

$$g^\pi(s) = \lim_{n\to\infty} \frac{1}{N} v^\pi_{N+1} = \lim_{n\to\infty} \frac{1}{N} \sum_{n=1}^{N} [P^{n-1}_\pi r_{d_s}](s). \quad (5)$$

As mentioned earlier, the major drawback is that for a given policy π, the gain may not even exist. An important result, however, states that if we confine ourselves to stationary policies, we can in fact be assured that the gain is well defined. Our ability to solve a given infinite horizon average reward problem depends on the form of the Markov chains induced by the deterministic, stationary policies available in the problem. Thus, we divide the set of average reward MDPs according to the structure of the underlying Markov chains. We say that an MDP is

- *Unichain* if the transition matrix corresponding to *every* deterministic stationary policy is unichain, that is, it consists of a single recurrent class plus a possibly empty set of transient states, or
- *Multichain* if the transition matrix corresponding to *at least one* stationary policy contains two or more closed irreducible recurrent classes

If an MDP is unichain, then the gain for any given stationary, deterministic policy can be defined by a single number (independent of starting state). This makes intuitive sense since if we assume that it is possible to visit every state from every other one (possibly minus some set of transient states that may be visited initially but will eventually be abandoned) then it would seem reasonable to assume that over the infinite horizon the initial starting state would not impact the average reward. However, if the initial state impacts what set of states can be visited in the future (i.e., the MDP is multichain) then clearly it is likely that the expected average reward will be dependent on the initial state.

If the average reward MDP is unichain then the *gain* can be uniquely determined by solving

$$v(s) = \max_{a\in A(s)} \left\{ r(s,a) - g + \sum_{s'\in S} p(s'|s,a)v(s') \right\}. \quad (6)$$

Notice that the above equation has $|S|+1$ unknowns but only $|S|$ equations. Thus, v is not uniquely determined. To specify v uniquely, it is sufficient to set $v(s') = 0$ for some $s' \in S$. If this is done, then $v(s)$ is called the relative value function and $v(j) - v(k)$ is the difference in expected total reward obtained in using an optimal policy and starting in state j as opposed to state k. It is also often represented by the letter h and called the *bias*.

As in the discounted infinite horizon MDP, there are three potential methods for solving the average reward case. We present only policy iteration here and refer the reader to the recommended readings for value iteration and linear programming.

Policy Iteration

1. Set $n = 0$, and choose an arbitrary decision d_n.
2. (Policy evaluation) Solve for g_n, v_n:

$$0 = r_{d_n} - ge + (P_{d_n} - I)v.$$

3. Choose d_{n+1} to satisfy

$$d_{n+1} \in \text{argmax}_{d \in D}\{r_d + P_d v_n\}.$$

Setting $d_{n+1} = d_n$ if possible.
4. If $d_{n+1} = d_n$, stop, set $d^* = d_n$. Else, increment n by 1 and return to step 2.

As mentioned earlier, the equation in step 2 fails to provide a unique v_n since we have $|S| + 1$ unknowns and only $|S|$ equations. We therefore need an additional equation. Any one of the following three will suffice:

1. Set $v_n(s_0) = 0$ for some fixed $s_0 \in S$.
2. Choose v_n to satisfy $P^*_{d_n} v_n = 0$.
3. Choose v_n to satisfy $-v_n + (P_d - I)w = 0$ for some $w \in V$.

Continuous Time Models

So far, we have assumed that decision epochs occur at regular intervals but clearly in many applications this is not the case. Consider, for instance, a queueing control model where the service rate can be adjusted in response to the size of the queue. It is reasonable to assume, however, that changing the service rate is only possible following the completion of a service. Thus, if the service time is random then the decision epochs will occur at random time intervals. We will therefore turn our attention now to systems in which the state changes and decision epochs occur at random times. At the most general level, decisions can be made at any point in time but we will focus on the subset of models for which decision epochs only occur at state transitions. It turns out that this is usually sufficient as the added benefit of being able to change decisions apart from state changes does not generally improve performance. Thus, the models we study generalize the discrete time MDP models by:

1. Allowing, or requiring, the decision-maker to choose actions whenever the system changes state
2. Modeling the evolution of the system in continuous time, and
3. Allowing the time spent in a particular state to follow an arbitrary probability distribution

Semi-Markov decision processes (SMDP) are continuous time models where decisions are made at some but not necessarily all state transitions. The most common subset of these, called exponential SMDPs, are SMDPs where the intertransition times are exponentially distributed.

We distinguish between two processes:

1. The natural process that monitors the state of the system as if it were observed continually through time and
2. The embedded Markov chain that monitors the evolution of the system at the decision epochs only

For instance, in a queueing control model one may decide only to change the rate of service every time there is an arrival. Then the embedded Markov chain would only keep track of the system at each arrival while the natural process would keep track of all state changes – including both arrivals and departures.

While the actions are generally only going to depend on the state of the system at each decision epoch, it is possible that the rewards/costs to the system may depend on the natural process. Certainly, in the queueing control model the cost to the system would go down as soon as a departure occurs. In discrete models it was sufficient to let the reward depend on the current state s and the current action a and possibly the next state s'. However, in an SMDP, the natural process may change between now and the next decision epoch and moreover, the time the process stays in a given state is no longer fixed. Thus we need to consider two types of rewards/costs. First, a lump sum reward, $k(s, a)$, for taking action a when in state s. Second, a reward *rate*, $c(j, s, a)$, paid out for each time unit that the natural process spends in state j until the next decision epoch when the state at the last decision epoch was s and the action taken was a. Note that if we insist that every state transition triggers a decision epoch, we can reduce this to $c(s, a)$ since the system remains in s until the next decision epoch.

Before we can state our objective we need to determine what we mean by discounting. Again, because we are dealing with continuous time so that decision epochs are not evenly spaced, it is not sufficient to have a fixed discount factor λ. Instead, we will discount future rewards at rate $e^{-\alpha t}$, for some $\alpha > 0$. If we let $\lambda = e^{-\alpha}$

(the discount rate for one time unit) then $\alpha = 0.11$ corresponds to $\lambda = 0.9$. Thus an α around 0.1 is commonly used.

We can now state our objective. We look to find a policy that maximizes the total expected discounted reward over the infinite horizon. There is an average reward model for continuous time models as well but we will not discuss that here. Given a policy π we can write its total expected discounted reward as:

$$v_\alpha^\pi(s) = E_s^\pi\left[\sum_{n=0}^{\infty} e^{-\alpha\sigma_n}\left(K(X_n, Y_n) + \int_{\sigma_n}^{\sigma_{n+1}} e^{-\alpha(t-\sigma_n)} c(W_t, X_n, Y_n)\, dt\right)\right], \quad (7)$$

where X_n and Y_n are the random variables that represent the state and action at time n respectively, W_t is the random variable that represents the state of the natural process at time t, and σ_n is the random time of the nth decision epoch. Again, if we assume that each state transition triggers a decision epoch, $X_n = W_t$ for all $t \in [\sigma_n, \sigma_{n+1})$. We seek to find a policy π such that

$$v_\alpha^\pi(s) = v_\alpha^*(s) = \max_{\pi \in \Pi^{HR}} v_\alpha^\pi(s) \quad (8)$$

for all $s \in S$. Perhaps surprisingly, (7) can be reduced to one that has the same form as in the discrete time case for any SMDP. As a consequence, all the theory and the algorithms that worked in the discrete version can be transferred to the continuous model! Again, we refer the reader to the recommended readings for the details.

Extensions

►Partially Observed MDPs

In some instances, the state of the system may not be directly observable but instead, the decision-maker receives a signal from the system that provides information about the state. For example, in medical decision making, the health care provider will not know the patient's true health status but will have on hand some diagnostic information that may be related to the patient's true health. These problems are modeled from a Bayesian perspective. The decision-maker uses the signal to update his estimate of the probability distribution of the system state. He then bases his decision on this probability distribution. The computational methods for solving partially observed MDPs are significantly more complex than in the fully observable case and only small problems have been solved numerically.

Parameter-Adaptive Dynamic Programming

Often the transition probabilities in an MDP are derived from a system model, which is determined by a few parameters. Examples include demand distributions in inventory control and arrival and/or service distributions in queueing systems. In these cases the forms of the distributions are known (for example, Poisson for demand models and exponential for arrival or service models) but their parameter values are not. Herein, the decision-maker seeks a policy that combines *learning* with *control*. A Bayesian approach is used. The parameter is related to the system state through a likelihood function and after observing the system state, the probability distribution on the parameter is updated. This updated probability distribution provides the basis for choosing a policy.

Approximate Dynamic Programming

Arguably the greatest challenge to implementing MDP theory in practice is "the curse of dimensionality." As the complexity of a problem grows, the amount of information that needs to be stored in the state space quickly reaches a point where the MDP is no longer computationally tractable. There now exist several methods for dealing with this problem, all of which are grouped under the title of approximate dynamic programming or neuro-dynamic programming. These potential methods begin by restricting the value function to a certain class of functions and then seeking to find the optimal value function within this class. A typical approximation scheme is based on the linear architecture:

$$v^*(s) \approx \tilde{v}(s, r) = \sum_{i=1}^{k} r_i \phi_i(s),$$

where $\phi_i(s), i = 1, \ldots, k$ are pre-defined basis functions that attempt to characterize the state space and r is a set of weights applied to the basis functions. This reduces the problem from one with $|S|$-dimensions to one with $|k|$-dimensions. The questions are (1) how do you determine what class of functions (determined by ϕ) to choose and (2) how to find the best approximate value function within the chosen class (i.e., the

best values for r)? The first question is still very much wide open.

Answers to the second question fall into two main camps. On the one hand, there are a number of methods that seek to iteratively improve the approximation through the simulation of sample paths of the decision process. The second method uses linear programming but restricts the value function to the approximate form. This reduces the number of variables in the primal to a reasonable number (equal to the number of basis functions chosen). One can then determine the optimal set of weights, r, through column generation. One of the major challenges facing approximate dynamic programming is that it is difficult to determine how close the approximate value function is to its true value. In other words, how much more reward might have been accumulated had the original MDP been solved directly? Though there are some attempts in the literature to answer this question, it remains a significant challenge.

Cross References

►Markov Decision Processes
►Partially Observable Markov Decision Processes

Recommended Reading

Bertsekas, D. (2000). *Dynamic programming and optimal control*. Belmont: Athena Scientific.
Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Belmont: Athena Scientific.
Feinberg, E., & Shwartz, A. (2002). *Handbook of Markov decision processes*. Boston, MA: Kluwer Academic Publishers.
Puterman, M. (1994). *Markov decision processes*. New York: Wiley.
Sutton, R., & Barto, A. (1998). *Reinforcement learning*. Cambridge, MA: MIT Press.

Dynamic Programming For Relational Domains

►Symbolic Dynamic Programming

Dynamic Systems

The dynamic systems approach emphasizes the human, and animal, interaction with the environment. Interactions are described by partial differential equations. Attractors and limit cycles represent stable states which may be analogous to attribute-values.

E

EBL

►Explanation-Based Learning

Echo State Network

►Reservoir Computing

ECOC

►Error Correcting Output Codes

Edge Prediction

►Link Prediction

Efficient Exploration in Reinforcement Learning

John Langford

Synonyms

PAC-MDP learning

Definition

An agent acting in a world makes observations, takes actions, and receives rewards for the actions taken. Given a history of such interactions, the agent must make the next choice of action so as to maximize the long-term sum of rewards. To do this well, an agent may take suboptimal actions which allow it to gather the information necessary to later take optimal or near-optimal actions with respect to maximizing the long-term sum of rewards. These information gathering actions are generally considered exploration actions.

Motivation

Since gathering information about the world generally involves taking suboptimal actions compared with a later learned policy, minimizing the number of information gathering actions helps optimize the standard goal in reinforcement learning. In addition, understanding exploration well is key to understanding reinforcement learning well, since exploration is a key aspect of reinforcement learning which is missing from standard supervised learning settings (Fig. 1).

Efficient Exploration in Markov Decision Processes

One simplification of reinforcement learning is the ►Markov decision process setting. In this setting, an agent repeatedly takes an action a, resulting in a transition to a state according to a conditional probability transition matrix $P(s'|s,a)$, and a (possibly probabilistic) reward $R(s',a,s) \in [0,1]$. The goal is to efficiently output a policy π which is ϵ-optimal over T timesteps. The value of policy π in a start state s is defined as

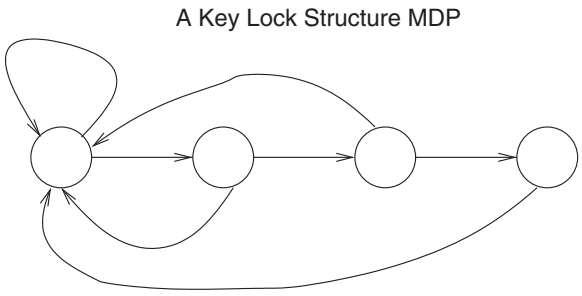
$$\eta(\pi,s) = E_{(a,s,r)^T \sim (\pi,P,R)^T} \sum_{t=1}^{T} r_t,$$

which should be read as the expectation over T-length sequences drawn from the interaction of the policy π with the world as represented by P and R. An ϵ-optimal policy π therefore satisfies:

$$\max_{\pi'} \eta(\pi',s) - \eta(\pi,s) \leq \epsilon.$$

There are several notable results in this setting, typically expressed in terms of the dependence on the number of actions A, and the number of states S. The first is for the β-greedy strategy commonly applied when using ►Q-learning (Watkins & Dayan, 1992) which explores randomly with probability β.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,



Efficient Exploration in Reinforcement Learning. Figure 1. An example of a keylock MDP. The state are arranged in a chain. In each state, one of the two actions leads to the next state while the other leads back to the beginning. The only reward is in the transition to the last state in the chain. Keylock MDPs defeat simple greedy strategies, because the probability of randomly reaching the last transition is exponentially small in the length of the chain

Theorem 1 *There exists MDPs such that with probability at least* $1/2$*,* β*-greedy requires* $\Theta(A^S)$ *explorations to find an* ϵ*-optimal policy.*

This is essentially a negative result, saying that a greedy exploration strategy cannot quickly discover a good policy in some settings. The proof uses an MDP with a key-lock like structure where for each state all actions but one take the agent back to the beginning state, and the reward is at the end of a chain of states.

It turns out that there exists algorithms capable of finding a near-optimal policy in an MDP with only a polynomial number of exploratory transitions.

Theorem 2 *For all MDPs, for any* $\delta > 0$*, with probability* $1 - \delta$*, the algorithm Explicit-Explore-or-Exploit finds an* ϵ*-optimal policy after* $\tilde{O}(S^2A)$ *explorations.*

In other words, E^3 (Kearns & Singh, 1998) requires exploration steps at most proportional to the size of the probability table driving the dynamics of the agent's world. The algorithm works in precisely the manner which might be expected: it builds a model of the world based on its observations and solves the model to determine whether to explore or exploit. The basic approach was generalized to stochastic games and reformulated as an "optimistic initialization" style algorithm named R-MAX (Brafman & Tennenholtz, 2002).

It turns out that an even better dependence is possible using the delayed Q-learning (Strehl, Li, Wiewiora, Langford, & Littman, 2006) algorithm.

Theorem 3 *For all MDPs, for any* $\delta > 0$*, with probability* $1 - \delta$*, the algorithm delayed Q-learning finds an* ϵ*-optimal policy after* $\tilde{O}(SA)$ *explorations.*

The delayed Q-learning algorithm requires explorations proportional to the size of the solution policy rather than proportional to the size of world dynamics. At a high level, delayed Q-learning operates by keeping values for exploration and exploitation of observed state-actions, uses these values to decide between exploration and exploitation, and carefully updates these values. Delayed Q-learning does not obsolete E^3, because the (nonvisible) dependence on ϵ and T are worse (Strehl, 2007).

This is a best possible result in terms of the dependence on S and A (up to log factors), as the following theorem (Kakade, 2003) states:

Theorem 4 *For all algorithms, there exists an MDP such that with* $\Omega(SA)$ *explorations are required to find an* ϵ *optimal policy with probability at least* $\frac{1}{2}$.

Since even representing a policy requires a lookup table of size SA, this algorithm-independent lower bound is relatively unsurprising.

Variations on MDP Learning

There are several minor variations in the setting and goal definitions which do not qualitatively impact the set of provable results. For example, if rewards are in a bounded range, they can be offset and rescaled to the interval $[0,1]$.

It's also common to use a soft horizon (or discounting) where the policy evaluation is changed to:

$$\eta_\gamma(\pi, s) = E_{(a,s,r)^\infty \sim (\pi,P,R)^\infty} \sum_{t=1}^{\infty} \gamma^t r_t$$

for some value $\gamma < 1$. This setting is not precisely equivalent to the hard horizon, but since

$$sum_{t=(\ln(1/\epsilon)+\ln(1/1-\gamma))/1-\gamma}^{\infty} \gamma^t r_t \leq \epsilon$$

similar results are provable with $1/(1-\gamma)$ taking the role of T and slightly altered algorithms.

One last variation changes the goal. Instead of outputting an ϵ-optimal policy for the next T timesteps, we could have an algorithm to handle both the exploration and exploitation, then retrospectively go back over a trace of experience and mark a subset of the actions as "exploration actions," with a guarantee that the remainder of the actions are according to an ϵ-optimal policy (Kakade, 2003). Again, minor alterations to known algorithms in the above setting appear to work here.

Alternative Settings

There are several known analyzed variants of the basic setting formed by making additional assumptions about the world. This includes Factored MDPs (Kearns & Koller, 1999), Metric MDPs (Kakade, Kearns, & Langford, 2003), Continuous MDPs (Brunskill, Leffler, Li, Littman, & Roy, 2008), MDPs with a Bayesian prior (Poupart, Vlassis, Hoey, & Regan, 2006), and apprenticeship learning where there is access to a teacher for an MDP (Abbeel & Ng, 2005). The structure of these results are all similar at a high level: with some additional information, it is possible to greatly ease the difficulty of exploration allowing tractable application to much larger problems.

Cross References

►*k* Armed Bandit
►Reinforcement Learning

Recommended Reading

Abbeel, P., & Ng, A. (2005). Exploration and apprenticeship learning in reinforcement learning. In *ICML 2005*, Bonn, Germany.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX – A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research, 3*, 213–231.

Brunskill, E., Leffler, B. R., Li, L., Littman, M. L., & Roy, N. (2008). CORL: A continuous-state offset-dynamics reinforcement learner. In *UAI-08*, Helsinki, Finland, July 2008.

Kakade, S. (2003). Thesis at Gatsby Computational Neuroscience Unit.

Kakade, S., Kearns, M., & Langford, J. (2003). Exploration in metric state spaces. In *ICML 2003*.

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. In *Proceedings of the 16th international joint conference on artificial intelligence* (pp. 740–747). San Francisco: Morgan Kaufmann.

Kearns, M., & Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. In *ICML 1998* (pp. 260–268). San Francisco: Morgan Kaufmann.

Poupart, P., Vlassis, N., Hoey, J., & Regan, K. (2006). An analytic solution to discrete Bayesian reinforcement learning. In *ICML 2006* (pp. 697–704). New York: ACM Press.

Strehl, A. (2007). Thesis at Rutgers University.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., & Littman, M. L. (2006). PAC model-free reinforcement learning. In *Proceedings of the 23rd international conference on machine learning (ICML 2006)* (pp. 881–888).

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning Journal, 8*, 279–292.

EFSC

►Evolutionary Feature Selection and Construction

Elman Network

►Simple Recurrent Network

EM Algorithm

►Expectation–Maximization Algorithm

EM Clustering

►Expectation Maximization Clustering

Embodied Evolutionary Learning

►Evolutionary Robotics

Emerging Patterns

Definition

Emerging pattern mining is an area of ▶supervised descriptive rule induction. Emerging patterns are defined as itemsets whose support increases significantly from one data set to another (Dong & Li, 1999). Emerging patterns are said to capture emerging trends in time-stamped databases, or to capture differentiating characteristics between classes of data.

Recommended Reading

Dong, G., & Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99)* (pp. 43–52).

Empirical Risk Minimization

Xinhua Zhang
Australian National University
NICTA London Circuit
Canberra, Australia

Definition

The goal of learning is usually to find a model which delivers good generalization performance over an underlying distribution of the data. Consider an input space $\mathcal{X}$ and output space $\mathcal{Y}$. Assume the pairs $(X \times Y) \in \mathcal{X} \times \mathcal{Y}$ are random variables whose (unknown) joint distribution is P_{XY}. It is our goal to find a predictor $f : \mathcal{X} \mapsto \mathcal{Y}$ which minimizes the expected risk:

$$P(f(X) \neq Y) = \mathbb{E}_{(X,Y)\sim P_{XY}}\left[\delta(f(X) \neq Y)\right],$$

where $\delta(z) = 1$ if z is true, and 0 otherwise.

However, in practice we only have n pairs of training examples (X_i, Y_i) drawn identically and independently from P_{XY}. Since P_{XY} is unknown, we often use the risk on the training set (called empirical risk) as a surrogate of the expected risk on the underlying distribution:

$$\frac{1}{n}\sum_{i=1}^{n}\delta(f(X_i) \neq Y_i).$$

Empirical risk minimization (ERM) refers to the idea of choosing a function f by minimizing the empirical risk. Although it is often effective and efficient, ERM is subject to ▶overfitting, i.e., finding a model which fits the training data well but predicts poorly on unseen data. Therefore, ▶regularization is often required.

More details about ERM can be found in Vapnik (1998).

Recommended Reading

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.

Ensemble Learning

Gavin Brown
The University of Manchester
Manchester, UK

Synonyms

Committee machines; Multiple classifier systems

Definition

Ensemble learning refers to the procedures employed to train multiple learning machines and combine their outputs, treating them as a "committee" of decision makers. The principle is that the decision of the committee, with individual predictions combined appropriately, should have better overall ▶accuracy, on average, than any individual committee member. Numerous empirical and theoretical studies have demonstrated that ensemble ▶models very often attain higher accuracy than single models.

The members of the ensemble might be predicting real-valued numbers, class labels, posterior probabilities, rankings, clusterings, or any other quantity. Therefore, their decisions can be combined by many methods, including averaging, voting, and probabilistic methods. The majority of ensemble learning methods are generic, applicable across broad classes of model types and learning tasks.

Motivation and Background

If we could build the "perfect" machine learning device, one which would give us the best possible answer every time, there would be no need for *ensemble learning* methods – indeed, there would be no need for this encyclopedia either. The underlying principle of ensemble learning is a recognition that in real-world situations, every model has limitations and will make errors. Given that each model has these "limitations," the aim of ensemble learning is to manage their strengths and weaknesses, leading to the best possible decision being taken overall. Several theoretical and empirical results have shown that the accuracy of an ensemble can significantly exceed that of a single model.

The principle of combining predictions has been of interest to several fields over many years. Over 200 years ago, a controversial question had arisen, on how best to estimate the mean of a probability distribution given a small number of sample observations. Laplace (1818) demonstrated that the sample mean was not always optimal: under a simple condition, the sample median was a better combined predictor of the population mean. The financial forecasting community has analyzed model combination for several decades, in the context of stock portfolios. The contribution of the machine learning (ML) community emerged in the 1990s – automatic construction (from data) of both the models and the method to combine them. While the majority of the ML literature on this topic is from 1990 onward, the principle has been explored briefly by several independent authors since the 1960s. See Kuncheva (2004b) for historical accounts.

The study of ensemble methods, with model outputs considered for their abstract properties rather than the specifics of the algorithm which produced them, allows for a wide impact across many fields of study. If we can understand precisely why, when, and how particular ensemble methods can be applied successfully, we would have made progress toward a powerful new tool for Machine Learning: *the ability to automatically exploit the strengths and weaknesses of different learning systems*.

Methods and Algorithms

An ensemble consists of a set of models and a method to combine them. We begin this section by assuming that we have a set of models, generated by any of the learning algorithms in this encyclopedia; we explore popular methods of combining their outputs, for classification and regression problems. Following this, we review some of the most popular ensemble algorithms, for *learning* a set of models given the knowledge that they will be combined, including extensive pointers for further reading. Finally, we take a theoretical perspective, and review the concept of ensemble *diversity*, the fundamental property which governs how well an ensemble can perform.

Methods for Combining a Set of Models

There exist numerous methods for model combination, far too many to fully detail here. The *linear* combiner, the *product* combiner, and the *voting* combiner are by far the most commonly used in practice. Though a combiner could be specifically chosen to optimize performance in a particular application, these three rules have shown consistently good behavior across many problems, and are simple enough that they are amenable to theoretical analysis.

The linear combiner is used for models that output real-valued numbers, so is applicable for ▶regression ensembles, or for ▶classification ensembles producing class probability estimates. Here, notation for the latter case is only shown. We have a model $f_t(y|\mathbf{x})$, an estimate of the probability of class y given input $\mathbf{x}$. For a set of these, $t = \{1, \ldots, T\}$, the ensemble probability estimate is,

$$\tilde{f}(y|\mathbf{x}) = \sum_{t=1}^{T} w_t f_t(y|\mathbf{x}). \quad (1)$$

If the weights $w_t = 1/T, \forall t$, this is a simple uniform averaging of the probability estimates. The notation clearly allows for the possibility of a nonuniformly weighted average. If the classifiers have different accuracies on the data, a nonuniform combination could *in theory* give a lower error than a uniform combination. However, in practice, the difficulty of estimating the $\mathbf{w}$ parameters without overfitting, and the relatively small gain that is available (see Kuncheva, 2004b, p. 282), have meant that in practice the uniformly weighted average is by far the most commonly used. A notable exception, to be discussed later in this article, is the *mixture of experts* paradigm – in MoE, weights are nonuniform,

but are learnt and dependent on the input value $\mathbf{x}$. An alternative combiner is the *product rule*:

$$\tilde{f}(y|\mathbf{x}) = \frac{1}{Z}\prod_{t=1}^{T} f_t(y|\mathbf{x})^{w_t}, \tag{2}$$

where Z is a normalization factor to ensure $\tilde{f}$ is a valid distribution. Note that Z is not *required* to make a valid decision, as the order of posterior estimates remain unchanged before/after normalization. Under the assumption that the class-conditional probability estimates are independent, this is the theoretically optimal combination strategy. However, this assumption is highly unlikely to hold in practice, and again the weights $\mathbf{w}$ are difficult to reliably determine. Interestingly, the linear and product combiners are in fact special cases of the *generalized mean* (Kuncheva, 2004b) allowing for a continuum of possible combining strategies.

The linear and product combiners are applicable when our models output real-valued numbers. When the models instead output class labels, a majority (or plurality) vote can be used. Here, each classifier votes for a particular class, and the class with the most votes is chosen as the ensemble output. For a two-class problem the models produce labels, $h_t(\mathbf{x}) \in \{-1, +1\}$. In this case, the ensemble output for the *voting* combiner can be written as

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} w_t h_t(\mathbf{x})\right). \tag{3}$$

The weights $\mathbf{w}$ can be uniform for a simple majority vote, or nonuniform for a weighted vote.

We have discussed only a small fraction of the possible combiner rules. Numerous other rules exist, including methods for combining rankings of classes, and unsupervised methods to combine clustering results. For details of the wider literature, see Kuncheva (2004b) or Polikar (2006).

Algorithms for Learning a Set of Models

If we had a committee of *people* taking decisions, it is self-evident that we would not want them all to make the same bad judgments *at the same time*. With a committee of learning models, the same intuition applies: we will have no gain from combining a set of identical models. We wish the models to exhibit a certain element of "diversity" in their group behavior, though still retaining good performance individually.

We therefore make a distinction between two types of ensemble learning algorithms, those which encourage diversity *implicitly*, and those which encourage it *explicitly*. The vast majority of ensemble methods are *implicit*, in that they provide different *random subsets* of the training data to each learner. Diversity is encouraged "implicitly" by *random* sampling of the data space: at no point is a *measurement* taken to ensure diversity will emerge. The random differences between the datasets might be in the selection of examples (the ▶Bagging algorithm), the selection of features (▶Random Subspace Method, Ho, 1998 or ▶Rotation Forests, Rodriguez, Kuncheva, & Alonso, 2006), or combinations of the two (the Random Forests algorithm, Breiman, 2001). Many other "randomization" schemes are of course possible.

An alternative is to *explicitly* encourage diversity, constructing each ensemble member with some measurement ensuring that it is substantially different from the other members. ▶Boosting algorithms achieve this by altering the distribution of training examples for each learner such that it is encouraged to make more accurate predictions where previous predictors have made errors. The DECORATE algorithm (Melville & Mooney, 2005) explicitly alters the distribution of class labels, such that successive models are forced to learn different answers to the same problem. ▶Negative correlation learning (see Brown, 2004; Brown, Wyatt, Harris, & Yao, 2005), includes a penalty term when learning each ensemble member, explicitly *managing* the accuracy-diversity trade-off.

In general, ensemble methods constitute a large class of algorithms – some based on heuristics, and some on sound learning-theoretic principles. The three algorithms that have received the most attention in the literature are reviewed here. It should be noted that we present only the most basic form of each; numerous modifications have been proposed for a variety of learning scenarios. As further study the reader is referred to the many comprehensive surveys of the field (Brown et al., 2005; Kuncheva, 2004b; Polikar, 2006).

Bagging

In the Bagging algorithm (Breiman, 1996), each member of the ensemble is constructed from a different training dataset, and the predictions combined either

by uniform averaging or voting over class labels. Each dataset is generated by sampling from the total N data examples, choosing N items uniformly at random *with replacement*. Each sample is known as a *bootstrap*; the name Bagging is an acronym derived from *B*ootstrap *AGG*regat*ING*. Since a bootstrap samples N items uniformly at random with replacement, the probability of any individual data item *not* being selected is $p = (1 - 1/N)^N$. Therefore with large N, a single bootstrap is expected to contain approximately 63.2% of the original set, while 36.8% of the originals are not selected.

Like many ensemble methods, Bagging works best with *unstable* models, that is those that produce differing generalization behavior with small changes to the training data. These are also known as *high variance* models, examples of which are ►decision trees and ►neural networks. Bagging therefore tends not to work well with very simple models. In effect, Bagging samples randomly from the space of possible models to make up the ensemble – with very simple models the sampling produces almost identical (low diversity) predictions.

Despite its apparent capability for variance reduction, situations have been demonstrated where Bagging can converge *without* affecting variance (see Brown et al., 2005). Several other explanations have been proposed for Bagging's success, including links to Bayesian model averaging. In summary, it seems that several years from its introduction, despite its apparent simplicity, Bagging is still not fully understood.

Algorithm 1 Bagging

Input: Required ensemble size T
Input: Training set $S=\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
for $t = 1$ to T **do**
  Build a dataset S_t, by sampling N items, randomly *with replacement* from S.
  Train a model h_t using S_t, and add it to the ensemble.
end for
For a new testing point (x', y'),
If model outputs are continuous, combine them by averaging.
If model outputs are class labels, combine them by voting.

Adaboost

Adaboost (Freund & Schapire, 1996) is the most well known of the *Boosting* family of algorithms (Schapire, 2003). The algorithm trains models sequentially, with a new model trained at each round. At the end of each round, mis-classified examples are identified and have their emphasis increased in a new training set which is then fed back into the start of the next round, and a new model is trained. The idea is that subsequent models should be able to compensate for errors made by earlier models.

Adaboost occupies somewhat of a special place in the history of ensemble methods. Though the procedure seems heuristic, the algorithm is in fact grounded in a rich learning-theoretic body of literature. Schapire (1990) addressed a question posed by Kearns and Valiant (1988) on the nature of two complexity classes of learning problems. The two classes are *strongly learnable* and *weakly learnable* problems. Schapire showed that these classes were equivalent; this had the corollary that a weak model, performing only slightly better than random guessing, could be "boosted" into an arbitrarily accurate *strong* model. The original Boosting algorithm was a proof by construction of this equivalence, though had a number of impractical assumptions built-in. The Adaboost algorithm (Freund & Schapire, 1996) was the first practical Boosting method. The authoritative historical account of the development can be found in Schapire (1999), including discussion of numerous variants and interpretations of the algorithm. The procedure is shown in Algorithm 2. Some similarities with Bagging are evident; a key differences is that at each round t, Bagging has a uniform distribution D_t, while Adaboost adapts a nonuniform distribution.

The ensemble is constructed by iteratively adding models. Each time a model is learnt, it is checked to ensure it has at least $\epsilon_t < 0.5$, that is, it has performance *better than random guessing* on the data it was supplied with. If it does not, either an alternative model is constructed, or the loop is terminated.

After each round, the distribution D_t is updated to emphasize incorrectly classified examples. The update causes half the distribution mass of D_{t+1} to be over the examples incorrectly classified by the previous model. More precisely, $\sum_{h_t(\mathbf{x}_i) \neq y_i} D_{t+1}(i) = 0.5$. Thus, if h_t has an error rate of 10%, then examples from that small 10% will be allocated 50% of the next model's training "effort,"

Algorithm 2 Adaboost

Input: Required ensemble size T
Input: Training set $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where $y_i \in \{-1, +1\}$
Define a uniform distribution $D_1(i)$ over elements of S.
for $t = 1$ to T **do**
  Train a model h_t using distribution D_t.
  Calculate $\epsilon_t = P_{D_t}(h_t(x) \neq y)$
  If $\epsilon_t \geq 0.5$ break
  Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
  where Z_t is a normalization factor so that D_{t+1} is a valid distribution.
end for
For a new testing point (x', y'),
$H(x') = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x'))$

while the remaining examples (those correctly classified) are underemphasized. An equivalent (and simpler) writing of the distribution update scheme is to multiply $D_t(i)$ by $1/2(1-\epsilon_t)$ if $h_t(x_i)$ is correct, and by $1/2\epsilon_t$ otherwise.

The updates cause the models to sequentially minimize an exponential bound on the error rate. The training error rate on a data sample $\mathcal{S}$ drawn from the true distribution $\mathcal{D}$ obeys the bound,

$$P_{\mathbf{x},y\sim S}(yH(\mathbf{x}) < 0) \leq \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}. \qquad (4)$$

This *upper bound* on the training error (though not the *actual* training error) is guaranteed to decrease monotonically with T, given $\epsilon_t < 0.5$.

In an attempt to further explain the performance of Boosting algorithms, Schapire also developed bounds on the *generalization* error of voting systems, in terms of the voting margin, the definition of which was given in (10). Note that, this is not the same as the *geometric margin*, optimized by ▶support vector machines. The difference is that the voting margin is defined using the one-norm $\|\mathbf{w}\|_1$ in the denominator, while the geometric margin uses the *two*-norm $\|\mathbf{w}\|_2$. While this is a subtle difference, it is an important one, forming links between SVMs and Boosting algorithms – see Rätsch, Mika, Schölkopf, and Müller (2002) for details. The following bound holds with probability $1-\delta$,

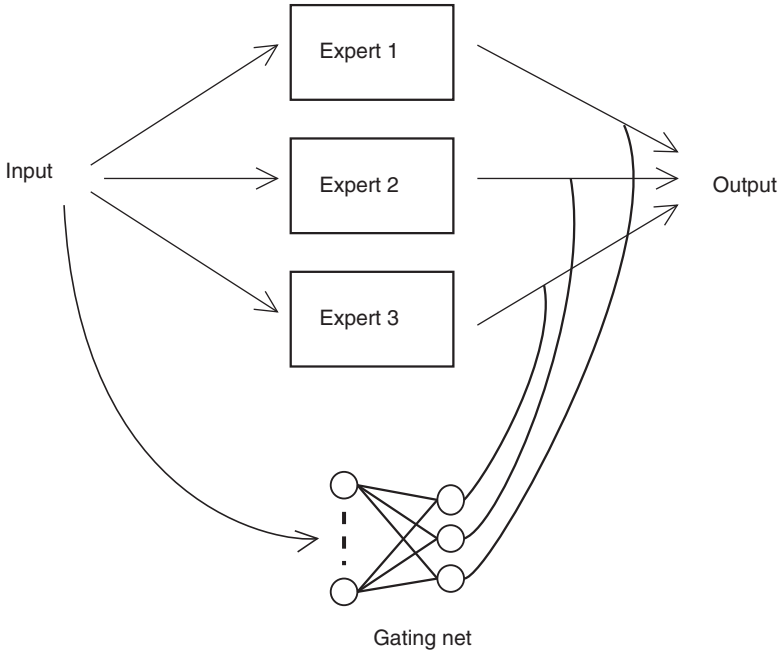
$$P_{\mathbf{x},y\sim\mathcal{D}}(H(\mathbf{x}) \neq y) \leq P_{\mathbf{x},y\sim\mathcal{S}}(yH(\mathbf{x}) < \theta) + \tilde{O}\left(\sqrt{\frac{d}{N\theta^2} - \ln\delta}\right), \qquad (5)$$

where the $\tilde{O}$ notation hides constants and logarithmic terms, and d is the ▶VC-dimension of the model used. Roughly, this states that the generalization error is less than or equal to the training error plus a term dependent on the voting margin. The larger the minimum margin in the training data, the lower the testing error. The original bounds have since been significantly improved, see Koltchinskii and Panchenko (2005) as a comprehensive recent work. We note that this bound holds generally for *any* voting system, and is not specific to the Boosting framework.

The margin-based theory is only one explanation of the success of Boosting algorithms. Mease and Wyner (2008) present a discussion of several questions on why and how Adaboost succeeds. The subsequent 70 pages of discussion demonstrate that the story is by no means simple. The conclusion is, while no single theory can fully explain Boosting, each provides a different part of the still unfolding story.

Mixtures of Experts

The mixtures of experts architecture is a widely investigated paradigm for creating a combination of models (Jacobs, Jordan, Nowlan, & Hinton, 1991). The principle underlying the architecture is that certain models will be able to "specialize" to particular parts of the input space. It is commonly implemented with a neural network as the base model, or some other model capable of estimating probabilities. A *Gating network* receives the same inputs as the component models, but its outputs are used as the weights for a linear combiner. The Gating network is responsible for learning the appropriate weighted combination of the specialized models ("experts") for any given input. Thus, the input space is "carved-up" between the experts, increasing and decreasing their weights for particular examples. In effect, a mixture of experts explicitly learns how to create expert ensemble members in different portions of the input space, and select the most appropriate subset for a new testing example (Fig. 1).



Ensemble Learning. Figure 1. The mixtures of experts architecture

The architecture has received wide attention, and has a strong following in the probabilistic modeling community, where it may go under the pseudonym of a "mixture model." A common training method is the ►expectation-maximization algorithm.

Theoretical Perspectives: Ensemble Diversity

We have seen that all ensemble algorithms in some way attempt to encourage "diversity." In this section, we take a more formalized perspective, to understand what is meant by this term.

What is Diversity?

The optimal "diversity" is fundamentally a *credit assignment* problem. If the committee as a whole makes an erroneous prediction, how much of this error should be attributed to each member? More precisely, how much of the committee prediction is due to the accuracies of the individual models, and how much is due to their interactions when they were combined? We would ideally like to reexpress the ensemble error as two distinct components: a term for the accuracies of the individual models, plus a term for their interactions, i.e., their *diversity*.

It turns out that this so-called *accuracy-diversity* breakdown of the ensemble error is not always possible, depending on the type of error function, and choice of combiner rule. It should be noted that when "diversity" is referred to in the literature, it is most often meant to indicate classification with a majority vote combiner, but for completeness we address the general case here. In the following sections, the existing work to understand diversity in three distinct cases is described: for regression tasks (a linear combiner), and classification tasks, with either a linear combiner or a voting combiner.

Regression Error with a Linear Combination Rule

In a regression problem, it is common to use the squared error criterion. The accuracy-diversity breakdown for this case (using a linear combiner) is called the *ambiguity decomposition* (Krogh & Vedelsby, 1995). The result states that the squared error of the linearly combined ensemble, $\bar{f}(\mathbf{x})$, can be broken into a sum of two components:

$$(\bar{f}(\mathbf{x}) - d)^2 = \frac{1}{T}\sum_{t=1}^{T}(f_t(\mathbf{x}) - d)^2 - \frac{1}{T}\sum_{t=1}^{T}(f_t(\mathbf{x}) - \bar{f}(\mathbf{x}))^2. \quad (6)$$

The first term on the right hand side is the average squared error of the individual models, while the second term quantifies the interactions *between* the predictions. Note that this second term, the "ambiguity," is always positive. This guarantees that, for an arbitrary data point, the ensemble squared error is always less than or equal to the average of the individual squared errors.

The intuition here can be understood as follows. Imagine five friends, playing "guess the weight of the cake" (an old English fairground game): if a player's guess is close enough to the true weight, they win the cake. Just as they are about to play, the fairground manager states that they can only submit *one* guess. The dilemma seems to be in whose guess they should submit – however, the ambiguity decomposition shows us that taking the average of their guesses, and submitting that, will *always* be closer (on average) than choosing a person at random and submitting their guess. Note that this is qualified with "on average" – it may well be that one of the predictions will in fact be closer than the average prediction, but we presume that we have no way of identifying *which* prediction to choose, other than random. It can be seen that greater diversity in the predictions (i.e., a larger ambiguity term) results in a larger gain over the average individual performance. However, it is also clear that there is a trade-off to be had: too much diversity and the average error is extremely large.

The idea of a trade-off between these two terms is reminiscent of the ►bias-variance decomposition (Geman, Bienenstock, & Doursat, 1992); in fact, there is a deep connection between these results. Taking the expected value of (6) over all possible training sets gives us the ensemble analogy to the bias-variance decomposition, called the ►bias-variance-covariance decomposition (Ueda & Nakano, 1996). This shows that the expected squared error of an ensemble $\bar{f}(\mathbf{x})$ from a target d is:

$$\mathcal{E}_{\mathcal{D}}\{(\bar{f}(\mathbf{x})-d)^2\} = \overline{\text{bias}}^2 + \frac{1}{T}\overline{\text{var}} + \left(1-\frac{1}{T}\right)\overline{\text{covar}}, \quad (7)$$

where the expectation is with respect to all possible training datasets $\mathcal{D}$. While the bias and variance terms are constrained to be positive, the covariance between models can become negative – thus the definition of diversity emerges as an extra degree of freedom in the bias-variance dilemma. This extra degree of freedom allows an ensemble to approximate functions that are difficult (if not impossible) to find with a single model. See Brown et al. (2005) for extensive further discussion of this concept.

Classification Error with a Linear Combination Rule

In a classification problem, our error criterion is the misclassification rate, also known as the *zero-one* loss function. For this type of loss, it is well known there is no unique definition of bias-variance; instead there exist multiple decompositions each with advantages and disadvantages (see Kuncheva, 2004b, p. 224). This gives us a clue as to the situation with an ensemble – there is also no simple accuracy-diversity separation of the ensemble classification error. Classification problems can of course be addressed either by a model producing class probabilities (where we linearly combine), or directly producing class labels (where we use majority vote). Partial theory has been developed for each case.

For linear combiners, there exist theoretical results that relate the correlation of the probability estimates to the ensemble classification error. Tumer and Ghosh (1996) showed that the reducible classification error (i.e., above the Bayes rate) of a simple averaging ensemble, e_{ave}, can be written as

$$e_{\text{ave}} = e_{\text{add}}\left(\frac{1+\delta(T-1)}{T}\right), \quad (8)$$

where e_{add} is the classification error of an individual model. The δ is a correlation coefficient between the model outputs. When the individual models are identical, the correlation is $\delta = 1$. In this case, the ensemble error is equal to the individual error, $e_{\text{ave}} = e_{\text{add}}$. When the models are statistically independent, $\delta = 0$, and the ensemble error is a fraction $1/T$ of the individual error, $e_{\text{ave}} = 1/T \times e_{\text{add}}$. When δ is negative, the models are negatively correlated, and the ensemble error is lower than the average individual error. However, (8)

is derived under quite strict assumptions, holding only for a local area around the decision boundary, and ultimately resting on the bias-variance-covariance theory from regression problems. Further details, including recent work to lift some of the assumptions (Kuncheva, 2004b).

Classification Error with a Voting Combination Rule

The case of a classification problem with a majority vote combiner is the most challenging of all. In general, there is no known breakdown of the ensemble classification error into neat accuracy and diversity components. The simplest intuition to show that correlation between models does affect performance is given by the Binomial theorem. If we have T models each with identical error probability $p = P(h_t(\mathbf{x}) \neq y)$, assuming they make statistically *independent* errors, the following error probability of the majority voting committee holds,

$$P(H(\mathbf{x}) \neq y) = \sum_{k>(T/2)}^{T} \binom{T}{k} p^k (1-p)^{(T-k)}. \quad (9)$$

For example, in the case of $T = 21$ ensemble members, each with error $p = 0.3$, the majority voting error will be 0.026, an order of magnitude improvement over the individual error. However, this *only* holds for statistically independent errors. The correlated case is an open problem. Instead, various authors have proposed their own heuristic definitions of diversity in majority voting ensembles. Kuncheva (2004b) conducted extensive studies of several suggested diversity measures; the conclusion was that "*no measure consistently correlates well with the majority vote accuracy.*" In spite of this, some were found useful as an approximate guide to characterize performance of ensemble methods, though should not be relied upon as the "final word" on diversity. Kuncheva's recommendation in this case is the *Q-statistic* (Kuncheva, 2004b, p. 299), due to its simplicity and ease of computation.

Breiman (2001) took an alternative approach, deriving not a *separation* of error components, but a *bound* on the generalization error of a voting ensemble, expressed in terms of the correlations of the models. To understand this, we must introduce concept of *voting margin*. The voting margin for a two-class problem, with $y \in \{-1, +1\}$, is defined,

$$m = \frac{y_t \sum_{t=1}^{T} w_t h_t(\mathbf{x})}{\sum_{t=1}^{T} |w_t|} = yH(\mathbf{x}). \quad (10)$$

If the margin is positive, the example is correctly classified, if it is negative, the example is incorrectly classified. The expected margin $s = \mathcal{E}_{\mathcal{D}}\{m\}$ measures the extent to which the average number of votes for the correct class exceeds the average vote for any other class, with respect to the data distribution $\mathcal{D}$. The larger the voting margin, the more confidence in the classification. Breiman's bound shows,

$$P_{\mathcal{D}}(H(\mathbf{x}) \neq y) = P_{\mathcal{D}}(yH(\mathbf{x}) < 0) \leq \frac{\bar{\rho}(1-s^2)}{s^2}. \quad (11)$$

Here $\bar{\rho}$ is the average pairwise correlation between the errors of the individual models. Thus, the generalization error is minimized by a small $\bar{\rho}$, and an s as close to 1 as possible. The balance between a high accuracy (large s) and a high diversity (low $\bar{\rho}$) constitutes the tradeoff in this case, although the bound is quite loose.

Summary

In summary, the definition of diversity depends on the problem. In a regression problem, the optimal diversity is the trade-off between the bias, variance and covariance components of the squared error. In a classification problem, with a linear combiner, there exists partial theory to relate the classifier correlations to the ensemble error rate. In a classification problem with a voting combiner, there is no single theoretical framework or definition of diversity. However, the lack of an agreed definition of diversity has not discouraged researchers from trying to achieve it, nor has it stalled the progress of effective algorithms in the field.

Conclusions & Current Directions in the Field

Ensemble methods constitute some of the most robust and accurate learning algorithms of the past decade (Caruana & Niculescu-Mizil, 2006). A multitude of heuristics have been developed for randomizing the ensemble parameters, to generate diverse models. It

is arguable that this line of investigation is nowadays rather oversubscribed, and the more interesting research is now in methods for nonstandard data. ►Cluster ensembles (Strehl & Ghosh, 2003) are ensemble techniques applied to unsupervised learning problems. Problems with *nonstationary* data, also known as *concept drift*, are receiving much recent attention (Kuncheva, 2004a). The most up to date innovations are to be found in the biennial *International Workshop on Multiple Classifier Systems* (Roli et al., 2000).

Recommended Reading

Kuncheva (2004b) is the standard reference in the field, which includes references to many further recommended readings. In addition, Brown et al. (2005) and Polikar (2006) provide extensive literature surveys. Roli et al. (2000) is an international workshop series dedicated to ensemble learning.

Breiman, L. (1996). Bagging predictors. *Machine Learning 24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine Learning 45*(1), 5–32.

Brown, G. (2004). *Diversity in neural network ensembles*. PhD thesis, University of Birmingham.

Brown, G., Wyatt, J. L., Harris, R., & Yao, X. (2005). Diversity creation methods: A survey and categorisation. *Journal of Information Fusion 6*(1), 5–20.

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on machine learning* (pp. 161–168). New York: ACM.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the thirteenth international conference on machine learning (ICML'96)* (pp. 148–156). San Francisco: Morgan Kauffman Publishers.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural computation 4*(1), 1–58.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*(8), 832–844.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation 3*(1), 79–87.

Kearns, M., & Valiant, L. G. (1988). *Learning Boolean formulae or finite automata is as hard as factoring*. Technical report TR-14-88, Harvard University Aiken Computation Laboratory.

Koltchinskii, V., & Panchenko, D. (2005). Complexities of convex combinations and bounding the generalization error in classification. *Annals of Statistics 33*(4), 1455.

Krogh, A., & Vedelsby, J. (1995). Neural network ensembles, cross-validation and active learning. In *Advances in neural information processing systems* (pp. 231–238). Cambridge, MA: MIT Press.

Kuncheva, L. I. (2004a). Classifier ensembles for changing environments. In *International workshop on multiple classifier systems*. Lecture Notes in Computer Science 3007. Berlin: Springer.

Kuncheva, L. I. (2004b). *Combining pattern classifiers: Methods and algorithms*. New York: Wiley.

Laplace, P. S. (1818). *Deuxieme supplement a la theorie analytique des probabilites*. Paris: Gauthier-Villars.

Mease, D., & Wyner, A. (2008). Evidence contrary to the statistical view of Boosting. *Journal of Machine Learning Research 9*, 131–156.

Melville, P., & Mooney, R. J. (2005). Creating diversity in ensembles using artificial data. *Information Fusion 6*(1), 99–111.

Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine, 6*(3), 21–45.

Rätsch, G., Mika, S., Schölkopf, B., & Müller, K. R. (2002). Constructing Boosting algorithms from SVMs: An application to one-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*(9), 1184–1199.

Rodriguez, J., Kuncheva, L., & Alonso, C. (2006). Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*(10), 1619–1630.

Roli, F., Kittler, J., Windridge, D., Oza, N., Polikar, R., Haindl, M., et al. (Eds.). *Proceedings of the international workshop on multiple classifier systems 2000–2009. Lecture notes in computer science*. Berlin: Springer. Available at: http://www.informatik.uni-trier.de/ley/db/conf/mcs/index.html

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning 5*, 197–227.

Schapire, R. E. (1999). A brief introduction to Boosting. In *Proceedings of the 16th international joint conference on artificial intelligence* (pp. 1401–1406). San Francisco, CA: Morgan Kaufmann.

Schapire, R. E. (2003). *The boosting approach to machine learning: An overview*. In D. D. Denison, M. H. Hansen, C. Holmes, B. Mallick, & B. Yu (Eds.), *Nonlinear estimation & classification Lecture notes in statistics* (pp. 149–172). Berlin: Springer.

Strehl, A., & Ghosh, J. (2003). Cluster ensembles – A knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research 3*, 583–617.

Tumer, K., & Ghosh, J. (1996). Error correlation and error reduction in ensemble classifiers. *Connection Science 8*(3–4), 385–403.

Ueda, N., & Nakano, R. (1996). Generalization error of ensemble estimators. In *Proceedings of IEEE international conference on neural networks* (Vol. 1, pp. 90–95). ISBN: 0-7803-3210-5

Entailment

Synonyms

Implication; Logical consequence

Definition

The term entailment is used in the context of logical reasoning. Formally, a logical formula T entails a formula c if and only if all models of T are also a model of c. This is usually denoted as $T \models c$ and means that c is a logical consequence of T or that c is implied by T.

Let us elaborate this definition for propositional clausal logic, where the formulae T could be the following expression:

```
flies :- bird, normal.
bird :- blackbird.
bird :- ostrich.
```

Here, the first clause or rule can be read as flies *if* normal *and* bird, that is, normal birds fly, the second and third one as stating that blackbirds, resp. ostriches, are birds. An interpretation is then an assignment of truth-values to the propositional variables. For instance, for the above domain

```
{ostrich, bird}
{blackbird, bird, normal}
```

are interpretations, specified through the set of propositional variables that are true. This means that in the first interpretation, the only true propositions are `ostrich` and `bird`. An interpretation specifies a kind of possible world. An interpretation I is then a model for a clause $h :- b_1, ..., b_n$ if and only if $\{b_1, ..., b_n\} \subseteq I \rightarrow h \in I$ and it is model for a clausal theory if and only if it is a model for all clauses in the theory. Therefore, the first interpretation above is a model for the theory, but the second one is not because the interpretation is not a model for the first clause (as `{bird, normal}` $\subseteq I$ but `flies` $\notin I$). Using these notions, it can now be verified that the clausal theory T above logically entails the clause

```
flies :- ostrich, normal.
```

because all models of the theory are also a model for this clause.

In machine learning, the notion of entailment is used as a covers relation in ►inductive logic programming, where hypotheses are clausal theories, instances are clauses, and an example is covered by the hypothesis when it is entailed by the hypothesis.

Cross References

►Inverse Entailment
►Learning from Entailment
►Logic of Generality

Recommended Reading

Russell, S., & Norvig, P. *Artificial intelligence: A modern approach* (2nd ed.). Prentice Hall.

Entity Resolution

Indrajit Bhattacharya[1], Lise Getoor[2]
[1]IBM India Research Laboratory, New Delhi, India
[2]University of Maryland, College Park, MD, USA

Synonyms

Co-reference resolution; Deduplication; Duplicate detection; Identity uncertainty; Merge-purge; Object consolidation; Record linkage; Reference reconciliation

Definition

A fundamental problem in data cleaning and integration (see ►Data Preparation) is dealing with uncertain and imprecise references to real-world entities. The goal of entity resolution is a take a collection of uncertain entity references (or references, in short) from a single data source or multiple data sources, discover the unique set of underlying entities, and map each reference to its corresponding entity. This typically involves two subproblems – identification of references with different attributes to the same entity, and disambiguation of references with identical attributes by assigning them to different entities.

Motivation and Background

Entity resolution is a common problem that comes up in different guises (and is given different names) in many computer science domains. Examples include computer vision, where we need to figure out when regions in two different images refer to the same underlying object (the correspondence problem); natural language processing when we would like to determine which noun phrases refer to the same underlying entity (co-reference resolution); and databases, where, when merging two databases or cleaning a database, we would

like to determine when two tuple records are referring to the same real-world object (deduplication and data integration). Deduplication is important for removing redundancy and for accurate analysis. In information integration, determining approximate joins is important for consolidating information from multiple sources; most often there will not be a unique key that can be used to join tables across databases.

Such ambiguities in entity references can occur due to multiple reasons. Often times, data may have data entry errors, such as typographical errors. Multiple representations, such as abbreviations, are also possible. Different databases typically have different keys – one person database may use social security numbers while another uses name and address.

Traditional entity resolution approaches focus on matching attributes of different references for resolving entities. However, many data sources have explicit or implicit relationships present among the entity references. These relations are indicative of relationships between the underlying entities themselves. For example, person records in census data are linked by family relationships such as sibling, parent, and spouse. Researchers collaborate mostly within their organization, or their research community, as a result of which references to related researchers tend to occur closely together. Recent entity resolution approaches in statistical relational learning make use of relationships between references to improve entity resolution accuracy, and additionally to discover relationships between the underlying entities.

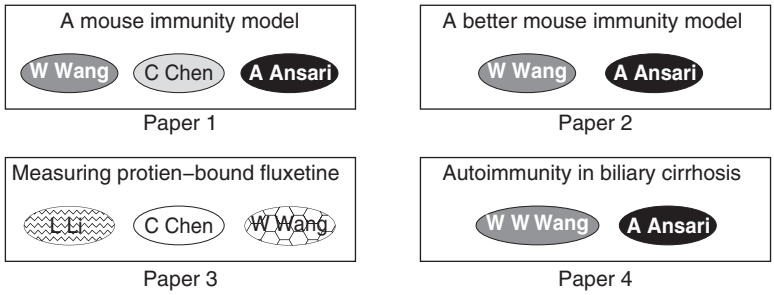
Theory/Solution

As an illustration of the entity resolution problem, consider the task of resolving the author references in a database of academic publications similar to DBLP, CiteSeer or PubMed. Let us take as an example the following set of four papers:

1. W. Wang, C. Chen, A. Ansari, "A mouse immunity model"
2. W. Wang, A. Ansari, "A better mouse immunity model"
3. L. Li, C. Chen, W. Wang, "Measuring protein-bound fluxetine"
4. W. W. Wang, A. Ansari, "Autoimmunity in biliary cirrhosis"

Now imagine that we would like to find out, given these four papers, which of these author names refer to the same author entities. This process involves determining whether paper 1 and paper 2 are written by the same author named Wang, or whether they are different authors. We need to answer similar questions about all such similar author names in the database.

In this example, it turns out there are six underlying author entities, which we will call *Wang*1 and *Wang*2, *Chen*1 and *Chen*2, *Ansari* and *Li*. The three references with the name "A. Ansari" correspond to author *Ansari* and the reference with name "L. Li" to author *Li*. However, the two references with name "C. Chen" map to two different authors *Chen*1 and *Chen*2. Similarly, the four references with name "W. Wang" or "W. W. Wang" map to two different authors. The "Wang" references from the first, second, and fourth papers correspond to author *Wang*1, while that from the third paper maps to a different author *Wang*2. This inference illustrates the twin problems of *identifying* "W. Wang" and "W. W. Wang" as the same author, and *disambiguating* two references with name "W. Wang" as different authors. This is shown pictorially in Fig. 1,



Entity Resolution. Figure 1. The references in different papers in the bibliographic example. References to the same entity are identically shaded

where references that correspond to the same authors are shaded identically. In the entity resolution process, all those and only those author references that are shaded identically should be resolved as corresponding to the same underlying entity.

Formally, in the entity resolution problem, we are given a set of references $\mathcal{R} = \{r_i\}$, where each reference r has attributes $r.A_1, r.A_2, \ldots, r.A_k$, such as observed names and affiliations for author references, as in our example above. The references correspond to some set of unknown entities $\mathcal{E} = \{e_i\}$. We introduce the notation $r.E$ to refer to the entity to which reference r corresponds. The goal is to recover the hidden set of entities $\mathcal{E} = \{e_i\}$ and the entity labels $r.E$ for individual references given the observed attributes of the references. In addition to the attributes, in some data sources we have information in the form of relationships between the references, such as coauthor relationships between author references in publication databases. We can capture the relationships with a set of hyper-edges $\mathcal{H} = \{h_i\}$. Each hyper-edge h may have attributes as well to capture the attributes of relationships, which we denote $h.A_1, h.A_2, \ldots, h.A_l$, and we use $h.R$ to denote the set of references that it connects. In our example, each rectangle denotes one hyper-edge corresponding to one paper in the database. The first hyper-edge corresponding to *Paper*1 has as its attribute the title "A mouse immunity model" and connects the three references having name attributes "W. Wang," "C. Chen," and "A. Ansari." A reference r can belong to zero or more hyper-edges and we use $r.H$ to denote the set of hyper-edges in which r participates. For example, if we have paper, author, and venue references, then a paper reference may be connected to multiple author references and also to a venue reference. In general, the underlying references can refer to entities of different types, as in a publication database, or in newspaper articles, which contain references to people, places, organizations, etc. When the type information is known for each reference, resolution decisions are restricted within references of the same type. Otherwise, the types may need to be discovered as well as part of the entity resolution process.

Traditional entity resolution approaches pose entity resolution as a pair-wise decision problem over references based on their attribute similarity. It can also be posed as a ►graph clustering problem, where references are clustered together based on their attribute similarities and each cluster is taken to represent one underlying entity. Entity resolution approaches differ in how the similarities between references are defined and computed and how the resolution decisions are made based on these similarities. Traditionally, each pair-wise decision is made independently of the others. For example, the decision to resolve the two *Wang* references from papers 1 and 3 would be made independently of the decision to resolve the two *Chen* references from the same papers.

The first improvement is to account for the similarity of the coauthor names when such relationships are available. However, this still does not consider the "entities" of the related references. For the two "Wang" references in the earlier example, the two "C. Chen" coauthors match regardless of whether they refer to *Chen1* or *Chen2*. The correct evidence to use here is that the "Chen's" are not co-referent. In such a setting, in order to resolve the "W. Wang" references, it is necessary to *resolve* the "C Chen" references as well, and not just consider their name similarity. In the collective relational entity resolution approach, resolutions are not made independently, but instead one resolution decision affects other resolutions via hyper-edges.

Below, we discuss the different entity resolution approaches in greater detail.

Attribute-Based Entity Resolution

As discussed earlier, exact matching of attributes does not suffice for entity resolution. Several sophisticated similarity measures have been developed for textual strings (Cohen, Ravikumar, & Fienberg, 2003; Chaudhuri, Ganjam, Ganti, & Motwani, 2003) that may be used for unsupervised entity resolution. Finally, a weighted combination of the similarities over the different attributes for each reference is used to compute the attribute similarity between two references. An alternative is to use adaptive supervised algorithms that learn string ►similarity metrics from labeled data (Bilenko & Mooney, 2003). In the traditional entity resolution approach (Fellegi & Sunter, 1969; Cohen et al., 2003), similarity is computed for each pair of references r_i, r_j based on their attributes and only those pairs that have similarity above some threshold are considered co-referent.

Efficiency

Even the attribute-only approach to entity resolution is known to be a hard problem computationally, since it is infeasible to compare all pairs of references using expensive similarity measures. Therefore, efficiency issues have long been a focus for data cleaning, the goal being the development of inexpensive algorithms for finding approximate solutions. The key mechanisms for doing this involve computing the matches efficiently and employing techniques commonly called "blocking" to quickly find potential duplicates (Hernández & Stolfo, 1995; Monge & Elkan, 1997), using cheap and index-based similarity computations to rule out non-duplicate pairs. Sampling approaches can quickly compute cosine similarity between tuples for fast text-joins within an SQL framework (Gravano, Ipeirotis, Koudas, & Srivastava, 2003). Error-tolerant indexes can also be used in data warehousing applications to efficiently look up a small but "probabilistically safe" set of reference tuples as candidates for matching for an incoming tuple (Chaudhuri et al., 2003). Generic entity resolution frameworks also exist for resolving and merging duplicates as a database operator and minimize the number of record-level and feature-level operations (Menestrina, Benjelloun, & Garcia-Molina, 2006).

Probabilistic Models for Pairwise Resolution

The groundwork for posing entity resolution as a probabilistic ▶classification problem was done by Fellegi and Sunter (1969), who studied the problem of labeling pairs of records from two different files to be merged as "match" (M) or "non-match" (U) on the basis of agreement γ among their different fields or attributes. Given an agreement pattern γ, the conditional probabilities $P(\gamma|M)$ and $P(\gamma|U)$ of γ given matches and non-matches are computed and compared to decide whether the two references are duplicates or not. Fellegi and Sunter showed that the probabilities $P(\gamma|M)$ and $P(\gamma|U)$ of field agreements can be estimated without requiring labeled training data if the different fields agreements are assumed to be independent. Winkler (2002) used the EM algorithm to estimate the probabilities without making the independence assumption.

Probabilistic Models for Relational Entity Resolution

Probabilistic models that take into account interaction between different entity resolution decisions through hyper-edges have been proposed for named-entity recognition in natural language processing and for citation matching (McCallum & Wellner, 2004; Singla & Domingos, 2004). Such ▶relational learning approaches introduce a decision variable y_{ij} for every pair of references r_i and r_j, but instead of inferring the y_{ij}'s independently, use conditional random fields for joint reasoning. For example, the decision variables for the "Wang" references and the "Chen" references in papers 1 and 3 would be connected to each other features functions would be defined to ensure that they are more likely to take up identical values.

Such relational models are supervised and require labeled data to train the parameters. One of the difficulties in using a supervised method for resolution is constructing a good training set that includes a representative collection of positive and negative examples. Accordingly, unsupervised relational models have also been developed (Bhattacharya & Getoor, 2006; Li, Morie, & Roth, 2005; Pasula, Marthi, Milch, Russell, & Shpitser, 2003). Instead of introducing pairwise decision variables, this category of approaches use generative models for references using latent entity labels. Note that, here, the number of entities is unknown and needs to be discovered automatically from the available references. Relationships between the references, such as co-mentions or co-occurrences, are captured using joint distributions over the entity labels.

All of these probabilistic models have been shown to perform well in practice and have the advantage that the match/non-match decisions do not depend on any user-specified similarity measures and thresholds but are learned directly from data. However, this benefit comes at a price. Inference in relational probabilistic models is an expensive process. Exact inference is mostly intractable and approximate strategies such as loopy belief propagation and Monte Carlo sampling strategies are employed. Even these approximate strategies take several iterations to converge and extending such approaches to large datasets is still an open problem.

Other Approaches for Relational Entity Resolution

Alternative approaches (Dong, Halevy, & Madhavan, 2005; Bhattacharya & Getoor, 2007; Kalashnikov, Mehrotra, & Chen, 2005) consider relational structure of the entities for data integration but avoid the complexity of probabilistic inference. By avoiding a formal probabilistic model, these approaches can handle complex and longer-range relationships between different entity references and the resolution process is significantly faster as well. Such approaches also create pairwise decision nodes between references and create a dependency graph over them to capture the relationships in the data. But instead of performing probabilistic inference, they keep updating the value associated with each decision node by propagating relational evidence from one decision node to another over the dependency graph.

When the relationships between the references and the entities can be captured in a single graph, the matching entity for a specific reference may be identified using path-based similarities between their corresponding nodes in the graph. The connection strength associated with each edge in the graph can be determined in the unsupervised fashion given all the references, their candidate entity choices, and the relationships between them, by solving a set of nonlinear equations (Kalashnikov et al., 2005). This approach is useful for incremental data cleaning when the set of entities currently in the database is known and an incoming reference needs to be matched with one of these entities.

An alternative approach to performing collective entity resolution using relational evidence is to perform collective relational clustering (Bhattacharya & Getoor, 2007). The goal here is to cluster the references into entities by taking into account the relationships between the references. This is achieved by defining a similarity measure between two clusters of references that take into account not only the attribute similarity of the references in the two clusters, but also the neighboring clusters of each cluster. The neighboring clusters of any reference cluster c are defined by considering the references r' connected to references r belonging to c via hyper-edges, and the clusters to which these related references belong. If the $r.C$ represents the current cluster for reference c, then $N(c) = \bigcup r'.C$, where $r.H = r'.H$ and $r.C = c$. For instance, the neighboring clusters for a *Wang* cluster in our example containing the *Wang* references from papers 1,2 and 4 are the *Ansari* cluster and the *Chen* clusters containing the other references from the same papers. The relational similarity between two clusters is then computed by comparing their neighborhoods. This relational similarity complements attribute similarity in the combined similarity between two clusters. Intuitively, two entities are likely to be the same if they are similar in attributes and are additionally connected to the same other entities. Collective relational clustering can be efficiently implemented by maintaining a priority queue for merge-able cluster pairs and updating the "neighboring" queue elements with every merge operation.

Applications

Data cleaning and reference disambiguation approaches have been applied and evaluated in a number of domains. The earliest applications were on medical data. Census data is an area where detection of duplicates poses a significant challenge and Winkler (Winkler, 2002) has successfully applied his research and other baselines to this domain. A great deal of work has been done making use of bibliographic data (Pasula et al., 2003; Singla & Domingos, 2004; Bhattacharya & Getoor, 2007). Almost without exception, the focus has been on the matching of citations. Work in coreference resolution and disambiguating entity mentions in natural language processing (McCallum & Wellner, 2004) has been applied to text corpora and newswire articles like the TREC corpus. There have also been significant applications in information integration in data-warehouses (Chaudhuri et al., 2003).

Cross References

- ▶Classification
- ▶Data Preparation
- ▶Graph Clustering
- ▶Similarity Metrics
- ▶Statistical Relational Learning

Recommended Reading

Bhattacharya, I., & Getoor, L. (2006). A latent dirichlet model for unsupervised entity resolution. In *The SIAM international conference on data mining* (SIAM-SDM), Bethesda, MD, USA.

Bhattacharya, I., & Getoor, L. (2007). Collective entity resolution in relational data. *ACM transactions on knowledge discovery from data, 1*(1), 5.

Bilenko, M., & Mooney, R. J. (2003). Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (KDD-2003), Washington, DC.

Chaudhuri, S., Ganjam, K., Ganti, V., & Motwani, R. (2003). Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on management of data* (pp. 313–324). San Diego, CA.

Cohen, W. W., Ravikumar, P., & Fienberg, S. E. (2003). A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-2003 workshop on information integration on the web* (pp. 73–78). Acapulco, Mexico.

Dong, X., Halevy, A., & Madhavan, J. (2005). Reference reconciliation in complex information spaces. In *The ACM international conference on management of data (SIGMOD)*, Baltimore, MD, USA.

Fellegi, I. P., & Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association, 64*, 1183–1210.

Gravano, L., Ipeirotis, P., Koudas, N., & Srivastava, D. (2003). Text joins for data cleansing and integration in an rdbms. In *19th IEEE international conference on data engineering.*

Hernández, M. A., & Stolfo, S. J. (1995). The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD international conference on management of data (SIGMOD-95)* (pp. 127–138). San Jose, CA.

Kalashnikov, D. V., Mehrotra, S., & Chen, Z. (2005). Exploiting relationships for domain-independent data cleaning. In *SIAM international conference on data mining (SIAM SDM)*, April 21–23 2005, Newport Beach, CA, USA.

Li, X., Morie, P., & Roth, D. (2005). Semantic integration in text: From ambiguous names to identifiable entities. *AI Magazine. Special issue on semantic integration, 26*(1).

McCallum, A., & Wellner, B. (2004). Conditional models of identity uncertainty with application to noun coreference. In *NIPS*, Vancouver, BC.

Menestrina, D., Benjelloun, O., & Garcia-Molina, H. (2006). Generic entity resolution with data confidences. In *First Int'l VLDB workshop on clean databases*, Seoul, Korea.

Monge, A. E., & Elkan, C. P. (1997). An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD 1997 workshop on research issues on data mining and knowledge discovery* (pp. 23–29). Tuscon, AZ.

Pasula, H., Marthi, B., Milch, B., Russell, S., & Shpitser, I. (2003). Identity uncertainty and citation matching. In *Advances in neural information processing systems* 15. Cambridge, MA: MIT Press.

Singla, P., & Domingos, P. (2004). Multi-relational record linkage. In *Proceedings of 3rd workshop on multi-relational data mining at ACM SI GKDD*, Seattle, WA.

Winkler, W. E. (2002). Methods for record linkage and Bayesian networks. *Technical Report, Statistical Research Division*, U.S. Census Bureau, Washington, DC.

EP

►Expectation Propagation

Epsilon Covers

Thomas Zeugmann
Hokkaido University
Sapparo, Japan

Definition

Let (M, ρ) be a metric space, let $S \subseteq M$, and let $\varepsilon > 0$. A set $E \subseteq M$ is an *ε-cover* for S, if for every $s \in S$ there is an $e \in E$ such that $\rho(s, e) \leq \varepsilon$.

An *ε-cover* E is said to be *proper*, if $E \subseteq S$.

Application

The notion of an ε-cover is frequently used in kernel-based learning methods.

For further information, we refer the reader to Herbrich (2002).

Cross References

►Statistical Machine Learning
►Support Vector Machines

Recommended Reading

Herbrich, R. (2002). *Learning kernel classifiers: Theory and algorithms*. Cambridge, MA: MIT Press.

Epsilon Nets

Thomas Zeugmann
Hokkaido University
Sapparo, Japan

Definition

Epsilon nets were introduced by Haussler and Welz (1987) and their usefulness for computational learning theory has been discovered by Blumer, Ehrenfeucht, Haussler, & Warmuth (1989).

Let $X \neq \varnothing$ be any learning domain and let $\mathcal{C} \subseteq \wp(X)$ be any nonempty concept class. For the sake of simplicity, we also use $\mathcal{C}$ here as hypothesis space. In order to guarantee that all probabilities considered below do exist, we restrict ourselves to *well-behaved* concept classes (►PAC Learning).

Furthermore, let D be any arbitrarily fixed probability distribution over the learning domain X and let $c \in \mathcal{C}$ be any fixed concept.
A hypothesis $h \in \mathcal{C}$ is said to be *bad* for c iff

$$d(c,h) = \sum_{x \in c \triangle h} D(x) > \varepsilon.$$

Furthermore, we use

$$\Delta(c) =_{df} \{h \triangle c \mid h \in \mathcal{C}\}$$

to denote the set of all possible *error regions* of c with respect to $\mathcal{C}$ and D. Moreover, let

$$\Delta_\varepsilon(c) =_{df} \{h \triangle c \mid h \in \mathcal{C},\ d(c,h) > \varepsilon\}$$

denote the set of all *bad error regions* of c with respect to $\mathcal{C}$ and D.

Now we are ready to formally define the notion of an ε-net.

Definition

Let $\varepsilon \in (0,1)$ and let $S \subseteq X$. The set S is said to be an *ε-net for* $\Delta(c)$ iff $S \cap r \neq \varnothing$ for all $r \in \Delta_\varepsilon(c)$.

Remarks

Conceptually, a set S constitutes an ε-net for $\Delta(c)$ iff every bad error region is hit by at least one point in S.

Example

Consider the one-dimensional Euclidean space $\mathbb{E}$ and let $X = [0,1] \subseteq \mathbb{E}$. Furthermore, let $\mathcal{C}$ be the set of all closed intervals $[a,b] \subseteq [0,1]$. Consider any fixed $c \in \mathcal{C}$ and let D be the uniform distribution, i.e., $D([a,b]) = 1/(b-a)$ for all $[a,b] \in \mathcal{C}$. Furthermore, let $h \in \mathcal{C}$; then we may write $c \triangle h = I_1 \cup I_2$, where $I_1, I_2 \in \mathcal{C}$. Let $\varepsilon \in (0,1)$ be arbitrarily fixed and let

$$S = \{k\varepsilon/2 \mid 0 \leq k \leq \lceil 2/\varepsilon \rceil,\ k \in \mathbb{N}\}.$$

Then, S forms an ε-net for $\Delta(c)$. This can be seen as follows. Assume $r \in \Delta_\varepsilon(c)$. Then, $D(I_1) > \varepsilon/2$ or $D(I_2) > \varepsilon/2$. Now, by the definition of S it is obvious that $D(I_i) > \varepsilon/2$ implies $I_i \cap S \neq \varnothing$, $i = 1, 2$.

Application

Recall that in ▶PAC Learning, the general strategy to design a learner has been to draw a sufficiently large finite sample and then to find a hypothesis that is consistent with it. For showing that this strategy is always successful, the notion of an ε-net plays an important role. This can be expressed by the following observation.

Observation. Let $S = \{x_1, \ldots, x_m\}$ be an ε-net for $\Delta(c)$, and let $h \in \mathcal{C}$ be any hypothesis such that $h(x_i) = c(x_i)$ for all $1 \leq i \leq m$, i.e., h is consistent. Then we have $d(c,h) \leq \varepsilon$.

It then remains to show that the ▶VC Dimension of $\mathcal{C}$ and of $\Delta(c)$ are the same and to apply Sauer's Lemma to complete the proof.

For further information, we refer the reader to Blumer, Ehrenfeucht, Haussler, & Warmuth (1989) as well as to Kearns and Vazirani (1994).

Cross References

▶PAC Learning
▶VC Dimension

Recommended Reading

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, *36*(4), 929–965.
Haussler, D., & Welz, E. (1987). Epsilon nets and simplex range queries. *Discrete & Computational Geometry*, *2*, 127–151.
Kearns, M. J., & Vazirani, U. V. (1994). *An introduction to computational learning theory*. Cambridge, MA: MIT Press.

Equation Discovery

Ljupčo Todorovski
University of Ljubljana
Ljubljana, Slovenia

Synonyms

Computational discovery of quantitative laws; Symbolic regression

Definition

Equation discovery is a machine learning task that deals with the problem of learning quantitative laws and models, expressed in the form of equations, in

collections of measured numeric data. Equation discovery methods take at input a ►data set consisting of measured values of a set of numeric variables of an observed system or phenomenon. At output, equation discovery methods provide a set of equations, such that, when used to calculate the values of system variables, the calculated values closely match the measured ones.

Motivation and Background

Equation discovery methods can be used to solve complex modeling tasks, i.e., establishing a mathematical model of an observed system. Modeling tasks are omnipresent in many scientific and engineering domains.

Equation discovery is strongly related to *system identification*, another approach to mathematical modeling. System identification methods work under the assumption that the structure of the model (the form of the model equations) is known or comes from a well-defined class of model structures, such as polynomials or neural networks. Therefore, they are mainly concerned with the parameter estimation task, that is, the task of determining the values of the model parameters that minimize the discrepancy between measured data and data obtained by simulating the model. Equation discovery methods, on the other hand, aim at identifying both, an adequate structure of the model equations and appropriate values of the model parameters.

►Regression also deals with building predictive models from numeric data. The focus of regression methods is on building descriptive black-box models that can reconstruct the training data with high accuracy. In contrast, equation discovery methods focus on establishing explanatory models that, beside accurate predictions, provide explanations of the mechanisms that govern the behavior of the modeled system.

Early equation discovery methods dealt with rediscovering empirical laws from the history of science (this is where the synonym "computational discovery of quantitative laws" comes from). Through the years, the focus of the equation discovery methods has shifted from discovering quantitative laws to modeling real-world systems.

Structure of the Learning System

The task of equation discovery can be decomposed into two closely coupled subtasks of structural identification and parameter estimation. The first task of structural identification deals with the problem of finding the optimal structure of an equation. The second task of parameter estimation deals with the problem of finding the optimal values of the constant parameters in the equation. General approaches to and specific methods for equation discovery use different techniques to solve these two subtasks.

Approaches and Methods

There are two general and fundamentally different approaches to equation discovery. The first approach relies on a definition of a space of candidate equation structures. Following this definition, a generate-and-test (or ►learning as search) approach is used to generate different equation structures, solve the parameter estimation task for each of them, and report those equations that most closely approximate the data. The second approach relies on heuristics, used by scientists and engineers in the discovery or modeling processes, to establish an appropriate equation structure.

The first equation discovery system, Bacon (Langley, 1981), follows the second approach described above. It incorporates a set of data-driven heuristics for detecting regularities (constancies and trends) in measured data and for formulating hypotheses based on them. An example heuristic would, when faced with a situation where the values of two observed variables increase/decrease simultaneously, introduce a new equation term by multiplying them. Furthermore, Bacon builds equation structure at different levels of description. At each level of description, all but two variables are held constant and hypotheses connecting the two changing variables are considered. Using a relatively small set of data-driven heuristics, Bacon is able to rediscover a number of physical laws including the ideal gas law, the law of gravitation, the law of refraction, and Black's specific heat law.

An alternative set of heuristics for equation discovery can be derived from dimensional analysis that is routinely used to check the plausibility of equations by using rules that specify the proper ways to combine variables and terms with different *measurements units*,

different measurement scales, or types thereof. Following these rules, equation discovery method Coper (Kokar, 1986) considers only equation structures that properly combine variables and constants, given the knowledge about their exact measurement units. Equation discovery method SDS (Takashi & Hiroshi, 1998) extends Coper to cases, where the exact measurement units of the variables and constants involved in the equation are not known, but only knowledge about the types of the ▶measurement scales is available.

Finally, the heuristics and design of the equation discovery method E* (Schaffer, 1993) is based on a systematic survey of more than a hundred laws and models published in the Physical Review journal. The review shows that many of the published laws and models follow one of five different equation structures. By including only these five structures as its main heuristic for solving the structure identification task (implementing it as a ▶language bias), E* was able to reconstruct the correct laws and models in about a third of the test cases collected from the same journal.

Abacus (Falkenhainer & Michalski, 1990) was the first equation discovery method that followed the generate-and-test (or ▶learning as search) approach, mentioned above. Abacus experimented with different search strategies within a fixed space of candidate equation structures. Other methods that follow the generate-and-test approach differ in the ways they define the space of candidate equation structures and solve the parameter estimation task.

Equation discovery methods EF (Zembowitz & Zytkow, 1992) and Lagrange (Džeroski & Todorovski, 1995) explore the space of polynomial equation structures that are linear in the constant parameters, so they apply ▶linear regression to estimate parameters. The user can shape the space of candidate structures by specifying parameters, such as, the maximal polynomial degree, the maximal number of multiplicative terms included in a polynomial, and a set of functions that can be used to transform the original variables before combining them into multiplicative terms.

While all of the above methods assume a fixed predefined ▶language bias (via specification of the class of candidate equation structures or via heuristics for establishing appropriate structure), equation discovery method Lagramge (Todorovski & Džeroski, 1997) employs dynamic declarative ▶language bias, that is, let the user of the equation discovery method choose or specify the space of candidate equation structures. In its first version, Lagramge uses the formalism of context-free grammars for specifying the space of equation structures. The formalism has been shown to be general enough to allow users to build their specification upon many different types of modeling knowledge, from measurement units to very specific knowledge about building models in a particular domain of interest (Todorovski & Džeroski, 2007). For solving the structure identification task, Lagramge defines a refinement operator that orders the search space of candidate equation structures, defined by the user-specified grammar, from the simplest ones to more complex. Exhaustive and ▶beam search strategies are then being employed to the search space and for each structure considered during the search, Lagramge uses gradient-descent methods for nonlinear optimization to solve the parameter estimation task. The heuristic function that guides the search is based on the ▶mean squared error that measures the discrepancy between the measured and simulated values of the observed system variables. Alternatively, Lagramge can use heuristic function that takes into account the complexity of the equation and is based on the ▶minimum description length principle.

Successors of Lagramge, equation discovery methods, Lagramge 2 (Todorovski & Džeroski, 2007), IPM (Bridewell, Langley, Todorovski, & Džeroski, 2008), and HIPM (Todorovski, Bridewell, Shiran, & Langley, 2005), primarily focus on the improvement of the knowledge representation formalism used to formalize the modeling knowledge and transform it to ▶language bias for equation discovery. All of them follow the paradigm of ▶inductive process modeling.

Types of Equations

At first, equation discovery methods dealt with the problem of learning algebraic equations from data. Equation discovery method Lagrange (Džeroski & Todorovski, 1995) extended the scope of equation discovery to modeling dynamics from ▶time series data with ordinary differential equations. It took a naïve approach based on transforming the task of discovering ordinary differential equations to the simpler task of discovering algebraic equations, by extending the set of observed system variables with numerically calculated time derivatives thereof. By doing so, any of the

existing equation discovery methods could be, in principle, used to discover differential equations. However, the naïve approach has a major drawback of introducing large numerical errors, due to instability of methods for numerical differentiation. Equation discovery method GoldHorn (Križman, Džeroski, & Kompare, 1995) replaced the instable numerical differentiation with the stable numerical methods for the inverse problem of integration. Goldhorn also upgrades Lagrange with filtering methods to cope with measurement errors and noisy data.

While ordinary differential equations can model systems that change their state along a single dimension, time, partial differential equations can be used to model systems that change along many (temporal and spatial) dimensions. The naïve approach of introducing numerically calculated partial derivatives has been used in the Paddles (Todorovski, Džeroski, Srinivasan, Whiteley, & Gavaghan, 2000) method for discovery of partial differential equations. The method first slices the measurement data into narrow spatial subsets, induces ordinary differential equations in each of them, and uses most frequently obtained equation structures to extend them with partial derivatives and to obtain a relatively small class of partial differential equation structures to explore. All the equation discovery tasks in Paddles are solved using Lagramge (Todorovski & Džeroski, 1997).

Applications

Equation discovery methods have been applied to various tasks of discovering equation-based laws and models from measured and/or simulation data. Application domains range from physics (mechanical and electrical engineering, fluid dynamics) (Takashi & Hiroshi, 1998; Todorovski & Džeroski, 1997, 2007), through ecology (population dynamics) (Todorovski & Džeroski, 2007; Todorovski et al., 2005) to biochemistry (chemical kinetics) (Džeroski & Todorovski, 2008; Langley, Shiran, Shrager, Todorovski, & Pohorille, 2006).

Cross References

►Inductive Process Modeling
►Language Bias
►Learning as Search
►Linear Regression
►Measurement Scales
►Regression
►System Identification

Recommended Reading

Bridewell, W., Langley, P., Todorovski, L., & Džeroski, S. (2008). Inductive process modeling. *Machine Learning, 71*(1), 1–32.

Džeroski, S., & Todorovski, L. (1995). Discovering dynamics: From inductive logic programming to machine discovery. *Journal of Intelligent Information Systems*, *4*(1), 89–108.

Džeroski, S., & Todorovski, L. (2008). Equation discovery for systems biology: Finding the structure and dynamics of biological networks from time course data. *Current Opinion in Biotechnology*, *19*, 1–9.

Falkenhainer, B., & Michalski, R. (1990). Integrating quantitative and qualitative discovery in the ABACUS system. In Y. Kodratoff & R. Michalski (Eds.), *Machine learning: An artificial intelligence approach*. San Mateo: Morgan Kaufmann.

Kokar, M. M. (1986). Determining arguments of invariant functional descriptions. *Machine Learning*, *1*(4), 403–422.

Križman, V., Džeroski, S., & Kompare, B. (1995). Discovering dynamics from measured data. *Electrotechnical Review*, *62*(3–4), 191–198.

Langley, P. (1981). Data-driven discovery of physical laws. *Cognitive Science*, *5*(1), 31–54.

Langley, P., Shiran, O., Shrager, J., Todorovski, L., & Pohorille, A. (2006). Constructing explanatory process models from biological data and knowledge. *Artificial Intelligence in Medicine*, *37*(3), 191–201.

Schaffer, C. (1993). Bivariate scientific function finding in a sampled, real-data testbed. *Machine Learning*, *12*(1–3), 167–183.

Takashi, W., & Hiroshi, M. (1998). Discovery of first-principle equations based on scale-type-based and data-driven reasoning. *Knowledge-Based Systems*, *10*(7), 403–411.

Todorovski, L., Bridewell, W., Shiran, O., & Langley, P. (2005). Inducing hierarchical process models in dynamic domains. In M.M. Veloso & S. Kambhampati (Eds.), *Proceedings of the twentieth national conference on artificial intelligence*, Pittsburgh, PA, USA.

Todorovski, L., & Džeroski, S. (1997). Declarative bias in equation discovery. In D.H. Fisher (Ed.), *Proceedings of the fourteenth international conference on machine learning*, Nashville, TN, USA.

Todorovski, L., & Džeroski, S. (2007). Integrating domain knowledge in equation discovery. In S. Džeroski & L. Todorovski (Eds.), *Computational discovery of scientific knowledge. LNCS* (Vol. 4660). Berlin: Springer.

Todorovski, L., Džeroski, S., Srinivasan, A., Whiteley, J., & Gavaghan, D. (2000). Discovering the structure of partial differential equations from example behaviour. In P. Langley (Ed.), *Proceedings of the seventeenth international conference on machine learning*, Stanford, CA, USA.

Zembowitz, R., & Zytkow, J. (1992). Discovery of equations: Experimental evaluation of convergence. In W. R. Swartout (Ed.), *Proceedings of the tenth national conference on artificial intelligence*, San Jose, CA, USA.

Error

►Error Rate

Error Correcting Output Codes

Synonyms
ECOC

Definition
Error correcting output codes are an ►ensemble learning technique. It is applied to a problem with multiple classes, decomposing it into several binary problems. Each class is first encoded as a binary string of length T, assuming we have T models in the ensemble. Each model then tries to separate a subset of the original classes from all the others. For example, one model might learn to distinguish "class A" from "not class A." After the predictions, with T models we have a binary string of length T. The class encoding that is closest to this binary string (using Hamming distance) is the final decision of the ensemble.

Recommended Reading
Kong, E. B., & Dietterich, T. G. (1995). Error-correcting output coding corrects bias and variance. In *International conference on machine learning*.

Error Curve

►Learning Curves in Machine Learning

Error Rate

Kai Ming Ting

Synonyms
Error

Definition
Error rate refers to a measure of the degree of prediction error of a ►model made with respect to the true model.

The term *error rate* is often applied in the context of ►classification models. In this context, *error rate* = $P(\lambda(X) \neq Y)$, where XY is a joint distribution and the classification model λ is a function $X \rightarrow Y$. Sometimes this quantity is expressed as a percentage rather than a value between 0.0 and 1.0.

Two common measures of *error rate* for ►regression models are ►mean squared error and ►mean absolute error.

The error rate of a model is often assessed or estimated by applying it to test data for which the ►class labels (Y values) are known. The error rate of a classifier on test data may be calculated as *number of incorrectly classified objects/total number of objects*. Alternatively, a smoothing function may be applied, such as a ►Laplace estimate or an ►m-estimate.

Error rate is directly related to ►accuracy, such that *error rate* = 1.0 − *accuracy* (or when expressed as a percentage, *error rate* = 100 − *accuracy*).

Cross References
►Accuracy
►Confusion matrix
►Mean absolute error
►Mean squared error

Error Squared

Synonyms
Squared error

Definition
Error squared is a common ►loss function used with ►regression. This is the square of the difference between the predicted and true values.

Estimation of Density Level Sets

►Density-Based Clustering

Evaluation

Evaluation is a process that assesses some property of an artifact. In machine learning, two types of artifacts are most commonly evaluated, ►models and algorithms. ►Model evaluation often focuses on the predictive efficacy of the model, but may also assess factors such as its complexity, the ease with which it can

be understood, or the computational requirements for its application. ►Algorithm evaluation often focuses on evaluation of the models an algorithm produces, but may also appraise its computational efficiency.

Evaluation Data

►Test Data
►Test Set

Evaluation Set

►Test Set

Evolution of Agent Behaviors

►Evolutionary Robotics

Evolution of Robot Control

►Evolutionary Robotics

Evolutionary Algorithms

Synonyms

Evolutionary computation; Evolutionary computing; Genetic and evolutionary algorithms

Definition

Generic term subsuming all machine learning and optimization methods inspired by neo-Darwinian evolution theory.

Cross References

►Coevolutionary Learning
►Compositional Coevolution
►Evolutionary Clustering
►Evolutionary Computation in Economics
►Evolutionary Computation in Finance
►Evolutionary Computational Techniques in Marketing
►Evolutionary Feature Selection and Construction
►Evolutionary Fuzzy Systems
►Evolutionary Games
►Evolutionary Kernel Learning
►Evolutionary Robotics
►Neuroevolution
►Nonstandard Criteria in Evolutionary Learning
►Test-Based Coevolution

Evolutionary Clustering

David Corne[1], Julia Handl[2], Joshua Knowles[2]
[1]Heriot-Watt University, Edinburgh, UK
[2]University of Manchester

Synonyms

Cluster optimization; Evolutionary grouping; Genetic clustering; Genetic grouping

Definition

Evolutionary clustering refers to the application of ►evolutionary algorithms (also known as genetic algorithms) to data ►clustering (or cluster analysis), a general class of problems in machine learning, with numerous applications throughout science and industry. Different definitions of data clustering exist, but it generally concerns the identification of homogeneous groups of data (clusters) within a given data set. That is, data items that are similar to each other should be grouped together in the same *cluster* or *group*, while (usually) dissimilar items should be placed in separate clusters. The output of any clustering method is therefore a specific collection of clusters. If we have a specific way to evaluate (calculate the quality of) a given grouping into clusters, then we can consider the clustering task as an optimization problem. In general, this optimization problem is NP hard, and it is common to address it with advanced heuristic or metaheuristic methods. Evolutionary algorithms are prominent among such methods, and have led to a variety of promising and successful techniques for cluster optimization.

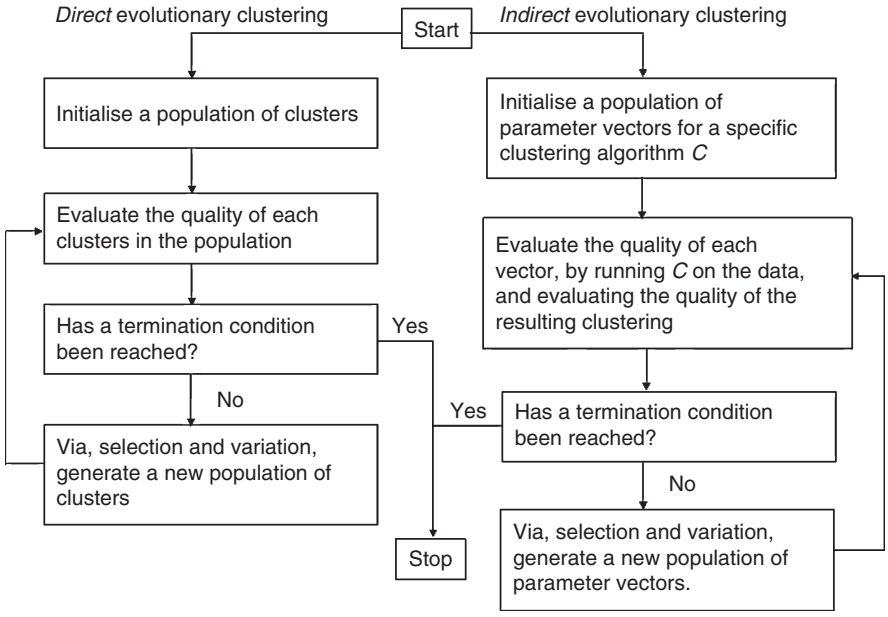
Motivation and Background

In many problem-solving scenarios, we have large amounts of data. We need to cluster those data sensibly into groups in order to help us understand the problem and decide how to proceed further (see ►clustering). It is common, in fact, for this initial "cluster analysis" stage to be the most important (or only) stage in the investigation. In bioinformatics, for example, a frequent activity is the clustering of gene expression data (data that indicate, for a specific cell, how active each of several thousands of genes are at different points in time, or under different experimental conditions). A very important current challenge is to understand the role of each gene; by clustering such data, which means arranging genes into groups such that genes in the same group have similar patterns of activity, we find important clues about genes whose role is currently unknown, simply by assigning their putative role as being related to that of genes (whose role is known) that are in the same cluster. Meanwhile, a ubiquitous situation in industry and commerce is the clustering of data about customers or clients. Here, the role of clustering is all about identifying what types of clients (for example, based on age, income, postcode, and many other attributes that may make up a customer's profile) buy or use certain kinds of products and services. Effective ways to identify groups enable companies to better target their products and their direct marketing campaigns, and/or make more effective decisions about loans, credit and overdrafts. Many machine learning techniques can be used to predict things about customers, or predict things about genes, and so forth. However, the value of clustering (in a similar way to visualization of the data) is that it can lead to a much deeper understanding of the data, which in turn informs the continuing process of applying machine learning methods to it. In this general context, there are many well-known and well-used clustering methods, such as *k*-means, hierarchical agglomerative clustering, neighbor-joining, and so forth. However, there are also well-known difficulties with these methods; specifically, there is often a need to choose in advance the number of clusters to find in the data, and: they tend to be strongly biased towards finding certain types of groupings. For these reasons, methods that are more flexible have been recently investigated, and evolutionary clustering techniques are prominent among these. They are flexible in that (unlike *k*-means, for example), the choice of the number of clusters does not have to be made a priori, and the method is not tied to any particular way of identifying the distance between two items of data, nor is there any a priori ►inductive bias concerning what counts as a good clustering. That is, in broad terms, an evolutionary clustering algorithm allows a user to decide in advance on a definition of cluster quality that is suitable for the problem at hand, and to decide in advance how many clusters are sought, or to leave that decision open; these decisions are then "plugged in to" the algorithm which then proceeds to search for good clusterings.

Structure of Learning System

Evolving Clusters and Evolving Clustering Algorithms

Given a dataset to be clustered, the concept of evolutionary clustering covers two distinct ways in which we can address the problem of finding the best clustering. Each of these approaches is under continuing research, and has proven successful under different conditions. The first approach is to use an evolutionary algorithm to search the space of candidate groupings of the data; this is the most straightforward approach, and perhaps the most flexible in the sense discussed above. The second approach is to "wrap" an evolutionary algorithm around a simpler clustering algorithm (such as *k*-means), and either use the evolutionary algorithm to search the space of features for input to the clustering algorithm (i.e., the evolutionary algorithm is doing ►feature selection in this case), or to search a space of parameters, such as the number of clusters, feature weights, and/or other parameters of the clustering algorithm in use. Central in all of these approaches is a way to measure the quality of a clustering, which in turn depends on some given metric that provides a distance between any pair of data items. Although some applications often come with pre-identified ways to measure distance and cluster quality, in the following we will assume the most common approach, in which distance is the Euclidean distance between the data items (perhaps Hamming distance, in cases where the data are not numeric), and the measure of quality for a given clustering is the ratio of *within-cluster* and *between-cluster*, where *within-cluster* is the mean distance between pairs of items that are in the same cluster, and *between-cluster*



Evolutionary Clustering. Figure 1. The two main approaches to evolutionary clustering; direct (*left*) and indirect (*right*)

is the mean distance between pairs of items that are in different clusters.

We illustrate the two main approaches to evolutionary clustering in Fig. 1.

On the left in Fig. 1, we see the direct approach, in which the evolutionary algorithm searches the space of clusterings of the data. The key features in this approach are the encoding and ▶genetic operators. After evaluating the quality of each of a population of clusterings, a new population is generated from the old one via selection and variation. Essentially, some individuals from the current population are treated as "parents," and new ones are produced from these by using genetic operators. The encoding specifies precisely how a specific data clustering is represented; while the operators specify how new clusterings are derived from the old ones. To take a simple example, suppose we needed to cluster 10 items (A, B, C,…, J) into an arbitrary number of groups. In a simple encoding, we might represent a clustering as a vector of 10 labels, independently chosen from 1 to 10, in which the *i*th element gives the group label of the *i*th item. Hence, the following individual in our population of clusterings:

2 3 5 5 1 5 7 3 2 7

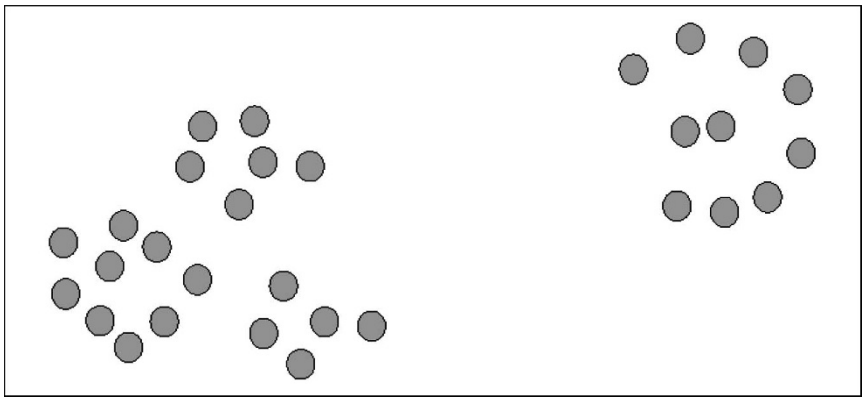
represents the following grouping:

(A, I) (B, H) (C, D, F) (E) (G, I)

Given such a representation, a typical genetic operator might be to randomly change a single label in a single parent. For example, we may choose the fifth element in the above vector and change it randomly to 7, effectively placing item E in the same group as items G and I. Further notes about operators for this and other encodings are given in a special subsection below.

There are several examples of the second type of approach, called "indirect" evolutionary clustering in the Fig. 1 (right). This approach is often used where the "internal" clustering method ("C," in the figure) is very sensitive to initialization conditions and/or parameters of the metric in use to measure distance between items. For example, if C is the *k*-means algorithm, then, for each application of C, we need choices for the parameter *k*, and for each of *k* initial cluster center positions in the data space. The parameter vectors referred to in the figure would be precisely these; the evolutionary algorithm searches this parameter space, finding those that lead to an optimized clustering from *k*-means.

Figure 2 illustrates why this will often be a more effective approach than *k*-means alone. In this case, it is entirely unclear whether these data form two, four, or even five clusters. There are two widely separated groups of points, and this two-cluster solution may be



Evolutionary Clustering. Figure 2. An example with many potential interpretations of the number of clusters

easily found by a 2-means algorithm. However, to the human eye there is also a clear four-cluster solution, further analysis of which may lead to better understanding of these data. This four-cluster solution is difficult for a 4-means algorithm to find, depending on very fortunate initial settings for the cluster centers. Meanwhile, it is worth noting that there are potentially five clusters, as the group on the right can be perceived as a central group of two items, surrounded by a single backward-C-shaped group. The "backward-C" cluster is an example that simply cannot be reliably detected (*as a distinct cluster from the group of two items contained within it*), with most standard cluster analysis methods. Traditional approaches invariably incorporate the assumption that clusters will be centered around a particular position, with the likelihood of a point belonging to that cluster depending monotonically on distance from that position. However, on of the strengths of evolutionary clustering is that it provides the flexibility to work effectively with arbitrary definitions of what may constitute a valid cluster.

Encodings and Operators for Evolutionary Clustering

The more frequently researched style of evolutionary clustering is the direct approach, and the development of this approach in recent years is essentially characterized by certain key ideas for the encoding method. Encodings range from the straightforward representation noted above (with the *i*th gene coding for the cluster membership of the *i*th data item), to more complex representations, such as matrix-based or permutation-based representations.

Before providing a brief description of other encodings it is worth briefly examining a well-known disadvantage of the simple encoding. Given that they have a population, evolutionary algorithms offer the opportunity to use multi-parent genetic operators – that is, we can design operators that produce a new candidate clustering given two or more "parent" clusterings. Such operators are neither mandatory nor necessarily beneficial in evolutionary algorithms, and there is much literature discussing their merits and how this depends on the problem at hand. However, they are often found helpful, especially in cases where we can see some intuitive merit in combining different aspects of parent solutions, resulting in a new solution that seems to have a chance at being good, but which we would have been immensely unlikely to obtain from single-parent operators given the current population. In this context, we can see, as follows, that the opposite seems to be the case when we use standard multi-parent operators with the simple encoding. Suppose the following are both very good clusterings of ten items:

Clustering 1: 1 1 1 1 1 2 2 2 2 2

Clustering 2: 2 2 2 2 2 1 1 1 1 1

Clearly, a good clustering of these items places items 1–5 together, and items 6–10 together, in separate groups. It is also clear, however, that using a standard crossover operator between these two parents (e.g., producing a child by randomly choosing between clusterings for each item in turn) will lead to a clustering that mixes items from these two groups, perhaps even combining

them all into one group. The main point is that a crossover operation destroys the very relationships between the items that underpinned the fitness of the parents.

One of the more prominent and influential representations for clustering, incorporating a design for far more effective multi-parent operators, was that of Falkenauer's "Grouping Genetic Algorithm," which also provides a general template for the implementation of evolutionary algorithms for grouping problems. The essential element of Falkenauer's method is that multi-parent operators recombine entire groups rather than item labels. For example, suppose we encode two clusterings explicitly as follows:

Clustering 3: (A, I, B, H) (C, G) (D, E, F, J)

Clustering 4: (A, I, B, H) (C, D, J) (E, F, G)

A Falkenauer-style crossover operator works as follows. First, we randomly choose some entire groups from the first parent and some entire groups from the second parent; the child in this case might then be:

(A, I, B, H) (C, G) (E, F, G)

in which the groups that come from the first parent are underlined. Typically, we will now have some repeated items; we remove the entire groups that contain these items and came from the first parent, in this case leaving us with:

(A, I, B, H) (E, F, G)

The final step is to add back the missing items, placing them one by one into one of the existing groups, or perhaps forming one or more new groups. The application in hand will often suggest heuristics to use for this step. In clustering, for example, we could make use of the mean Euclidean distance from items in the groups so far. Whatever the end result in this case, note that the fact that A, I, B, and H were grouped together in both parents will be preserved in the child. Similarly, the E, F, G grouping is inherited directly from a parent.

A more recent and effective approach, specifically for clustering, is one first proposed in Park and Song (1998) called a link-based encoding. In this approach, the encoding is simply a list of item indices, and is interpreted as follows. If the *i*th element in the permutation is *j*, then items *i* and *j* are in the same group. So, for example,

B C E E A E G C B G

represents the following grouping:

(A, B, C, D, E, H, I) (F, G, J)

Standard crossover operators may be used with this encoding, causing (intuitively) a reasonable degree of exploration of the space of possible clusterings, yet preserving much of the essential "same-group" relationships between items that were present in the parents. In Handl and Knowles (2007) it is shown why this encoding is effective compared with some alternatives.

We also briefly note other encodings that have been prominent in the history of this subfield. An early approach was that of Jones and Beltramo, who introduced a "permutation with separators" encoding. In this approach, a clustering is encoded by a permutation of the items to be clustered, with a number of *separators* indicating cluster boundaries. For example, if we have ten items to cluster (A–J) and use S as the separator, the following is a candidate clustering:

A I B H S C G S D E F J

representing the same grouping as that of "Clustering 3" above. Jones and Beltramo offered a variant of this encoding that is a cross between the direct and indirect approaches. In their *greedy permutation* encoding, a clustering is represented by a permutation (with no separator characters), with the following interpretation: the first *k* items in the permutation become the centers of the first *k*clusters. The remaining items, in the order they appear, are added to whichever cluster is best for that item according to the objective function (clustering quality metric) in use.

Evolutionary Multiobjective Clustering

It can be strongly argued that the clustering problem is inherently *multiobjective*, yet most methods employ only a single performance criterion to optimize. In fact, there are at least three groups of criteria commonly used (but usually one at a time) in clustering (both evolutionary clustering and other methods). These are: compactness, connectedness, and spatial separation. When an algorithm optimizes for compactness, the idea is that clusters should consist of highly homogeneous data items only – that is, the distance (or other measure of variation) between items in the same cluster should

be small. In contrast, if we optimize the degree of connectedness, then we are increasing the extent to which neighboring data items should share the same cluster. This can deal with arbitrarily-shaped clusters, but can lack robustness when there is little spatial separation between clusters. Finally, spatial separation is usually used as a criterion in combination with compactness, or with a measure of the balance of cluster sizes.

In *multiobjective clustering*, the idea is to explicitly explore the solutions that are trade-offs between the conflicting criteria, exploiting the fact that these trade-off solutions are often the ones that most appeal as intuitively "correct" solutions to a clustering problem. Handl and Knowles make use of Park and Song's link-based encoding in their multiobjective evolutionary algorithm, MOCK, which treats a clustering problem as a two-objective problem, using measures of compactness and connectedness for the two objectives. MOCK's multiobjective search process is based on the PESA-II evolutionary multiobjective optimizer (Corne, Jerram, Knowles & Oates, 2001). Following use of MOCK for a clustering problem, an intermediate result (inherent in multiobjective optimization methods) is a (possibly large) collection of different clusterings. These will range from clusterings that score very well on compactness but poorly on connectedness, through to clusterings that achieve excellent connectedness at the expense of poor compactness. It is useful to note that the number of clusters tends to increase as we go from poor connectedness to high-connectedness clusters. Arguably, in many applications such a collection of alternative solutions is useful for the decision-maker. Nevertheless, the MOCK approach incorporates an automated model-selection process that attempts to choose an ideal clustering from the discovered approximate Pareto front. This process is oriented around the notion of determining the "right" number of clusters, and makes use of Tibshirani, Walther, and Hastie (2001) gap statistic (full details are in Handl & Knowles, 2007). Extensive comparison studies, using a wide variety of clustering problems and comparing with many alternative clustering methods, show consistent performance advantages for the MOCK's approach.

Cross References

►Clustering
►Feature Selection
►Semi-Supervised Learning
►Supervised Learning
►Unsupervised Learning

Recommended Reading

Cole, R. M. (1998). *Clustering with genetic algorithms*. Masters dissertation, Department of Computer Science, University of Western Australia.

Corne, D. W., Jerram, N. R., Knowles, J. D., & Oates, M. J. (2001). PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the GECCO* (pp. 283–290).

Delattre, M., & Hansen, P. (1980). Bicriterion cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *2*(4), 277–291.

Falkenauer, E. (1998). *Genetic algorithms and grouping problems*. New York: Wiley.

Handl, J., & Knowles, J. (2007). An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, *11*(1), 56–76.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Surveys*, *31*(3), 264–323.

Jones, D. R., & Beltramo, M. A. (1991). Solving partitioning problems with genetic algorithms. In R. K. Belew & L. B. Booker (Eds.), *Proceedings of the fourth international conference on genetic algorithms* (pp. 442–449). Morgan Kaufmann.

Park, Y.-J., & Song, M.-S. (1998). A genetic algorithm for clustering problems. In *Proceedings of the third annual conference on genetic programming* (pp. 568–575). Morgan Kaufman.

Tibshirani, R., Walther, G., & Hastie, T. (2001). Estimating the number of clusters in a dataset via the Gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *63*(2), 411–423.

Evolutionary Computation

►Evolutionary Algorithms

Evolutionary Computation in Economics

Serafín Martínez-Jaramillo, Biliana Alexandrova-Kabadjova,
Alma Lilia García-Almanza,
Tonatiuh Peña Centeno
Bank of Mexico,
Mexico, D.F.

Definition

Evolutionary computation (EC) in economics is an area of knowledge which involves the use of any of the EC techniques, also known as evolutionary algorithms (EAs), in order to approach the topics within the economic sciences. This area of knowledge is different from

the Evolutionary Economics field which does not necessarily apply EC techniques to study economic problems. The use of EC in economics pursues different purposes mainly to overcome some of the limitations of the classical economic models and to relax some of the strong assumptions made in such models.

Motivation and Background

Evolutionary computation (EC) is a branch of Machine Learning which is inspired in many forms by the principle of evolution. EC techniques, among many other machine learning techniques, have proven to be quite flexible and powerful tools in many different fields and disciplines. Economics-affine fields are by no means the exception for this widespread use of these evolutionary inspired techniques.

In addition to the undeniable necessity of computing in almost every aspect of our modern lives, numerous problems in economics possess algorithmic nature. Therefore, economists must consider computational complexity as an important analysis tool due to the fact that some of such problems belong to the dislikable class of NP-complete (The NP-complete computational complexity class is a subset of "harder" problems from the NP computational class, which is the set of all the decision problems which can be solved using a Nondeterministic Turing Machine in polynomial time). problems. Having said so, EC has been intensively used as an alternative approach to analytical methods in order to tackle numerous NP-complete problems with relative good success.

The first work in economics (Clarifying: such first work approached a classic game known as the Prisoners' Dilemma), which involved the use of EC dates back to the 1980s, in Axelrod and Hamilton (1981) and Axelrod (1987) the authors used Genetic Algorithms (GAs) to derive strategies for the Iterated Prisoner's Dilemma (IPD). From then, EC techniques in economics had been used in areas such as macroeconomics, econometrics, game theory, auctions, learning and agent-based models. There is even a school of thought in economics known as "Evolutionary Economics" (See for example Witt (2008) for an introduction), whose approach to the study of economics involves concepts in evolution but does not necessarily rely on EC techniques.

Rationality and Learning

One of the most relevant concepts in the economics science is the concept of rationality. This concept is at the core of most of the economic models, since it is frequently assumed that economic agents behave in a fully rational way. Unfortunately, it is not clear if such assumption holds after the irrational behavior observed during the recurrent financial crises.

Herbert A. Simon is probably the best known scientist to claim that "decision-making" under uncertainty is not a fully rational process. He developed his theory based on the concept of "bounded rationality" (Simon, 1957), and he was one of the pioneers in the field of artificial intelligence (AI) as well as a highly reputed psychologist and economist. Later, in Arthur (1991), the author made important contributions to the development of agents with bounded rationality, using computational tools. In addition, recent ideas about rationality from a computer scientist's point of view are found in Tsang (2008). In this context to be more precise about the meaning of bounded rationality, let us quote Herbert A. Simon:

- ... boundedly rational agents experience limits in formulating and solving complex problems and in processing (receiving, storing, retrieving, transmitting) information...

Some other common assumptions behind the classical economic theory are that the participants of the model have *homogeneous preferences* and they *interact globally* (Axtell, 2000). In other words, having limited number of participants in the model, the theorists assume that those individuals exhibit the same preferences and all of them interact with each other. These agents are called "representative agents." Moreover, the analysis is focused only at the point of equilibrium, and aspects such as asymmetric information, imperfect competition and network externalities are not considered.

Departing from the assumption of full rationality and homogeneous expectations, the horizon (and the design issues) opens widely. The modeling of the learning behavior of the agents is a central part of the research agenda in computational economics. Regarding the agents' learning process, in Lucas (1986), the author provided an interpretation of adaptive behavior from the economics point of view:

- In general terms, we view or model an individual as a collection of decision rules (rules that dictate the action to be taken in given situations) and a set of preferences used to evaluate the outcomes arising from particular situation-action combinations. These decision rules are continuously under review and revision; new decision rules are tried and tested against experience, and rules that produce desirable outcomes supplant those that do not.

There are many useful techniques to implement what Lucas defined as adaptive learning, like ►genetic algorithms (GAs), as has been done in Bullard and Duffy (1999), and ►genetic programming (GP) as has been done in Martinez-Jaramillo and Tsang (2009b). GP has been previously described as a suitable way to model economic learning in Edmonds (1999). In Brenner (2006), the author provides us a summary of the available options to model agent behavior and learning in economics.

Nevertheless, the more traditional economists are still reluctant to accept an approach in which there is not a rational expectations type of agent, where instead there are inductive, boundedly rational heterogeneous agents (Arthur, 1994).

Economic and Econometric Models

Two of the most relevant areas in economics are macroeconomics and econometrics. Macroeconomics is the branch of economics which analyzes the national economy and its relations with the international economy. Macroeconomic analysis tries to understand the relationships between the broad sectors of the economy by making use of aggregated economic variables such as inflation, unemployment, interest rates, total output, etc. EC has been used in order to analyze some of such macroeconomic variables, a field which is dominated by econometric analysis. Econometrics is a field within the wider area of economics which involves the use of statistics and its tools for the measurement of relationships postulated by economic theory (Greene, 2003).

Many methods in econometrics involve an optimization process, and it is well known that EC is particularly suitable for optimization problems. Probably one of the first applications of GP in econometrics was done by the creator of GP himself in Koza (1992). Additionally, in Agapie and Agapie (2001) the authors use GAs and simulated annealing (SA) for econometric modeling. They found that the performance of the evolutionary algorithms (EAs) is better than the performance of traditional gradient techniques on the specific models in which they performed the comparison. Finally, Östermark (1999) uses a Hybrid GA in several ill-conditioned econometric and mathematical optimization problems with good results.

In addition to the usage of EC in econometrics, some classical economic models such as the Cobweb model and exchange rate models had been also approached with EC techniques. For instance, in Arifovic (1994) and Chen and Yeh (1996) to approach the Cobweb model, in the former work the author uses GAs, whereas in the latter the authors use GP. Furthermore, Arifovic explores the use of GAs in foreign exchange markets in Arifovic (1996). The GA mechanism developed in such works evolved decision rules that were used to determine the composition of the agents' portfolios in a foreign exchange market. Arifovic made two observations rarely seen in the standard overlapping generations (OLG) model with two currencies. First, she evidenced that the returns and exchanges rates were generated endogenously, and second, she observed that the model's equilibrium dynamics is not stable and shows bounded oscillations (the theoretical model implies a constant exchange rate).

The use of GAs in economic modeling is not restricted to the above mentioned works. In Bullard, Arifovic, and Duffy (1995), the authors studied a version of the growth model in which the physical capital is accumulated in a standard form, but the human capital accumulation is subject to increasing returns. In their model, the agents take two decisions when they are young: how much to save by renting physical capital to the companies and how much to invest in training. Returns on training depend on the average level of the human capital of the economy. The authors introduce the agents' learning by means of GAs. In Marimon, McGrattan, and Sargent (1990), Marimon develops an economic model in which the agents adapt by means of a GA.

Game Theory

Game Theory is a branch of applied mathematics that attempts to model the individual's strategic behavior.

The first study considered to establish the fundamentals of the field is the book "Theory of Games and Economic Behavior" (von Neumann & Morgenstern, 1944). The idea behind this theory is that the success of the individual's decisions depends on the decisions of others. While originally, the aim of the theory was to study the competition in which the agent does better at another's expense (zero sum games), now it has been extended to study a wider class of interactions among individuals. Furthermore, it is extensively used in economics, biology, and political science among some other disciplines.

A well-defined mathematical object, the game consists of a set of players and a set of strategies (decisions) available to those players. In addition, for each combination of strategies a specification of payoffs is provided. The aim of the traditional applications of the game theory was to find a Nash equilibrium, a solution concept, in which each player of the game adopts a strategy that is unlikely to be changed. This solution concept was named after John Nash, whose work was published in the early 1950s (Nash, 1950). Nevertheless, it took almost 20 years to fully realize what a powerful tool Nash has created. Nowadays, Game Theory is one of the best established theories in economics and it has been extensively used to model the interactions between the economic agents. However, games typically have many Nash equilibria and one of the main assumptions is that the agents behave in a rational way. In more realistic games, the equilibrium selection problem does not have an easy solution though, and the human behavior observed in real life is frequently irrational.

Given the above mentioned constraints, in order to go further, the Evolutionary Game Theory was originated as an application of the mathematical theory of games to biological contexts (see ►Evolutionary Games). In this field, Maynard Smith is considered to be the first one to define the concept of Evolutionary Stable Strategy in Maynard Smith (1972). Furthermore, the possibility of using computer modeling as an extension of the theory of games was first explored in Axelrod and Hamilton (1981). Since then, computer science has been used in traditional game theory problems, like the strategic behavior of agents in auctions, auction mechanism design, etc. By providing approximate solutions in such complex problems this approach can be useful where analytical solutions have not been found. For instance, the iterative prisoners' dilemma is one of the most studied games by researchers in computer science (Axelrod, 1987). The prisoners' dilemma is a classic game that consists of the decision-making process by two prisoners who can choose to cooperate or to defect. In the case that the two prisoners choose to cooperate they get a payoff of three each, in the case that both choose to defect they get a payoff of one each, and in the case that any of them decides to defect and the other to cooperate, the former gets a payoff of five and the latter a payoff of zero. In equilibrium, both players decide to defect despite the fact that would be better for them to cooperate.

Axelrod organized a tournament on the iterated prisoners' dilemma in which he asked people from game theory and amateurs to provide him with strategies. The surprising result was that a very simple strategy (Tit for Tat) won the tournament (Axelrod, 1987). After the reporting of the results from such tournament, Axelrod was able to provide some mathematical results on how cooperation can emerge in a population of egoists. The previous example clearly illustrates how beneficial was the use of computer science to obtain theoretical results in a problem where analytical methods alone have not delivered the desired outcomes.

Game theory is one of the most important areas in economics because it has applications to many fields, such as corporate decision making, microeconomics, market modeling, public policy analysis, environmental systems, etc. We can find more applications of EC to game theory than the IPD. For example, another work related to game theory and EC is the one done by Duffy and Engle-Warnick (2002), which deals with the well-known two-player, repeated ultimatum game. In this work they used GP as a means of inferring the strategies that were played by subjects in economic decision-making experiments. Other works, within the field of EC and game theory, are the duopoly and oligopoly games (Chen & Ni, 2000). References regarding cooperation, coalition, and coordination are also made often and usually driven by EC techniques, Vriend (1995). In Jin and Tsang (2006), the authors applied GP to find strategies for sequential bargaining procedure and confirmed that equilibria can be approximated by GP. This gives opportunity to find approximate solutions to more complex situations for which theoretical solutions are yet to be found.

The interesting research by Riechmann (2002) proposes to study the foundations of the GAs by means of game theory. Riechmann interprets the GA as an N-players repeated game in which an individual of the GA represents a player with a different strategy. Once the author achieves the interpretation of the learning process of a GA as an evolutionary game, he attempts to shed some light on the fundamentals of GAs.

Auction Theory

Auction theory studies the behavior of the participants in auction markets. The study of auctions is relevant because they define the protocol which is followed by the participants in some important markets; for example, some stock markets, such as the New York Stock Exchange, operate under a double auction-like mechanism. There are many different types of auctions: the English auction, the Dutch auction, the Vickrey auctions, etc. In Klemperer (2004), there is a good introduction to the field.

EC techniques, particularly GAs, have been intensively used in auctions to derive bidding strategies in simulated auctions. In Andreoni and Miller (1995), the author uses adaptive learning, modelled with a GA, in order to capture patterns which arise in experimental auctions with humans. Such bidding patterns cannot be explained by the theoretical models, something that allowed the exploration of alternative methods such as adaptive behavior by means of EC. Some other relevant examples of the study of auctions using EC techniques are Anthony and Jennings (2003), Byde (2003), Cliff (2003), Dawid (1999), Mochon, Quintana, Sáez, and Isasi (2005), and Saez, Quintana, Isasi, and Mochon (2007).

Agent-Based Models

Agent-based computational economics (ACE) can be thought of as a branch of a wider area: Agent-based Modeling (ABM) (Wooldridge, 2002). The field of agent-based modeling is not restricted to economics, it has been applied in social sciences in general (Axelrod, 2003), in some classical and not so classical problems in computer science, and in some other disciplines. Axelrod provides an account of his experience using the agent-based methodology for several problems and he suggests that the ABM can be seen as a bridge between disciplines. Axelrod and Tesfatsion provide a good guide to the relevant literature of the ABM in Axelrod and Tesfatsion (2006). In Chen (2007), there is a good introduction to agents in economics and finance; in such work, Chen conceives the agents not just as economic agents but as computational intelligent units.

Most of the economic and finance theory is based on what is known as investor homogeneity or the representative agent. In ACE the researchers can depart from the assumptions of homogeneous expectations and perfect rationality by means of computational-based economic agents. In 2006, Tesfatsion surveys some of the most important works and topics on this area of research.

In ACE one of the main goals is to explain the macrodynamics of the economy by means of the microinteractions of the economic agents. This approach to the study of the economy has been called a "bottom-up" approach in opposition to the more traditional approaches in economics. An additional purpose of ACE is to handle real-world issues, which has become possible due to the technological advances in computational tools. With the use of programming languages, the agent-based approach allows us to represent explicitly agents with bounded rationality and heterogeneous preferences. Given a specific social structure, the simulation of the interaction among agents is the strength and the heart of the ABM. Even in its early stage of development, ABM is a promising area of research, which has opened the opportunity to social scientists to look for new insights in resolving relevant real-world issues. Considered "the third way of doing science" (Axelrod, 2003), modeling the behavior of the autonomous decision-making entities allows researchers to simulate the emergence of certain phenomena in order to gain better understanding of the object of study (Axtell, 2000). In this sense ACE, defined as "the computational study of economic processes modelled as dynamic systems of interacting agents" (Tesfatsion, 2006), is a growing area in the field of ABM. ACE research is developing rapidly, by using machine learning techniques, the researchers model the agents as software programs able to take autonomous decisions. Consequently, the interactions among the individuals at the microlevel give rise to regularities at the macrolevel (globally). The intention is to observe the emerging self-organizing process for a certain period of time, in order to study the presence of patterns or the lack of them.

Currently, the study of this self-organizing capability is one of the most active areas of ACE research.

One of the most crucial tasks in representing explicitly the market participants is the simulation of their autonomous decisions. Nowadays, advances in AI have opened possibilities of tackling this issue. In particular, techniques such as neural networks (NNs), genetic algorithms (GAs), genetic programming (GP), and other population-based algorithms are widely used in the field.

There are some interesting works in which the agent-based methodology is compared with experiments performed with human beings (Chan, LeBaron, Lo, & Poggio, 2001; Duffy, 2006). In both the works, the benefits that each type of research has on each other are identified. For instance, experimental research can be used as an important method to calibrate an agent-based model. On the other hand, agent-based simulations can be used to explain certain phenomena present in human experiments. To summarize, there are many beneficial ways in which both types of research influence each other.

According to Tesfatsion, the economic research being done with the ACE methodology can pursue one of two main objectives: the first one is the constructive explanation of macrophenomena and the second is the design of new economic mechanisms. In Tesfatsion (2006), Tesfatsion updates the classification of the research being made in ACE into four main categories: empirical understanding, normative understanding, methodological advancement, and finally, qualitative insight and theory generation.

EAs have been used for the modeling of the agents' learning in multiagent simulations. In multiagent simulations of economics systems, it is possible to find very different approaches and topics, just to illustrate some few examples of the immense amount of works, let us take a look at the following list:

- Electricity markets (Amin, 2002) (Learning Classifier System).
- Payment card markets (Alexandrova-Kabadjova, 2008) (Population Based Incremental Learning).
- Retail petrol markets (Heppenstall, Evans, & Birkin, 2006) (Genetic Algorithms).
- Stock markets (Arthur et al., 1997) (Learning Classifier Systems) and (Martinez-Jaramillo & Tsang, 2009b) (GP).
- Foreign exchange markets (Arifovic, 1996; Izumi & Ueda, 2001) (Genetic Algorithms).

Related to payment methods and systems, another economic phenomena characterized with complex social interaction suitable for ABM is the market dynamics of some electronic payment instruments, such as payment cards. In this field, the first evolutionary computation model was introduced in Alexandrova-Kabadjova (2008). This paper studies the competition among payment card scheme. The authors apply a Generalized Population Based Incremental Learning Algorithm (GPBIL), an extended version of the PBIL algorithm, in order to find an optimal price strategy for the electronic payment instrument.

Cross References

►Evolutionary Algorithms
►Evolutionary Computation in Finance
►Evolutionary Computational Techniques in Marketing
►Genetic Algorithms
►Genetic Programming

Recommended Reading

Agapie, A., & Agapie, A. (2001). Evolutionary computation for econometric modeling. *Advanced Modeling and Optimization, 3*, 1–5.

Alexandrova-Kabadjova, B. (2008). *Evolutionary learning of the optimal pricing strategy in an artificial payment card market*, *Studies in computational intelligence* (Vol. 100). Berlin: Springer.

Amin, M. (2002). Restructuring the electric enterprise: Simulating the evolution of the electric power industry with intelligent adaptive agents. In A. Faruqui, & K. Eakin, (Eds.), *Market analysis and resource management* (Chap. 3). Boston: Kluwer Publishers.

Andreoni, J., & Miller, J. H. (1995). Auctions with artificial adaptive agents. *Games and Economic Behavior, 10*, 39–64.

Anthony, P., & Jennings, N. R. (2003). Developing a bidding agent for multiple heterogeneous auctions. *ACM Transactions on Internet Technology, 3*, 185–217.

Arifovic, J. (1994). Genetic algorithm learning and the cobweb model. *Journal of Economic Dynamics and Control, 18*, 3–28.

Arifovic, J. (1996). The behavior of the exchange rate in the genetic algorithm and experimental economics. *Journal of Political Economy, 104*, 510–541.

Arthur, W. B. (1991). Learning and adaptiver economic behavior. Designing economic agents that act like human agents: A behavioral approach to bounded rationality. *American Economic Review, 81*, 353–359.

Arthur, W. B. (1994). Inductive reasoning and bounded rationality: The El Farol problem. *American Economic Review, 84*, 406–411.

Arthur, W. B., Holland, J. H., LeBaron, B., Palmer, R. G., & Talyer, P. (1997). Asset pricing under endogenous expectations in an artificial stock market. In W. Brian Arthur, S. Durlauf, & D. Lane, (Eds.), *The economy as an evolving complex system II*. Menlo Park: Addison-Wesley.

Axelrod, R. (1987). The evolution of strategies in the iterated prisoner's dilemma. In L. Davis (Ed.), *Genetic algorithms and simulated annealing, Research notes in AI* (Chap. 3, pp. 32–41). Los Altos, CA: Morgan Kaufmann.

Axelrod, R. (2003). Advancing the art of simulation in the social sciences. *Japanese Journal for Management Information System, Special Issue on Agent-Based Modeling, 12* (3).

Axelrod, R., & Hamilton, W. D. (1981). The evolution of cooperation. *Science, 211*, 1390–1396.

Axelrod, R., & Tesfatsion, L. (2006). A guide for newcomers to agent-based modeling in the social sciences. In K. L. Judd, & L. Tesfatsion, (Eds.), *Handbook of computational economics, Volume 2: Agent-based computational economics*, *Handbooks in economics (Appendix A, pp. 1647–1656)*. Amsterdam: North-Holland.

Axtell, R. (2000). *Why agents? on the varied motivations for agent computing in the social sciences*. Working Paper 17, Center on Social and Economic Dynamics.

Brenner, T. (2006). Agent learning representation – advice in modelling economic learning. In K. L. Judd, & L. Tesfatsion, (Eds.), *Handbook of computational economics, Volume 2: Agent-based computational economics*, *Handbooks in economics* (Chap. 18, pp. 895–948). Amsterdam: North-Holland.

Bullard, J., Arifovic, J., & Duffy, J. (1995). *Learning in a model of economic growth and development*. Working Paper 1995-017A, Federal Reserve Bank Of St. Louis.

Bullard, J., & Duffy, J. (1999). Using genetic algorithms to model the evolution of heterogeneous beliefs. *Computational Economics, 13*, 41–60.

Byde, A. (2003). Applying evolutionary game theory to auction mechanism design. In *ACM conference on electronic commerce* (pp. 192–193). New York: ACM.

Chan, N. T., LeBaron, B., Lo, A. W., & Poggio, T. (2001). *Agent-based models of financial markets: A comparison with experimental markets*. MIT Sloan Working Paper 4195-01, Massachusetts Institute of Technology.

Chen, S.-H. (2007). Editorial: Computationally intelligent agents in economics and finance. *Information Science, 177*(5), 1153–1168.

Chen, S.-H., & Ni, C. C. (2000). Simulating the ecology of oligopolistic competition with genetic algorithms. *Knowledge Information Systems, 2*(2), 285–309.

Chen, S.-H., & Yeh, C.-H. (1996). Genetic programming learning in the cobweb model with speculators. In *International computer symposium (ICS'96). Proceedings of international conference on artificial intelligence* (pp. 39–46), National Sun Yat-Sen University, Kaohsiung, Taiwan, R.O.C.

Cliff, D. (2003). Explorations in evolutionary design of online auction market mechanisms. *Electronic Commerce Research and Applications, 2*, 162–175.

Dawid, H. (1999). On the convergence of genetic learning in a double auction market. *Journal of Economic Dynamics and Control, 23*, 1545–1567.

Duffy, J. (2006). Agent-based models and human subject experiments. In K. L. Judd, & L. Tesfatsion, (Eds.), *Handbook of computational economics, Volume 2: Agent-based computational economics*, *Handbooks in economics* (Chap. 19, pp. 949–1012). Amsterdam: North-Holland.

Duffy, J., & Engle-Warnick, J. (2002). Using symbolic regression to infer strategies from experimental data. In S.-H. Chen (Ed.), *Evolutionary computation in economics and finance* (pp. 61–82). New York: Physica-Verlag.

Edmonds, B. (1999). Modelling bounded rationality in agent-based simulations using the evolution of mental models. In T. Brenner (Ed.), *Computational techniques for modelling learning in economics* (pp. 305–332). Dordrecht: Kluwer.

Greene, W. H. (2003). *Econometric analysis* (5th ed.). Upper Saddle River, NJ: Prentice Hall.

Heppenstall, A., Evans, A., & Birkin, M. (2006). Using hybrid agent-based systems to model spatially-influenced retail markets. *Journal of Artificial Societies and Social Simulation, 9*, 3.

Izumi, K., & Ueda, K. (2001). Phase transition in a foreign exchange market-analysis based on an artificial market approach. *IEEE Transactions of Evolutionary Computation, 5*(5), 456–470.

Jin, N., & Tsang, E. P. K. (2006). Co-adaptive strategies for sequential bargaining problems with discount factors and outside options. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 7913–7920). Washington, DC: IEEE Press.

Klemperer, P. (2004). *Auctions: Theory and practice. The Toulouse lectures in economics*. Princeton, NJ: Princeton University Press.

Koza, J. (1992). A genetic approach to econometric modelling. In P. Bourgine, & B. Walliser, (Eds.), *Economics and cognitive science* (pp. 57–75). Oxford: Pergamon Press.

Lucas, R. E. (1986). Adaptive behavior and economic theory. In R. M. Hogarth, & M. W. Reder, (Eds.), *Rational choice: The contrast between economics and psychology* (pp. 217–242). Chicago: University of Chicago Press.

Marimon, R., McGrattan, E., & Sargent, T. J. (1990). Money as a medium of exchange in an economy with artificially intelligent agents. *Journal of Economic Dynamics and Control, 14*, 329–373.

Martinez-Jaramillo, S., & Tsang, E. P. K. (2009). An heterogeneous, endogenous and coevolutionary gp-based financial market. *IEEE Transactions on Evolutionary Computation, 13*, 33–55.

Mochon, A., Quintana, D., Sáez, Y., & Isasi, P. (2005). Analysis of ausubel auctions by means of evolutionary computation. In *IEEE congress on evolutionary computation (CEC 2005)* (pp. 2645–2652). Edinburgh, Scotland.

Nash, J. (1950). The barganing problem. *Econometrica, 18*, 155–162.

Östermark, R. (1999). Solving irregular econometric and mathematical optimization problems with a genetic hybrid algorithm. *Computational Economics, 13*(2), 103–115.

Riechmann, T. (2002). *Genetic algorithm learning and economic evolution. Studies in fuzziness and soft computing* (pp. 45–59). Heidelberg: Physica-Verlag.
Saez, Y., Quintana, D., Isasi, P., & Mochon, A. (2007). Effects of a rationing rule on the ausubel auction: A genetic algorithm implementation. *Computational Intelligence, 23*(2), 221–235.
Simon, H. A. (1957). *Models of man: Social and rational*. New York: John Wiley.
Maynard Smith, J. (1972). *Game theory and the evolution of fighting* (pp. 8–28). Edinburgh: Edinburgh University Press.
Tesfatsion, L. (2006). Agent-based computational economics: A constructive approach to economic theory. In K. L. Judd & L. Tesfatsion, (Eds.), *Handbook of computational economics, Volume 2: Agent-based computational economics*, *Handbooks in economics* (Chap. 16, pp. 831–880). Amsterdam: North-Holland.
Tsang, E. P. K. (2008). Computational intelligence determines effective rationality. *International Journal of Automation and Computing, 5*, 63–66.
von Neumann, J., & Morgenstern, O. (1944). *Theory of games and economic behavior*. Princeton, NJ: Princeton University Press.
Vriend, N. J. (1995). Self-organization of markets: An example of a computational approach. *Computational Economics, 8*, 205–231.
Witt, U. (2008). *Evolutionary economics* (2nd ed.). Basingstoke, UK: Palgrave Macmillan.
Wooldridge, M. (2002). *An Introduction to multiagent systems*. Chichester: Wiley.

Evolutionary Computation in Finance

Serafín Martínez-Jaramillo, Alma Lilia García-Almanza,
Biliana Alexandrova-Kabadjova,
Tonatiuh Peña Centeno
Bank of Mexico,
Mexico, D.F

Definition

Evolutionary computation (EC) in finance is an area of research and knowledge which involves the use of techniques, known as evolutionary algorithms (EAs), to approach topics in finance. This area of knowledge is similar to EC in economics, in fact such areas frequently overlap regarding some of the topics approached. The application of EC in finance pursues two main goals: first, to overcome the limitations of some theoretical models (and the strong assumptions being made by such models) and second, to innovate in this extremely competitive area of research.

Motivation and Background

Evolutionary computation is a field in Machine Learning in which the developed techniques apply the principle of Evolution in several different ways. The application of EC in finance includes portfolio optimization, financial forecasting, asset pricing, just to mention some examples.

In finance, competition is at the center of the everyday activities by the individuals and companies that participate in this field. For example, in the stock markets everybody is trying to beat the market in order to make more profits than the other participants.

As a result of this fierce competition, there is a constant need to innovate and machine learning has provided novel and competitive tools in financial research. Therefore, it is natural to find numerous problems in finance being approached by any of the existent EC techniques like ►Genetic Programming (GP), ►Genetic Algorithms (GAs), Evolutionary Strategies (EAs), etc. This field has been called in many different ways like computational finance, computational intelligence in finance, etc. Research in this area is still evolving; therefore, it is difficult to define it clearly or to establish its limits. Moreover, nowadays it is almost impossible to provide a full account of all the relevant work that involves any form of EC in finance. It is also hard to organize this vast amount of human knowledge.

Nowadays, computing in finance is an almost unavoidable tool, from Monte Carlo simulation and optimization to computer intensive methods to valuate complex derivatives; in fact, some of the most critical processes in finance make heavy use of computers. Moreover, this research and professional practices have been known as computational finance and the application of evolutionary techniques in finance fit within such definition. Computational finance is now a frequently mentioned term and is frequently associated with financial engineering. However, in this context we refer to computational finance as the use of nonconventional computational techniques, like EC or other machine learning techniques, to tackle problems in

finance. See for example, Tsang and Martinez-Jaramillo (2004) for a good introduction to the field.

Financial Forecasting

Financial forecasting is one of the most important fields in the computational finance area (Tsang & Martinez-Jaramillo, 2004) and EC has been used to solve a great variety of financial forecasting problems, such as, detection of stock price movements, volatility prediction, forecasting of foreign exchange markets, and so on.

Machine learning classifiers, like other forecasting techniques, extend past experience into the future. The aim is to analyze past data in order to identify patterns in the interest of creating a model to predict future events. In this section we will introduce some important works in the financial forecasting area, which take advantage of some of the EC distinctive features. First, the relevance of the interpreatability of the solution is illustrated; after that, some examples about the usefulness of genetating multiple solutions for the same problem are given. Then, some works that use EC as an optimization approach to solve forecasting problems are presented. Finally, the use of a great variety of representations is highlighted. Evolutionary techniques are able to produce interpretable solutions, this property is especially important for predictions, since the main goals of classification are: to generate an accurate classification model that is be able to predict unseen cases and to discover the predictive structure of the problem (Breiman, Friedman, Olshen, & Stone, 1984). Models for understanding provide information about the structural patterns in data that can be useful to recognize the variables' interactions. There are classification models that have good predictive power, however, these provide a poor representation of the solution; for example, ▶Artificial Neural Networks (ANNs). Since EC techniques provide not just a good prediction but an interpretable solution, these have been used in financial problems to acquire knowledge of the event to predict. For example, Tsang, Yung, and Li (2004) trained a GP using past data from the financial stock markets to predict price movements of at least r% in a period of at most n times. The attributes used to train the GP were indicators from technical analysis. Due to the interpretability of the solution, the authors were able to analyze the most successful indicators in the result. In fact, some researchers have used EC in order to discover new financial indicators such as Allen and Karjalainen (1999), who made use of a GP system to infer technical trading rules from past. In the same vein, Bhattacharyya, Pictet, and Zumbach (2002) used GP to discover trading decision models from high-frequency foreign exchange (FX) markets data. In other research, Bhattacharyya et al. (2002) used GA for mining financial ▶time-series to identify patterns, with the aim to discover trading decision models. In a different approach, Potvin, Soriano, and Vallée (2004) applied GP to automatically generate short-term trading rules on the stock markets, the authors used historical pricing and transaction volume data reported for 14 Canadian companies from the Toronto stock exchange market. Other approach called grammatical evolution (GE) (Brabazon & O'Neill, 2004) was applied to discover new technical trading rules, which can be used to trade foreign exchange markets. In that approach, each of the evolved programs represents a market trading system.

As it was mentioned earlier, EC techniques are able to generate a set of solutions for a single problem, this quality has been used to collect a set of results, with the aim of applying the most suitable solution according to the situation, for instance Lipinski (2004) analyzed high-frequency data, from the Paris Stock Exchange Market. In that model, stock market trading rules were combined into stock market trading experts, which defined the trading expertise. The author used a simple GA, a population-based incremental learning, the compact genetic algorithm, and the extended compact genetic algorithm to discover optimal trading experts in a specific situation, the author argues that the optimal solution depends on the specific situation on the stock market, which varies with time. EC plays an important role in the learning and continual adaptation to the changing environment.

Taking advantage of the EC's ability to generate multiple solutions, Garcia-Almanza and Tsang (2008) proposed an approach, called evolving comprehensible rules (ECR), to discover patterns in financial data sets to detect investment opportunities. ECR was designed to classify the minority class in imbalanced environments, which is particularly useful in financial forecasting because the number of profitable chances is scarce. The approach offers a range of solutions to suit the investor's risk guidelines and so, the user can choose

the best trade-off between miss-classification and false alarm costs according to the investor's requirements. Another approach proposed by Ghandar et al. (2008) was designed to generate trading rules, the authors implemented an adaptive computational intelligent system by using an evolutionary algorithm and a fuzzy logic rule base representation. The data to train the system was composed just by volume and price. The authors' objective was to create a system to generate rules for buy recommendations in dynamic market conditions. An analysis of the results was provided by applying the system for portfolio construction in historical data for companies listed as part of the MSCI Europe Index from 1990 to 2005. The results showed that their approach was able to generate trading rules that beat traditional, fixed rule-based strategies, as the price momentum and alpha portfolios, but this also beat the market index.

Given that EC can be used as an optimization technique, it has been combined with other approaches. As an instance, Chen, Wang, and Zhang (1999) used a genetic algorithm to determine the number of input variables and the number of hidden layers in an ANN for forecasting foreign exchange rates of the Dollar/Deutsche mark. Chen and Lu (1999) used GP to optimize an ANN, this approach is called evolutionary neural trees (ENT). The objective was to forecast the high-frequency stock returns of the Heng–Sheng stock index. Schoreels, Logan, and Garibaldi (2004) investigated the effectiveness of an agent based trading system. The system employs a simple GA to optimize the trading decisions for every agent, the knowledge was based on a range of technical indicators generating trading signals. In Dempster, Payne, Romahi, and Thompson (2001) the authors aim to detect buy and sell signals in the exchange (FX) markets. The authors analyzed and compare the performance of a GP combined with a reinforcement learning system to a simple linear program characterizing a ►Markov decision process and a heuristic in high-frequency (intraday) foreign exchange trading. The authors considered eight popular technical indicators used by intraday FX traders, Based on simple trend-indicators such as moving averages as well as more complex rules. From experimental results the authors found that all methods were able to create significant in-sample and out-of-sample profits when transaction costs are zero. The GP approach generated profits for nonzero transaction costs, although none of the methods produce significant profits at realistic transaction costs.

As it can be seen from the previous paragraphs, EC techniques allow representing the solutions using different structures, such as, decision trees (Potvin et al. (2004)), finite states automats, graphs, grammar (Brabazon & O'Neill, 2004), networks, binary vectors (Lipinski, 2004) among may others. In fact, this characteristic lets us to choose the best representation for the problem.

Portfolio Optimization

Portfolio optimization is probably the most important task in finance. The most relevant aspects in finance are involved in such task: the determination of the price, the estimation of the volatility, the correlation among stocks, etc. The portfolio selection problem can be described in a simple way as the problem of choosing the assets and the proportion of such assets in an investor's portfolio that wants to maximize his profits and minimize the risk.

As its name suggest, Portfolio Optimization is an optimization problem and EC has proven to be very useful in difficult (sometimes intractable) optimization problems. In (Maringer, 2005), the author explains extensively the portfolio optimization problem and the possible heuristic approaches, including ant systems (AS), memetic algorithms (MAs), GAs, and ESs.

Multi-objective evolutionary optimization is an important field within EC and the portfolio optimization problem is one its more important applications in finance. Being a multi-objective optimization problem, EC provides plenty of opportunities and different approaches can be used for the portfolio optimization problem. For example, Hassan and Clack (2008) use a multi-objective GP to approach this problem. In Diosan (2005), the author compares different multi-objective evolutionary algorithms for the portfolio optimization problem.

The number of papers on portfolio optimization using any form of EC techniques is huge and still growing. For example, in (Loraschi et al., 1995) the authors use distributed genetic algorithms to approach the portfolio optimization problem, whereas in (Loraschi and Tettamanzi, 1995) and (Streichert, Ulmer, and Zell, 2004) the authors use EAs.

Financial Markets

Financial markets are mechanisms where buyers and sellers exchange goods like bonds, gold, options, currencies, etc. Some examples of such markets are the New York Stock Exchange, the Chicago Mercantil Exchange, and the NASDAQ Stock Market.

Financial markets are essential for financial systems and for the overall economy. Such markets represent one of the most efficient ways to allocate financial resources into companies, due to the low transaction costs and the public information available for buyers and sellers. However, bubbles and crashes are recurrent phenomena which have enormous repercussions to global economy. In fact, nowadays we can see as never before that one single crash in one market could lead to a worldwide slump on most of the remaining stock markets. Crises in financial markets could affect other aspects of the (real) economy; for example, interest rates, inflation, unemployment, etc. This, in turn could cause even more instability on the financial markets as we have witnessed recently. Moreover, market crashes occur with an unpleasant higher frequency than is predicted by the standard economic theory.

One of the most important research issues in financial markets is the explanation of the process that determines the asset prices and as a result the rate of return. There are many models that can be used to explain such process, like the capital asset pricing model (CAPM), the arbitrage pricing theory (APT) or the black-scholes option pricing. Unfortunately, such models do not explain, as one would expect, the behavior of prices in real markets. The contradictions between the existing theory and the empirical properties of the stock market returns are one of the motivations for some researchers to develop and use different approaches to study financial markets. An additional aspect on the study of financial markets is the complexity of the analytical models of such markets. Financial markets are also very complex to analyze the wide variety of participants and their ever-changing nature. Previous to the development of some new simulation techniques, very important simplifying (unrealistic) assumptions had to be made in order to allow tractability of the theoretical models.

Behavioral finance, agent-based computational economics (ACE) (Tesfatsion, 2002) and computational finance (Tsang & Martinez-Jaramillo, 2004) have risen as alternative ways to overcome some of the problems of the analytical models. AI and in particular EC have been used in the past to study some financial and economic problems. However, the development of a well established community, known as the ACE community, facilitates the study of phenomena in financial markets that was not possible in the past. Within such community, a vast number of works and a different number of approaches are being produced by numbers in order to solve or gain more understanding of some economic and financial problems.

The influential work of Arthur, Holland, LeBaron, Palmer, & Talyer, (1997) and previously the development of the concept of bounded rationality (Arthur, 1991; Simon, 1982) changed the way in which we conceive and model the economic agents. This change in conception, modified dramatically the possibilities to study some economic phenomena and in particular the Financial Markets. The new models of economic agents have changed, there is no need any more of fully rational representative agents, there is no need of homogeneous expectations and information symmetry. Furthermore, the development of artificially adapted agents (Holland & Miller, 1991) gives to the economics science a way forward into the study of economic systems.

Agent-based financial markets of different characteristics have been developed for the study of such markets in the last decade since the influential Santa Fe Artificial Market. (The Santa Fe Artificial Stock Market is a simulated stock market developed at the Santa Fe Institute. Such market was developed by team of highly reputed researchers, among them John Holland, the inventor of genetic algorithms (Holland, 1975).) (Arthur et al., 1997). Some of them differ from the original Santa Fe market on the type of agents used like Chen and Yeh (2001), Gode and Sunder (1992), Martinez-Jaramillo and Tsang (2009b); on market mechanism like Bak, Paczuski, and Shubik (1997), Gode and Sunder (1992). Other markets borrow ideas from statistical mechanics like Levy, M., Levy, H., and Solomon (1994) and Lux (1998). Some important research has been done modelling stock markets inspired on the minority game. (The Minority Game was first proposed by Yi-Cheng Zhang and Damien Challet (1997) inspired by El Farol bar problem introduced by Brian Arthur (1994).) like

Challet, Marsili, and Zhang (2000). There are financial simulated markets in which several stocks are traded like in Cincotti, Ponta, and Raberto (2005). However, there are some criticisms to this approach like the problem of calibration, the numerous parameters needed for the simulation program, the complexity of the simulation, etc.

Although they all differ in the sort of assumptions made, methodology and tools; these markets share the same essence: the macro behavior of such market (usually the price) should emerge endogenously as a result of the micro-interactions of the (heterogeneous) market participants. This approach is in opposition with the traditional techniques being used in Economics and Finance.

One of the most crucial aspects on the modelling of financial markets is the modelling of the market participants also known as "agents". Unfortunatelly, for the sake of mathematical tractability, theoretical models assume that all the market participants can be modelled by a representative agent. The *representative agent* is a common, yet very strong, assumption in the modeling of financial markets. This concept has been the source of controversy and strong criticisms. For example, in Kirman (1992), the author criticizes the *representative individual* approach in economics. Moreover, Lux and Ausloos (2002) declare:

> Unfortunately, standard modelling practices in economics have rather tried to avoid heterogeneity and interaction of agents as far as possible. Instead, one often restricted attention to the thorough theoretical analysis of the decisions of one (or few) *representative* agents

In order to overcome the limitations of such an assumption, some researchers has opted for less orthodox techniques like GAs and GPs to model the participants in financial markets. Such evolutionary techniques have been widely used to model the agents' behaviour and adaptation in financial markets. In order to understand the different approaches of the variety of artificial (simulated) financial markets, it is useful to describe the different types of markets on the basis of the framework proposed in LeBaron (2001). In such work, LeBaron identifies the key design issues present in every artificial financial market and describes some of the most important works until then. The main design issues identified in LeBaron (2001) are:

- Agents
- Market mechanism
- Assets
- Learning
- Calibration
- Time

In addition to the description of the different approaches in artificial financial markets by using the above described framework, there is a fairly detailed extension of it in Grothmann (2002) that is worth looking at. In such work the basic design issues proposed in LeBaron (2001) are extended and detailed. For a more complete and detailed guide to the application of EC techniques in artificial financial markets, see Martinez-Jaramillo and Tsang (2009).

Option Pricing

Derivatives (See Hull, 2008 for an introduction.) are financial instruments whose main purpose is to hedge risk; however, they can also be used with speculation purposes with potentially negative effects on the financial health of the companies. Derivatives markets are having an important expansion in recent years; futures, forwards, swaps, and options are the best known types of derivatives. Having said so, option pricing is an extremely important task in finance. The Black–Scholes model for option pricing is the reference analytical model as it has an important theoretical framework behind it. However, in practice prices show that there is a departure from the prices obtained with such model. One possible reason that could explain such departure is the assumptions being made in such model (the assumption of constant volatility and the assumption that prices follow a geometric Brownian motion). This is why GP has been used as an alternative to perform option pricing in Chen, Yeh, and Lee (1998), Fan, Brabazon, O'Sullivan, and O'Neill (2007), Yin, Brabazon, and O'Sullivan (2007). Interestingly, not only GP has been used to perform option pricing but also ACO has been explored to approach this important problem in finance (Kumar, Thulasiram, & Thulasiraman, 2008).

Credit Scoring, Credit Rating, and Bankruptcy Prediction

Credit rating and credit scoring are two examples of financial problems that have been traditionally approached through statistical analyzes. Credit rating is an estimate of a corporation's worthiness to be given a credit and is generally expressed in terms of an ordinal value; whereas credit scoring is a technique used express the potential risk of lending money to a given consumer in terms of a probability measure. Both techniques are therefore similar in their ends but applied to different domains.

The seminal work in the field of credit scoring is that of Altman (1968), who proposed the application of linear discriminant analysis (Fisher, 1936) to a set of measurements known as financial ratios, i.e., indicators of a corporation's financial health, that are obtained from the corporation's financial statements. One of the main applications of Altmans' method, also known as *Z*-score, is bankruptcy prediction. Understandably, a series of improvements have been achieved by means of applying more powerful classifiers, such as decision trees, genetic programming, neural networks and support vector machines, among others. References that apply such techniques or that make a literature review of their application are Atiya (2001), Huang, Chen, and Wang (2007), Ong, Huang, and Tzeng (2005), Shin and Lee (2002), and Martens, Baesens, Gestel, and Vanthienen (2007).

Another method to evaluate credit worthiness is the one provided by specialized agencies. The so-called credit ratings are nothing more than ordinal values expressing the financial history, current assets, and liabilities of entities such as individuals, organizations, or even sovereign countries, such that they represent their risk of defaulting a loan. Although each rating agency uses its own methodology and scale these are usually not disclosed, nevertheless, on the academic realm, several superseding techniques to ordinal regression have been applied. For example, Huang, Chen, H., Hsu, Chen, W. H., and Wu (2004) and Paleologo, Elisseeff, and Antonini (2010) have proposed computationally oriented methods to solve this problem.

Related to bankruptcy prediction, NNs have been the standard selection apart from the traditional statistical methods (discriminant analysis, logit and probit models). Quintana, Saez, Mochon, and Isasi (2008) explore the feasibility of using the evolutionary nearest neighbor classifier algorithm (ENPC) suggested by (Fernández & Isasi, 2004) in the domain of early bankruptcy prediction. They assess its performance comparing it to six other alternatives, their results suggest that this algorithm might be considered as a good choice. Another relevant work is Turku, Back, Laitinen, Sere, and Wezel (1996) in which the authors compare discriminant analysis, logit analysis, and GAs for the selection of the independent variables used for the prediction model. Finally, in Lensberg, Eilifsen, and McKee (2006), the authors use GP to study bankruptcy in Norwegian companies and find acceptable accuracy in addition to information about the usefulness of the variables used for the prediction task.

Cross References

►Evolutionary Algorithms
►Evolutionary Computation in Economics
►Evolutionary Computational Techniques in Marketing
►Genetic Algorithms
►Genetic Programming

Recommended Reading

Allen, F., & Karjalainen, R. (1999). Using genetic algorithms to find technical trading rules. *Journal of Financial Economics, 51*, 245–271.

Altman, E. I. (1968). Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *Journal of Finance, 23*(4), 589–609.

Arthur, W. B. (1991). Learning and adaptive economic behavior. Designing economic agents that act like human agents: A behavioral approach to bounded rationality. *American Economic Review, 81*, 353–359.

Arthur, W. B. (1994). Inductive reasoning and bounded rationality: The El Farol problem. *American Economic Review, 84*, 406–411.

Arthur, W. B., Holland, J. H., LeBaron, B., Palmer, R. G., & Talyer, P. (1997). Asset pricing under endogenous expectations in an artificial stock market. In W. B. Arthur, S. Durlauf, & D. Lane (Eds.), *The economy as an evolving complex system II*. Reading, MA: Addison-Wesley.

Atiya, A. F. (2001). Bankruptcy prediction for credit risk using neural networks: A survey and new results. *IEEE Transactions on Neural Networks, 12*(4), 929–935.

Bak, P., Paczuski, M., & Shubik, M. (1997). Price variations in a stock market with many agents. *Physica A, 246*, 430–453.

Bhattacharyya, S., Pictet, O. V., & Zumbach, G. (2002). Knowledge-intensive genetic discovery in foreign exchange markets. *IEEE Transactions on evolutionary computation, 6*(2), 169–181.

Brabazon, A., & O'Neill, M. (2004). Evolving technical trading rules for spot foreign-exchange markets using grammatical evolution. *Computational Management Science, 1*(3), 311–327.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth, CA: Wadsworth International Group.

Challet, D., Marsili, M., & Zhang, Y. C. (2000). Modeling market mechanism with minority game. *Physica A, 276*, 284–315.

Challet, D., & Zhang, Y. C. (1997). Emergence of cooperation and organization in an evolutionary game. *Physica A, 246*, 407.

Chen, S. H., & Lu, C. F. (1999). Would evolutionary computation help in designs of artificial neural nets in forecasting financial time series? In *IEEE Proceedings of 1999 congress on evolutionary computation,* (pp. 275–280). New Jersey: IEEE Press.

Chen, S. H., Wang, H. S., & Zhang, B. T. (1999). Forecasting high-frequency financial time series with evolutionary neural trees: The case of hang-seng stock index. In H. R. Arabnia (Ed.), *Proceedings of the international conference on artificial intelligence, IC-AI 99* (Vol. 2, pp. 437–443). Las Vegas, NV: CSREA Press.

Chen, S. H., & Yeh, C. H. (2001). Evolving traders and the business school with genetic programming: A new architecture of the agent-based artificial stock market. *Journal of Economic Dynamics and Control, 25*(3–4), 363–393.

Chen, S. H., Yeh, C. H., & Lee, W. C. (1998). Option pricing with genetic programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, et al. (Eds.), *Proceedings of the third annual conference, genetic programming 1998:* (pp. 32–37). San Francisco: Morgan Kaufmann.

Cincotti, S., Ponta, L., & Raberto, M. (2005). A multi-assets artifcial stock market with zero-intelligence traders. In *WEHIA 2005*, Essex, UK.

Dempster, M. A. H., Payne, T. W., Romahi, Y., & Thompson, G. W. P. (2001). Computational learning techniques for intraday fx trading using popular technical indicators. *IEEE Transactions on Neural Networks, 12*, 744–754.

Diosan, L. (2005). A multi-objective evolutionary approach to the portfolio optimization problem. In *CIMCA '05: Proceedings of the international conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce Vol. 2 (CIMCA-IAWTIC'06)* (pp. 183–187). Washington DC: IEEE Computer Society.

Fan, K., Brabazon, A., O'Sullivan, C., & O'Neill, M. (2007). Option pricing model calibration using a real-valued quantum-inspired evolutionary algorithm. In *GECCO '07: Proceedings of the 9th annual conference on genetic and evolutionary computation* (pp. 1983–1990). New York: ACM.

Fernández, F., & Isasi, P. (2004). Evolutionary design of nearest prototype classifiers. *Journal of Heuristics, 10*(4), 431–454.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics, 7*, 179.

Garcia-Almanza, A. L., & Tsang, E. P. K. (2008). Evolving decision rules to predict investment opportunities. *International Journal of Automation and Computing, 5*(1), 22–31.

Ghandar, A., Michalewicz, Z., Schmidt, M., To, T. D., & Zurbrugg, R. (2008). Computational intelligence for evolving trading rules. *IEEE Transactions on Evolutionary Computation, 13*(1), 71–86.

Gode, D. K., & Sunder, S. (1992). *Allocative efficiency of markets with zero intelligence (z1) traders: Market as a partial substitute forindividual rationality*. GSIA working papers 1992-16, Carnegie Mellon University, Tepper School of Business.

Grothmann, R. (2002). *Multi-agent market modeling based on neural networks*. PhD thesis, University of Bremen, Germany.

Hassan, G., & Clack, C. D. (2008). Multiobjective robustness for portfolio optimization in volatile environments. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation* (pp. 1507–1514). New York: ACM.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Holland, J. H., & Miller, J. H. (1991). Artificial adaptive agents in economic theory. *The American Economic Review, 81*, 365–370.

Huang, C. L., Chen, M. C., & Wang, C. J. (2007). Credit scoring with a data mining approach based on support vector machines. *Expert Systems with Applications, 33*(4), 847–856.

Huang, Z., Chen, H., Hsu, C. J., Chen, W. H., & Wu, S. (2004). Credit rating analysis with support vector machines and neural networks: A market comparative study. *Decision Support Systems, 37*(4), 543–558.

Hull, J. (2008). *Options, futures and other derivatives*. *Prentice Hall Series in Finance*. New Jersey: Prentice Hall. 7th Edition.

Kirman, A. P. (1992). Whom or what does the representative individual represents? *The Journal of Economic Perspectives, 6*, 117–136.

Kumar, S., Thulasiram, R. K., & Thulasiraman, P. (2008). A bioinspired algorithm to price options. In *C3S2E '08: Proceedings of the 2008 C3S2E conference* (pp. 11–22). New York: ACM.

LeBaron, B. (2001). A builder's guide to agent based financial markets. *Quantitative Finance, 1*, 254–261.

Lensberg, T., Eilifsen, A., & McKee, T. E. (2006). Bankruptcy theory development and classification via genetic programming. *European Journal of Operational Research, 169*(2), 677–697; Feature cluster on scatter search methods for optimization.

Levy, M., Levy, H., & Solomon, S. (1994). A microscopic model of the stock market: cycles, booms and crashes. *Economics Letters, 45*, 103–111.

Lipinski, P. (2004). *Evolutionary data-mining methods in discovering stock market expertise from financial time series*. PhD thesis, University of Wroclaw, Wroclaw, Poland.

Loraschi, A., & Tettamanzi, A. (1995). An evolutionary algorithm for portfolio selection in a downside risk framework. *The European Journal of Finance*, 1994.

Loraschi, A., Tettamanzi, A., Tomassini, M., Svizzero, C., Scientifico, C., & Verda, P. (1995). Distributed genetic algorithms with an application to portfolio selection. In *Artificial neural nets and genetic* (pp. 384–387). Berlin: Springer.

Lux, T. (1998). The socio-economic dynamics of speculative markets: Interacting agents, chaos, and the fat tails of return distributions. *Journal of Economic Behavior and Organization, 33*, 143–165.

Lux, T., & Ausloos, M. (2002). Market fluctuations I: Scaling, multiscaling and their possible origins. In A. Bunde, J. Kropp, & H. J. Schellnhuber (Eds.), *Theories of disaster – scaling laws governing weather, body, and stock market dynamics* (pp. 373–409). Berlin: Springer.

Maringer, D. (2005). *Portfolio management with heuristic optimization. Advances in computational management science* (Vol. 8). Berlin: Springer.

Martens, D., Baesens, B., Gestel, T. V., & Vanthienen, J. (2007). Comprehensible credit scoring models using rule extraction from support vector machines. *European Journal of Operational Research, 183*(3), 1466–1476.

Martinez-Jaramillo, S., & Tsang, E. P. K. (2009a). *Evolutionary computation and artificial financial markets. Studies in computational intelligence* (Vol. 185, pp. 137–179). Berlin: Springer.

Martinez-Jaramillo, S., & Tsang, E. P. K. (2009b). An heterogeneous, endogenous and coevolutionary gp-based financial market. *IEEE Transactions on Evolutionary Computation, 13*, 33–55.

Ong, C. S., Huang, J. J., & Tzeng, G. H. (2005). Building credit scoring models using genetic programming. *Expert Systems with Applications, 29*(1), 41–47.

Paleologo, G., Elisseeff, A., & Antonini, G. (2010). Subagging for credit scoring models. *European Journal of Operational Research. 201*(2), 490–499.

Potvin, J. Y., Soriano, P., & Vallée, M. (2004). Generating trading rules on the stock markets with genetic programming. *Computers and Operations Research, 31*(7), 1033–1047.

Quintana, D., Saez, Y., Mochon, A., & Isasi, P. (2008). Early bankruptcy prediction using enpc. *Applied Intelligence, 29*(2), 157–161.

Schoreels, C., Logan, B., & Garibaldi, J. M. (2004). Agent based genetic algorithm employing financial technical analysis for making trading decisions using historical equity market data. In *IAT '04: Proceedings of the intelligent agent technology, IEEE/WIC/ACM international conference* (pp. 421–424). Washington, DC: IEEE Computer Society.

Shin, K. S., & Lee, Y. J. (2002). A genetic algorithm application in bankruptcy prediction modeling. *Expert Systems with Applications, 23*(3), 321–328.

Simon, H. A. (1982). *Models of bounded rationality* (Vol. 2). Cambridge, MA: MIT Press.

Streichert, F., Ulmer, H., & Zell, A. (2004). Evaluating a hybrid encoding and three crossover operators on the constrained portfolio selection problem. In *Proceedings of the 2004 congress on evolutionary computation* (pp. 932–939). New Jersey: IEEE Press.

Tesfatsion, L. (2002). Agent-based computational economics: Growing economies from the bottom up. *Artificial Life, 8*, 55–82.

Tsang, E. P. K., & Martinez-Jaramillo, S. (2004). Computational finance. In *IEEE computational intelligence society newsletter* (pp. 3–8). New Jersey: IEEE Press.

Tsang, E. P. K., Yung, P., & Li, J. (2004). Eddie-automation, a decision support tool for financial forecasting. *Journal of Decision Support Systems, Special Issue on Data Mining for Financial Decision Making, 37*(4), 559–565.

Turku, B. B., Back, B., Laitinen, T., Sere, K., & Wezel, M. V. (1996). Choosing bankruptcy predictors using discriminant analysis, logit analysis, and genetic algorithms. In *Proceedings of the first international meeting on artificial intelligence in accounting, finance and tax* (pp. 337–356). Huelva: Spain.

Yin, Z., Brabazon, A., & O'Sullivan, C. (2007). Adaptive genetic programming for option pricing. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on genetic and evolutionary computation* (pp. 2588–2594). New York: ACM.

Evolutionary Computational Techniques in Marketing

Alma Lilia García-Almanza, Biliana Alexandrova-Kabadjova,
Serafín Martínez-Jaramillo
Bank of Mexico, Mexico, D.F.

Definition

Evolutionary Computation (EC) in marketing is a field that uses evolutionary techniques to extract and gather useful patterns with the objective of designing marketing strategies and discovering products and services of superior value which satisfy the customers' necessities. Due to the fierce competition by some companies for attracting more customers and the necessity of innovation, it is common to find numerous marketing problems being approached by EC techniques.

Motivation and Background

The objective of marketing is to identify the customers' needs and desires in order to guide the entire organization to serve best by designing products, services, and programs which satisfy customers (Kotler & Armstrong, 1996). Nowadays, the market competition is very strong, since customers can choose from several alternatives. For that reason, marketing teams are facing the necessity of creating intelligent business strategies. Thus, new artificial intelligent approaches for marketing have emerged; especially, evolutionary algorithms have been used to solve a variety of marketing problems such as the design of more attractive products and services for consumers, the analysis of populations to target potential clients, the design of new marketing strategies, and more.

Applications

Nowadays, it is very easy to capture and store large sets of data. However, such data must be processed and analyzed in order to obtain useful information to make marketing decisions. Since EC techniques can be used to extract patterns from data, these have been used in marketing for multiple purposes. In order to illustrate

the application of EC in marketing, let us introduce some important works in this field.

Target potential clients

Bhattacharyya (2000) proposed a Genetic Algorithm (GA) in combination with a case-based reasoning system to predict customer purchasing behavior. The objective was to identify potential customers for a specific product or service. This approach was developed and tested with real cases from a worldwide insurance direct marketing company. An optimization mechanism was integrated into the classification system to select those customers who were most likely to acquire an insurance.

New Products design

As it was mentioned previously, one of the goals of marketing is to discover products of superior value and quality. To achieve this goal, Fruchter et al. (2006) resolved to design a product line rather than a single product. The authors argued that by offering a product line, the manufacturer can customize the products according to the necessities of different segments of the population, which would satisfy more customers. Since the amount of data about customer preferences was large, the optimization of the product line became very difficult. The authors used a GA to optimize the problem and the performance of the solutions was valued by measuring the manufacturer's profits. In the same vein, Liu and Ong (2008) used a GA to solve a problem of marketing segmentation, this approach was used to make strategy decisions for reaching effectively all customers. In other approach proposed by Balakrishnan and Jacob (1996), a GA was used to optimize the customer's preferences in new products' design. The authors explained that, to design a new product it is important to determine its attributes, such as color or shape. A study to gather the customers' preferences had to be carried out. Finally, a GA was used to select those attributes that satisfied a bigger number of customers.

Advertisement

Advertising is an important area of marketing; this is defined as the activity of attracting public attention to a product or business. Since personalized advertisement improves marketing efficiency, Kwon and Moon (2001) proposed a personalized prediction model to be used in email marketing. A circuit model combined with Genetic Programming (GP) was proposed to analyze customers' information. The result was a set of recommendation rules, which was tested over a general mass marketing. According to the authors, the model achieved a significant improvement in sales. In Naik, Mantrala, and Sawyer (1998), the authors used a GA combined with a Kalman filter procedure to determine the best media schedule for advertisement, which was constrained by a budget. This approach evaluated a large number of alternative media schedules to decide the best media planning solution.

Internet has become a very popular and convenient media to make businesses. Many products and services can be found easily in a very short time, increasing the competition between those providers. Since this kind of sales does not involve human interaction directly, it is essential to design new and better strategies to personalize the Web pages in order to contend in this media. As an instance, Abraham and Ramos (2003) proposed an ant clustering algorithm to discover Web usage patterns and a linear genetic program to analyze the visitor trends. The objective was to discover useful knowledge from interactions of the users with the Web. The knowledge was used to design adaptive Web sites, business and support services, personalization, network traffic flow analysis, and more.

According to Scanlon (2008), the company Staples used a software called IDDEA to redesign and relaunch its paper brand. This software was developed by Affinova Inc, and uses a GA to simulate the evolution of consumer markets where strong products survive and weak ones die out. The strongest possible design emerges after several generations. A panel of 750 consumers select their favorite options from each generation. The software analyze customers choices over multiple generations to identify preference patterns. Surveys include consumer profiles that comprised basic demographic information, customer beliefs and consumer habits. This allow them to understand how different designs attract different consumers. In another project, IDDEA was used to identify imagery and messaging that would be of interest to consumers. As can be seen from previous paragraphs, EC has been used to solve a variety of marketing problems.

Since EC is able to dcal with optimization, forecasting and data mining problems, among others, there is a great potential of usage in the field of marketing to optimize processes, to extract patterns of customers from large amount of data, to forecast purchasing tendencies, and many others.

Cross References

►Evolutionary Algorithms
►Evolutionary Computation in Economics
►Evolutionary Computation in Finance
►Genetic Algorithms
►Genetic Programming

Recommended Reading

Abraham, A., & Ramos, V. (2003). Web Usage mining using artificial ant colony clustering and linear genetic programming, *Genetic programming, Congress on Evolutionary Computation (CEC)*, IEEE, 2003, 1384–1391.

Balakrishnan, P. V. S., & Jacob, V. S. (1996). Genetic algorithms for product design. *Management Science, 42*(8), 1105–1117.

Bhattacharyya, S. (2000). Evolutionary algorithms in data mining: Multi-objective performance modeling for direct marketing. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 465–473). New York: ACM.

Fruchter, G.E., Fligler, A., Winer, R. S. (2006). Optimal product line design: A genetic algorithm approach to mitigate cannibalization. *Journal of Optimization Theory and Applications*, 131(2), (pp. 227-244), Springer Netherlands.

Kotler, P., & Armstrong, G. (1996), *Principles of marketing, 7* ed., Prentice Hall, Englewood Cliffs NJ.

Kwon, Y.-K., & Moon, B.-R. (2001). Personalized email marketing with a genetic programming circuit model. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen (Eds.), *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)* (pp. 1352–1358). San Francisco: Morgan Kaufmann.

Liu, H.-H., & Ong, C.-S. (2008). Variable selection in clustering for marketing segmentation using genetic algorithms. *Expert Systems Applications, 34*(1), 502–510.

Naik, P. A., Mantrala, M. K., & Sawyer, A. G. (1998). Planning media schedules in the presence of dynamic advertising quality. *Marketing Science, 17*(3), 214–235.

Scanlon, Jessie. "Staples' Evolution." BusinessWeek 29 Dec. 2008: 1-2. Web, http://www.businessweek.com/innovate/content/dec2008/id20081229_162381.htm

Evolutionary Computing

►Evolutionary Algorithms

Evolutionary Constructive Induction

►Evolutionary Feature Selection and Construction

Evolutionary Feature Selection

►Evolutionary Feature Selection and Construction

Evolutionary Feature Selection and Construction

Krzysztof Krawiec
Poznan University of Technology
Poznan, Poland

Synonyms

EFSC; Evolutionary constructive induction; Evolutionary feature selection; Evolutionary feature synthesis; Genetic attribute construction; Genetic feature selection

Definition

Evolutionary feature selection and construction (EFSC) is a bio-inspired methodology for explicit modification of input data of a learning system. It uses evolutionary computation (EC) to find a mapping from the original data representation space onto a *secondary* representation space. In evolutionary feature selection (EFS), that mapping consists in dropping off some of the features (►attributes) from the original representation, so the dimensionality of the resulting representation space is not greater than that of the original space. In evolutionary feature construction (EFC), evolutionary algorithm creates (synthesizes) new features (derived attributes) that complement and/or replace the original ones. Therefore, EFS may be considered as special case of EFC.

A typical EFSC algorithm maintains a population of solutions, each of them encoding a specific mapping. The best mapping found in evolutionary search becomes the data preprocessor for the classifier. Usually, EFSC takes place in training phase only, and the evolved mapping does not undergo further changes in the testing phase.

Though EFSC is technically a form of data preprocessing (see ►Data Preparation), some of its variants may as well involve an internal inductive process in the fitness function. Also, EFS and EFC may be considered as special cases of ►Feature Selection and ►Feature Construction, respectively. EFC is also partially inspired by ►Constructive Induction.

Motivation and Background

Real-world machine learning problems often involve a multitude of attributes, which individually have low informative content and cannot provide satisfactory performance of the learning system. This applies in particular to data-abundant domains like image analysis and signal processing. When faced with many low-quality attributes, induction algorithms tend to build classifiers that perform poorly in terms of classification accuracy. This problem may be alleviated by removing some features from the original representation space (*feature selection*) or introducing new features defined as informative expressions (arithmetic, logical, etc.) built of *multiple* attributes (*feature construction*).

Unfortunately, many learning algorithms lack the ability of discovering intricate dependencies between attributes, which is a necessary precondition for successful feature selection and construction. This gap is filled out by EFSC, which uses EC to get rid of superfluous attributes and to construct new features. To this extent, anticipated benefits from EFSC are similar to those of general ►Feature Selection and ►Feature Construction, and include reduced dimensionality of the input space, better predictive accuracy of the learning system, faster training and querying, and better readability of the acquired knowledge.

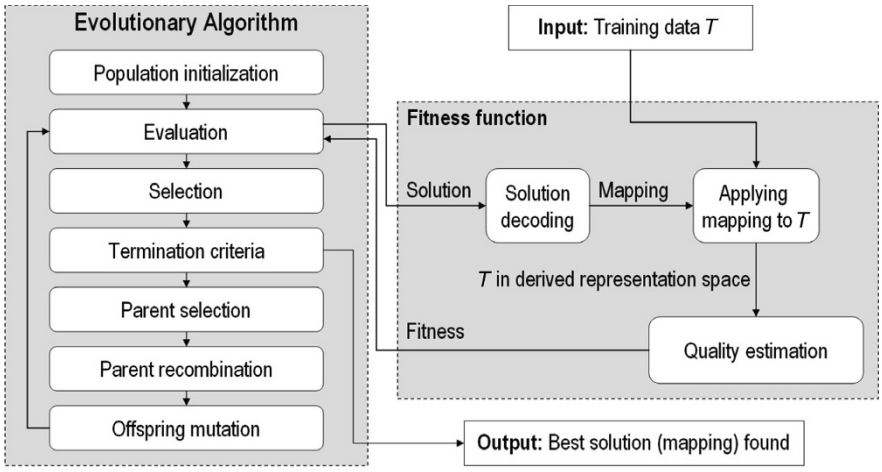
In general, both feature selection and feature construction may be conveniently formulated as an optimization problem with each solution corresponding to a particular feature subset (for feature selection) or to a particular definition of new features (for feature construction). The number of such solutions grows exponentially with the number of original features, which renders the exact search methods infeasible. Therefore, EC techniques with their ability of performing global parallel search with low risk of being trapped in local optima are particularly predisposed to solve these types of problems. Moreover, EC algorithms can optimize arbitrary function without demanding assumptions concerning solution space and objective function (like, for instance, the branch-and-bound algorithm). This is extremely important in the context of EFSC, where the so-called fitness landscape (the objective function spanned over the space of solutions) heavily depends on the training data, and it is therefore difficult to predict its properties.

The other strength of EC is the ease of adaptation to a specific task that usually boils down to the choice of solution representation and implementation of the fitness function. For instance, a subset of features in EFS may be directly encoded as a bit string solution in genetic algorithm (GA), where a bit at a particular position determines the selection or exclusion of the corresponding feature (Vafaie & Imam, 1994; Yang & Honavar, 1998). In EFC, definitions of constructed features may be conveniently represented as genetic programming (GP) expressions/procedures (Rizki, Zmuda, & Tamburino, 2002; Teller & Veloso, 1997). Also, unlike many other search algorithms, evolutionary algorithm can easily produce *many* solutions. This makes it a natural tool for, e.g., a parallel construction of multiple representations (feature subsets) that may be subsequently used in a compound classifier.

Structure of Learning System

Typically, EFSC uses a variant of evolutionary algorithm (usually GA for EFS or genetic programming for EFC) to maintain a population of solutions (individuals), each of them encoding a particular subset of features (for EFS) or definition of new features (for EFC). Solutions undergo mutations, crossing-over, and selective pressure that promotes the well-performing ones. Selective pressure is exerted by fitness function, which estimates solution's quality by measuring some properties of the secondary representation space (see Fig. 1). This usually involves three steps:

1. *Decoding* of solution (retrieving mapping from the encoded solution).
2. *Transforming* the training set into the secondary representation space according to the mapping.



Evolutionary Feature Selection and Construction. Figure 1. Evolutionary feature selection and construction

3. Estimating the *quality* of the secondary representation space which, after appropriate conversion (e.g., scaling), becomes solution's fitness.

Technically, step 3 usually boils down to one of two methods. *Filter approach* relies on measures that characterize the desired properties of training data in the secondary space (e.g., class separability), abstracting from any particular induction algorithm. *Wrapper approach* estimates the predictive ability that may be attained in the secondary representation space by a *specific* induction algorithm, usually by partitioning the training set into several subsets and performing multiple train-and-test experiment (e.g., cross-validation). In both cases, the particular implementation depends on the type of task being solved (classification or regression, predominantly the former one). Wrapper approach, though computationally more expensive, takes into account inductive and representational biases specific for induction algorithm, and often prove superior in terms of classification accuracy.

The result of a typical EFSC procedure is the best solution found in the evolutionary run, i.e., the superior representation mapping with respect to fitness function. This mapping serves as a preprocessor of input data and is subsequently used to induce the final classifier from the training set. The trained classifier together with the preprocessing provided by the mapping is the final outcome of the EFSC-enriched training process and may be used for classification of new examples.

EFS and EFC are predominantly applied to supervised learning from examples and attribute-value representation of training data. The above scheme remains relatively unchanged across various EFS and EFC approaches reported in literature, with main differences discussed in following.

Evolutionary Feature Selection

EFS is the simplest variant of EFSC. In this case, a solution encodes the indices of attributes that should be removed from the original representation (or, alternatively, which should be left in the resulting secondary representation). This leads to straightforward encoding characteristic for GA, with each solution being a bit string as long as the number of original attributes. EFS may be thus easily implemented using off-shelf generic software packages and involves relatively straightforward fitness function. However, more sophisticated approaches have been also considered, like evolving GP individuals to asses the quality of and rank feature subsets (Neshatian & Zhang, 2009).

Evolutionary feature weighting (EFW) is a direct generalization of EFS, where the evolutionary search weighs features instead of selecting them. Solutions in EFW are real-valued vectors evolved by evolutionary algorithm or evolutionary strategy. EFW requires use of a special wrapper fitness function that can take attribute weights into account. In (Komosiński & Krawiec, 2000), EFW has been used with a nearest neighbor-based wrapper fitness function to weigh features for a medical diagnosing problem.

Evolutionary Feature Construction

EFC requires sophisticated evolutionary representation of solutions to encode definitions of new features, and usually employs genetic programming for that purpose. Each GP solution encodes an expression tree that uses the original attributes and numeric constants as leaves (terminals), and functions from a predefined vocabulary as internal tree nodes (nonterminals). The value returned by such an expression when applied to an example is interpreted as the new feature. Function set usually encompasses simple arithmetics (typically $+$, $-$, $*$, $/$) and elementary functions (like *sin*, *cos*, *log*, *exp*). The evolved features replace or extend the original ones. As a single new feature is usually insufficient to provide satisfactory discriminative ability, it is common to encode several GP trees within each solution.

EFC may be conveniently adopted to image analysis or computer vision problems, or any other type of machine learning task that involves a large numbers of attributes. Commonly, an EFC algorithm evolves GP solutions that construct higher-level features from low-level image attributes (Krawiec & Bhanu, 2005) or implement advanced feature detectors (Howard, Roberts, & Ryan, 2006; Puente, 2009; Quintana, Poli, & Claridge, 2006). Alternatively, solutions encode chains of operations that process the entire image globally according to the goal specified by the fitness function. Many other variants of this approach have been studied in literature, involving, e.g., solutions represented as graphs (Teller & Veloso, 1997) or sequences of operations (linear genetic programming, (Bhanu et al., 2005)).

Applications

Real-world applications of EFSC are numerous and include medical and technical diagnosing, computer network intrusion detection, genetics, air quality forecasting, brain-computer interfaces, seismography, robotics, face recognition, handwriting recognition, vehicle detection in visual, infrared, and radar modality, image segmentation, satellite imaging, and stereovision. The conceptually simpler EFS has been implemented in several machine learning and neural-network software packages (WEKA, Statistica Neural Networks). EFC usually requires a more sophisticated and application-specific implementation. However, for standard learning-from-example tasks, it may be conveniently implemented by extending off-shelf libraries, like WEKA (Waikato Environment for Knowledge Analysis, http://www.cs.waikato.ac.nz/ml/weka/) and ECJ (Evolutionary Computation in Java, http://cs.gmu.edu/~eclab/projects/ecj/). More examples of real-world applications of EFSC may be found in (Langdon, Gustafson, & Koza, 2009).

Future Directions

Recent work on EFC employs various extensions of EC. It has been demonstrated that an EFC task may be decomposed into several semi-independent subtasks using cooperative coevolution, a variant of evolutionary algorithm that maintains several populations with solutions encoding partial solutions to the problem (Krawiec & Bhanu, 2005). Other recent work demonstrates that fragments of GP expressions encoding feature definitions may help to discover good features in other learning tasks (Jaśkowski, Krawiec, & Wieloch, 2007). With time, EFC becomes more and more unified with GP-based classification, where solutions are expected to perform the complete classification or regression task rather than to implement only feature definitions.

The online genetic programming bibliography (Langdon et al., 2009) provides quite complete coverage of state of the art in evolutionary feature selection and construction. A concise review of contemporary genetic programming research involving feature construction for image analysis and object detection may be found in (Krawiec, Howard, & Zhang, 2007). A more extensive and systematic study of different evolutionary approaches to feature construction is presented in (Bhanu et al., 2005).

Cross References

►Constructive Induction
►Data Preparation
►Feature Selection

Recommended Reading

Bhanu, B., Lin, Y., & Krawiec, K. (2005). *Evolutionary synthesis of pattern recognition systems*. New York: Springer-Verlag.

Howard, D., Roberts, S. C., & Ryan, C. (2006). Pragmatic genetic programming strategy for the problem of vehicle detection in

airborne reconnaissance. *Pattern Recognition Letters*, *27*(11), 1275–1288.

Jaśkowski, W., Krawiec, K., & Wieloch, B. (2007). Knowledge reuse in genetic programming applied to visual learning. In Dirk Thierens et al. (eds.), GECCO '07: In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, (Vol 2, pp. 1790–1797), London, 2007. ACM Press.

Komosiński, M., & Krawiec, K. (2000). Evolutionary weighting of image features for diagnosing of CNS tumors. *Artificial Intelligence in Medicine*, *19*(1), 25–38.

Krawiec, K., & Bhanu, B. (2005). Visual learning by coevolutionary feature synthesis. *IEEE Transactions on System, Man, and Cybernetics – Part B*, *35*(3), 409–425.

Krawiec, K., Howard, D., & Zhang, M. (2007). Overview of object detection and image analysis by means of genetic programming techniques. In *Proceedings of frontiers in the convergence of bioscience and information technologies 2007 (fbit2007), Jeju, Korea, october 11–13, 2007* (pp. 779–784). IEEE CS Press.

Langdon, W., Gustafson, S., & Koza, J. (2009). *The genetic programming bibliography.* ([online] http://www.cs.bham.ac.uk/ wbl/biblio/)

Neshatian, K., & Zhang, M. (2009). Genetic programming for feature subset ranking in binary classification problems. In L. Vanneschi, S. Gustafson, A. Moraglio, I. vanoe De Falco, & M. Ebner (Eds.), *Genetic programming* (pp. 121–132). Springer.

Puente, C., Olague, G., Smith, S. V., Bullock, S. H., González-Botello, M. A., & Hinojosa-Corona, A. (2009). A novel GP approach to synthesize vegetation indices for soil eros ion assessment. In M. Giacobini et al. (Eds.), *Applications of evolutionary computing* (pp. 375–384). Springer.

Quintana, M. I., Poli, R., & Claridge, E. (2006) Morphological algorithm design for binary images using genetic programming. *Genetic Programming and Evolvable Machines*, *7*(1), 81–102.

Rizki, M. M., Zmuda, M. A., & Tamburino, L. A. (2002). Evolving pattern recognition systems. *IEEE Transactions on Evolutionary Computation*, *6*(6), 594–609.

Teller, A., & Veloso, M. (1997). PADO: A new learning architecture for object recognition. In K. Ikeuchi & M. Veloso (Eds.), *Symbolic visual learning* (pp. 77–112). New York: Oxford Press.

Vafaie, H., & Imam, I. F. (1994). Feature selection methods: genetic algorithms vs. greedy-like search. In *Proceedings of international conference on fuzzy and intelligent control systems*.

Yang, J., & Honavar, V. (1998). Feature subset selection using a genetic algorithm. *IEEE Transactions on Intelligent Systems*, *13*(2), 44–49.

Evolutionary Feature Synthesis

►Evolutionary Feature Selection and Construction

Evolutionary Fuzzy Systems

Carlos Kavka
University of Trieste
Trieste
Italy

E

Definition

An evolutionary fuzzy system is a hybrid automatic learning approximation that integrates ►fuzzy systems with ►evolutionary algorithms, with the objective of combining the optimization and learning abilities of evolutionary algorithms together with the capabilities of fuzzy systems to deal with approximate knowledge. Evolutionary fuzzy systems allow the optimization of the knowledge provided by the expert in terms of linguistic variables and fuzzy rules, the generation of some of the components of fuzzy systems based on the partial information provided by the expert, and in some cases even the generation of fuzzy systems without expert information. Since many evolutionary fuzzy systems are based on the use of genetic algorithms, they are also known as *genetic fuzzy systems*. However, many models presented in the scientific literature also use genetic programming, evolutionary programming, or evolution strategies, making the term *evolutionary fuzzy systems* more adequate. Highly related is the concept of *evolutionary neuro-fuzzy systems*, where the main difference is that the representation is based on neural networks. Recently, the related concept of *evolving fuzzy systems* has been introduced, where the main objective is to apply evolutionary techniques to the design of fuzzy systems that are adequate to the control of nonstationary processes, mainly on real-time applications.

Motivation and Background

One of the most interesting properties of a fuzzy system is its ability to represent expert knowledge by using linguistic terms of everyday common use, allowing the description of uncertainty, vagueness, and imprecision in the expert knowledge. The linguistic terms, which are imprecise by their own nature, are, however, defined very precisely by using fuzzy theory concepts.

The usual approach to build a fuzzy system consists in the definition of the membership functions and the rule base in terms of expert knowledge. Compared with other rule-based approaches, the process of extracting knowledge from experts and representing it formally is simpler, since linguistic terms can be defined to match the terms used by the experts. In this way, rules are defined establishing relations between the input and output variables using these linguistic terms. However, even if there is a clear advantage of using the terms defined as ▶fuzzy sets, the knowledge extraction process is still difficult and time consuming, usually requiring a very difficult manual fine tuning process. It should be noted that no automatic framework to determine the parameters of the components of the fuzzy system exists yet, generating the need for methods that provide adaptability and learning ability for the design of fuzzy systems.

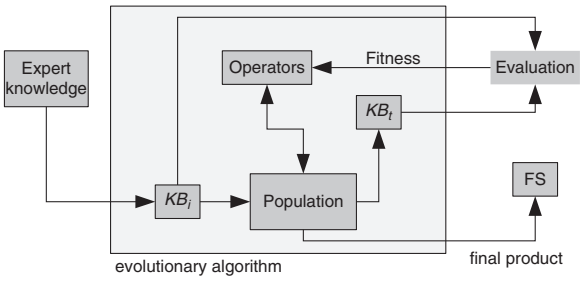
Since it is very easy to map a fuzzy system into a feedforward neural network structure, it is not surprising that many methods based on neural network learning have been proposed to automate the fuzzy system building process (Hoffmann, 2001; Karr & Gentry, 1993) The combined approach provides advantages from both worlds: the low level learning and computational power of neural networks is joined together with the high level human-like thinking and reasoning of fuzzy systems. However, this approach can still face some problems, such as the potential risk of its learning algorithms to get trapped in local minimum, the possible need for restriction of the membership functions to follow some mathematical properties (like differentiability), and the difficulties of inserting or extracting knowledge in some approaches, where the obtained linguistic terms can exhibit a poor semantic due to the usual black-box processing of many neural networks models.

Evolutionary algorithms provide a set of properties that make them ideal candidates for the optimization and design of fuzzy systems, and in fact, there are many methods that have been proposed in the literature to design or tune the different components of fuzzy systems. Evolutionary systems exhibit robust performance and global search characteristics, while requiring only a simple quality measure from the environment. There is no need for gradient information or input/output patterns. Other strengths come from its parallel nature: instead of selecting a single solution and refining it, in most evolutionary methods, a set of alternative solutions is considered and evolved in parallel.

Structure of the Learning System

The learning process defined by an evolutionary fuzzy system starts from the knowledge provided by the expert, which can include all or just some of the components of the knowledge base of a fuzzy system. The evolutionary algorithm that is behind this learning approach can perform the optimization of all the parameters that are provided by the expert, plus the generation of the missing components of the fuzzy system based on the partial specifications provided by the expert.

The model shown in Fig. 1 presents a general architecture of the learning and optimization process in evolutionary fuzzy systems. An initial knowledge base KB_i is built based on the knowledge provided by the expert. Note that KB_i could be (and usually is) a incompletely specified knowledge base. Based on this initial expert knowledge, the evolutionary algorithm creates a population of individuals, which can represent complete fuzzy systems or just a few components of them. The evaluation of the individuals is performed by creating a temporary knowledge base KB_t, which can also be complete or not. By using the information in KB_t, combined with the initial knowledge base KB_i, the individuals are evaluated by determining the error in the approximation of patterns if there are examples available, computing the reinforcement signal (typical situation in control problems), or in any other way depending on the problem characteristics (Babuska, 1998; Cordon, Gomide, Herrera, Hoffmann, & Magdalena, 2004). The



Evolutionary Fuzzy Systems. Figure 1. The general model of the evolutionary fuzzy systems learning and optimization

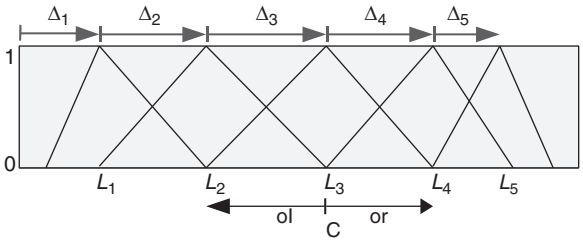
result of the evaluation is typically a single fitness measure, which provides the necessary information for the selection and the variational operators of the evolutionary algorithm. These operators, which can be standard or defined specifically for the problem, combine and mute the individuals based on the fitness value and their specific parameters. The process is repeated till a predefined criterion is fulfilled, obtaining as a final result the fuzzy system *FS*.

Depending on the information provided by the expert, the learning or optimization process performed by the evolutionary fuzzy system can be applied to the database, the fuzzy rule base or both of them. These three approaches are described below.

Optimization and Learning of the Fuzzy Database

In this case, it is assumed that the fuzzy rule base is known and provided by the expert. The initial knowledge base KB_i contains the fuzzy rule base, and if provided, the initial approximation of the parameters of antecedents and/or consequents. Since the expert has to define the rule base, and in order to do that, he/she needs to know the labels of the linguistic terms used for the antecedents and consequents, it is usual that the number of fuzzy sets is predefined and kept constant during the evolution.

The representation of the individuals contains only the parameters of the fuzzy sets associated to the input linguistic variables, and the fuzzy sets associated to the output variables in the case of a Mamdani fuzzy system, or the associated lineal approximators in the case of a Takagi-Sugeno fuzzy system. Other parameters could also be specified if necessary (scale factors, etc.). Usually, individuals are represented as a fixed length string that is defined as the concatenation of all parameters of the input and output fuzzy sets or approximators. Of course, the representation for the fuzzy sets depends on their particular class: for example, three values are required to represent triangular fuzzy sets, four values to represent trapezoidal fuzzy sets, and two for sigmoidal fuzzy sets. As an example, Fig. 2 shows that three values are necessary to represent a triangular fuzzy set: the center, the left width, and the right width, labeled as *c*, *ol*, and *od*, respectively. From this example, it can be seen that 15 values are required in order to represent the 5 fuzzy sets associated to this single linguistic variable.



Evolutionary Fuzzy Systems. Figure 2. A linguistic variable represented with five fuzzy sets

However, it is usual to apply fuzzy logic concepts (Zadeh, 1988) to simplify the representation, with the implied reduction in the search space, and also, to enhance the interpretability (Casillas, Cordon, Herrera, & Magdalena, 2003) of the resulting fuzzy system. As an example, it is desirable that the partition associated to a linguistic variable fulfills the completeness property, which establishes that for each point in the input domain, the summation of the membership values of all membership functions must be equal to 1. It is also desirable that the position of the fuzzy sets remains always the same during the evolution, for example in Fig. 2, it means that it is expected that the fuzzy set L_1 will be always at the left of L_2, L_2 always at the left of L_3, and so on. A representation that considers these two requirements can be defined by representing the whole partition specifying the distance from the center of a fuzzy set to the center of the next one (Hoffmann, 2001). The representation of five fuzzy sets then requires only five values (labeled in the figure as Δ_i), which reduces largely the search space and keeps the order of fuzzy sets, while fulfilling the completeness property. Most implementations use real values to represent the parameters.

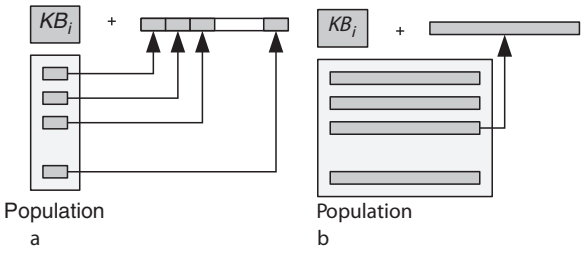
The operators of the evolutionary algorithm can be standard operators or can be defined specifically based on the selected representation. As an example, operators that modify the width of fuzzy sets, shift the centers, or perform other operations on the fuzzy set representations, linear approximators, or other parameters have been defined in the scientific literature.

Optimization and Learning of the Fuzzy Rule Base

In this case, the fuzzy rule base is not known, or only an initial approximation to it is provided. The other parameters of the knowledge base are known and

provided by the expert. The three most usual approximations are

1. Michigan approximation: Each individual of the population codifies a single rule (Bonarini, 1996), which means that each individual by itself cannot represent a complete solution to the problem. The knowledge base for evaluation KB_t is built based on the information defined in KB_i and the rules defined by all the individuals from the population combined together (see Fig. 3a). Rules are penalized or rewarded based on its performance during the evaluation. The fuzzy system is then built through the competition of a set of independent rules that have to be learned to collaborate during the evolution.
2. Pittsburgh approximation: Each individual represents the complete rule base. If dynamic creation and removal of rules is allowed, it is necessary to define special variational operators to deal with variable length individuals. Compared with the Michigan approach the evaluation is simpler, since by just combining each individual with KB_i it is possible to build KB_t for evaluation (see Fig. 3b). However, usually, the search space is larger when compared with the Michigan approach.
3. Iterative approximation: Each individual codifies a single rule (Cordon, Herrera, & Hoffmann, 2001) like in the Michigan approach. However, in each iteration of the algorithm, only the best rule is selected discarding all the others. This selection is based by considering the properties of the rule, such as for example, its covering degree on a set of examples. The algorithm is then competitive and not cooperative. It is usually necessary to apply algorithms to refine the fuzzy rule set obtained at the end of the evolutionary process, which can include operations, such as for example, the removal of similar rules.



Evolutionary Fuzzy Systems. Figure 3. The evaluation of individuals in the (a) Michigan and (b) Pittsburgh approaches

The representation in all of these approximations usually consists of individuals that contain references to the fuzzy sets already defined in KB_i. The representation of each individual can be a sequence of integers where each one is an index to the fuzzy sets associated to the corresponding linguistic variable. As an example, the fuzzy rule base could be represented as a matrix where each cell corresponds to the intersection of the input fuzzy sets, containing the index of the output fuzzy set associated to the rule. It is also possible to represent the fuzzy rule base as a decision table or simply as a list of rules. In these last two cases, the representation can have variable length, allowing to represent fuzzy rule sets with variable size.

The fitness calculation depends on the selected approximation. On a Pittsburgh approximation, the fitness corresponds to the evaluation of the complete fuzzy system on the corresponding problem. It is also possible to include in the fitness calculation other factors, such as for example, penalization for fuzzy rule bases that contains many rules or fuzzy rules with superposed application areas, etc. On a Michigan or Iterative model, the fitness indicates the degree of adequacy of the rule measured independently, considering also in the Michigan model its degree of cooperation with the other rules in the population.

The definition of the variational operators depends of course on the selected approximation. If the representation allows it, standard operators of crossover and mutation can be used. However, it can be convenient (or necessary) to define specific operators. As an example, variational operators can consider factors such as the time period since the rule has been used for the last time, its overall contribution to the final result, its performance when evaluated on the set of examples, etc.

Optimization and Learning of the Complete Knowledge Base

This case is a combination of the two models described before. The knowledge base KB_i contains the initial approximation to the definition of the antecedents and consequents, and the initial approximation to the fuzzy

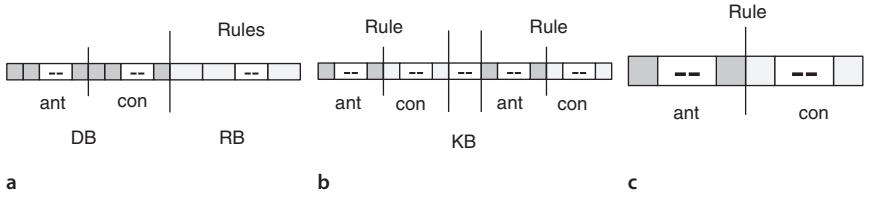
rule base as provided by the expert. Note that KB_i can also be empty if it is expected that the algorithm must generate all the parameters of the fuzzy system by itself.

The representation of the individuals contains all the parameters that define a knowledge base in order to allow its learning or optimization. The three most used representation schemes are shown in Fig. 4. In the first scheme, each individual contains the representation of all fuzzy sets, and the representation of all fuzzy rules using indexes to refer to the corresponding fuzzy sets. In the second scheme, each individual is structured as a set of rules, where each one specifies its own input and output fuzzy sets by directly including the parameters that define them. The representation (a) is adequate for descriptive fuzzy systems, since the rules contain references to the fuzzy sets used in their definition and can be shared by all of them. The representation (b) is adequate for approximative fuzzy systems, where each rule defines its own fuzzy sets. These two representations are adequate for the Pittsburgh approximation, while the third one (c) is adequate for the Michigan and the Iterative approximation. Of course, there can be many variations of this representations. For example, the input space partition can be predefined or obtained through fuzzy clustering algorithms, and if this partition is not expected to go under optimization, then it is not necessary to include the parameters of the input fuzzy sets in the representation.

Since this model is a combination of the two previous models, everything that was mentioned before concerning the fitness function and the variational operators also applies in this context. However, the fact that all parameters of the knowledge base are included in the representation allows to define more powerful variational operators. As an example, it is possible to define operators that decide the creation of new fuzzy sets, the elimination of some of them, and at the same time, the adaptation of the associated fuzzy rules, when for example, it is detected that there are areas in the input space that are not well covered, many rules with superimposed areas, etc. It is also possible to apply genetic programming techniques (Pedrycz, 2003), which are usually used to modify the structure of the fuzzy system, adding, removing, or combining sections of the fuzzy system with the objective of generating the most adequate structure.

Final Remarks

Clearly, the integration of fuzzy systems with evolutionary algorithms allows to overcome the limitations of each model considered independently, obtaining a powerful hybrid approach, which allows to learn and optimize fuzzy systems based on expert knowledge. Previous sections have discussed in general terms the evolutionary learning model. However, in order to get more details about particular implementations, it is recommended to read the publications referenced in the next section. The presentation from Karr & Gentry (1993) is interesting, not only because it provides a nice introduction and application of evolutionary fuzzy systems, but it has the additional value of being one of the first publications in the area. The presentation of Hoffmann (2001) is an excellent introduction to evolutionary fuzzy systems used for control applications. The other publications present details on evolutionary fuzzy systems (Babuska 1998; Bonarini 1996; Cordon et al., 2001; Juang Lin & Lin 2000; Lee & Takagi 1993), including representations based on neural networks (Hoffmann, 2001; Karr & Gentry, 1993), evolution strategies (Alpaydtn, Dundar, & Balktr, 2002), genetic programming (Pedrycz, 2003) and applications of evolutionary fuzzy systems to the domain of recurrent



Evolutionary Fuzzy Systems. Figure 4. Representations for the complete knowledge base adequate for (a) descriptive and (b) approximative fuzzy systems in the Pittsburgh approximation, and (c) representation of a single independent rule adequate for Michigan and Iterative approximations

fuzzy systems (Kavka, Roggero, & Schoenauer, 2005). The paper by Cordon et al. (2004) provides a very comprehensive reference list about the main developments on evolutionary fuzzy systems.

It should be stressed that a very important aspect to consider in the definition of evolutionary fuzzy systems is the interpretability of the resulting fuzzy systems (Casillas et al., 2003). Even if it has been mentioned that it is possible to design an evolutionary fuzzy system without expert information, by allowing the evolutionary algorithm to define all the components of the knowledge base by itself, it must always be considered that the interpretability of the results is essential. Designing a system that solves the problem, but that works as a black box, can be adequate in other contexts, but it is not desirable at all in the context of evolutionary fuzzy systems. An evolutionary fuzzy system algorithm must provide the means so that the expert knowledge defined in fuzzy terms can be considered and used appropriately during the evolution, and also, it must guarantee an adequate interpretability degree of the resulting fuzzy system.

Recommended Reading

Alpaydtn, G., Dundar, G., & Balktr, S. (2002). Evolution-based design of neural fuzzy networks using self-adapting genetic parameters. *IEEE Transactions of Fuzzy Systems, 10*(2), 211–221.

Babuska, R. (1998). *Fuzzy modeling for control*. Norwell, MA: Kluwer Academic Press.

Bonarini, A. (1996). Evolutionary learning of fuzzy rules: Competition and cooperation. In W. Pedrycz (Ed.), *Fuzzy modeling: Paradigms and practice*. Norwell, MA: Kluwer Academic Press.

Casillas, J., Cordon, O., Herrera, F., & Magdalena, L. (Eds.). (2003). *Interpretability issues in fuzzy modeling. Series: Studies in fuzziness and soft computing* (Vol. 128)

Cordon, O., Gomide, F., Herrera, F., Hoffmann, F., & Magdalena, L. (2004). Ten years of genetic fuzzy systems: Current framework and new trends. *Fuzzy Sets and Systems, 141*, 5–31.

Cordon, O., Herrera, F., & Hoffmann, F. (2001). *Genetic fuzzy systems*. Singapore: World Scientific Publishing.

Hoffmann, F. (2001). Evolutionary algorithms for fuzzy control system design. *Proceedings of the IEEE, 89*(9), 1318–1333.

Juang C. F., Lin, J. Y., & Lin, C. T. (2000). Genetic reinforcement learning through symbiotic evolution for fuzzy controller design. *IEEE Transactions on Systems, Man and Cybernetics, 30*(2), 290–302.

Karr, C. L., & Gentry, E. J. (1993). Fuzzy control of PH using genetic algorithms. *IEEE Transactions on Fuzzy Systems, 1*(1), 46–53.

Kavka, C., Roggero, P., & Schoenauer, M. (2005). Evolution of Voronoi based fuzzy recurrent controllers. In *Proceedings of GECCO* (pp. 1385–1392). NeW York: ACM Press.

Lee, M., & Takagi, H. (1993). Integrating design stages of fuzzy systems using genetic algorithms. In *Proceedings of the second IEEE international conference on fuzzy systems* (pp. 612–617).

Pedrycz, W. (2003). Evolutionary fuzzy modeling. *IEEE Transactions of Fuzzy Systems, 11*(5), 652–665.

Zadeh, L. (1988). Fuzzy logic. *IEEE Computer, 21*(4), 83–93.

Evolutionary Games

Moshe Sipper
Ben-Gurion University
Beer-Sheva, Israel

Definition

Evolutionary algorithms are a family of algorithms inspired by the workings of evolution by natural selection, whose basic structure is to

1. Produce an initial *population* of individuals, these latter being candidate solutions to the problem at hand
2. Evaluate the *fitness* of each individual in accordance with the problem whose solution is sought
3. *While* termination condition not met *do*
   a. *Select* fitter individuals for reproduction
   b. *Recombine* (*crossover*) individuals
   c. *Mutate* individuals
   d. *Evaluate* fitness of modified individuals
4. *End while*

Evolutionary games is the application of evolutionary algorithms to the evolution of game-playing strategies for various games, including chess, backgammon, and Robocode.

Motivation and Background

Ever since the dawn of artificial intelligence in the 1950s, games have been part and parcel of this lively field. In 1957, a year after the Dartmouth Conference that marked the official birth of AI, Alex Bernstein designed a program for the IBM 704 that played two amateur games of chess. In 1958, Allen Newell, J.C. Shaw, and Herbert Simon introduced a more sophisticated chess program (beaten in thirty-five moves by a ten-year-old beginner in its last official game played in 1960).

Arthur L. Samuel of IBM spent much of the 1950s working on game-playing AI programs, and by 1961 he had a checkers program that could play at the master's level. In 1961 and 1963, Donald Michie described a simple trial-and-error learning system for learning how to play Tic-Tac-Toe (or Noughts and Crosses) called MENACE (for Matchbox Educable Noughts and Crosses Engine). These are but examples of highly popular games that have been treated by AI researchers since the field's inception.

Why study games? This question was answered by Susan L. Epstein, who wrote:

> There are two principal reasons to continue to do research on games... First, human fascination with game playing is long-standing and pervasive. Anthropologists have cataloged popular games in almost every culture... Games intrigue us because they address important cognitive functions... The second reason to continue game-playing research is that some difficult games remain to be won, games that people play very well but computers do not. These games clarify what our current approach lacks. They set challenges for us to meet, and they promise ample rewards (Epstein, 1999).

Studying games may thus advance our knowledge in both cognition and artificial intelligence, and, last but not least, games possess a competitive angle which coincides with our human nature, thus motivating both researcher and student alike.

Even more strongly, Laird and van Lent proclaimed that,

> ...interactive computer games are the killer application for human-level AI. They are the application that will soon need human-level AI, and they can provide the environments for research on the right kinds of problems that lead to the type of the incremental and integrative research needed to achieve human-level AI (Laird & van Lent, 2000).

Recently, evolutionary algorithms have proven a powerful tool that can automatically "design" successful game-playing strategies for complex games (Azaria & Sipper, 2005a,b; Hauptman & Sipper, 2005b, 2007a,b; Shichel et al., 2005; Sipper et al., 2007).

Structure of the Learning System

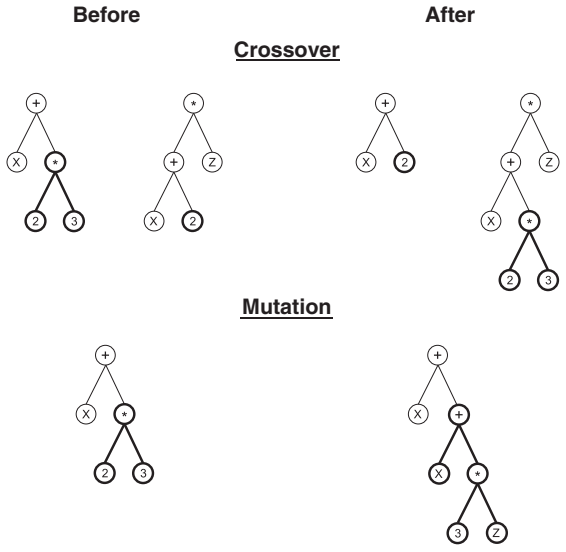
Genetic Programming

Genetic Programming is a subclass of evolutionary algorithms, wherein a *population* of individual programs is evolved, each program comprising *functions* and *terminals*. The functions are usually arithmetic and logic operators that receive a number of arguments as input and compute a result as output; the terminals are zero-argument functions that serve both as constants and as sensors, the latter being a special type of function that queries the domain environment.

The main mechanism behind genetic programming is precisely that of a generic evolutionary algorithm (Sipper, 2002; Tettamanzi & Tomassini, 2001), namely, the repeated cycling through four operations applied to the entire population: evaluate-select-crossover-mutate. Starting with an initial population of randomly generated programs, each individual is evaluated in the domain environment and assigned a *fitness* value representing how well the individual solves the problem at hand. Being randomly generated, the first-generation individuals usually exhibit poor performance. However, some individuals are better than others, that is, (as in nature) variability exists, and through the mechanism of natural (or, in our case, artificial) selection, these have a higher probability of being selected to parent the next generation. The size of the population is finite and usually constant.

Specifically, first a genetic operator is chosen at random; then, depending on the operator, one or two individuals are selected from the current population using a *selection operator*, one example of which is *tournament selection*: Randomly choose a small subset of individuals, and then select the one with the best fitness. After the probabilistic selection of better individuals the chosen genetic operator is used to construct the next generation. The most common operators are

- Reproduction (unary): Copy one individual to the next generation with no modifications. The main purpose of this operator is to preserve a small number of good individuals.
- Crossover (binary): Randomly select an internal node in each of the two individuals and swap the subtrees rooted at these nodes. An example is shown in Fig. 1.



Evolutionary Games. Figure 1. Genetic operators in genetic programming. LISP programs are depicted as trees. Crossover (*top*): Two subtrees (marked in *bold*) are selected from the parents and swapped. Mutation (*bottom*): A subtree (marked in *bold*) is selected from the parent individual and removed. A new subtree is grown instead

- Mutation (unary): Randomly select a node from the tree, delete the subtree rooted at that node, and then "grow" a new subtree in its stead. An example is shown in Fig. 1 (the growth operator as well as crossover and mutation are described in detail in Koza, 1992).

The generic genetic programming flowchart is shown in Fig. 2. When one wishes to employ genetic programming, one needs to define the following six desiderata:

1. Program architecture
2. Set of terminals
3. Set of functions
4. Fitness measure
5. Control parameters
6. Manner of designating result and terminating run

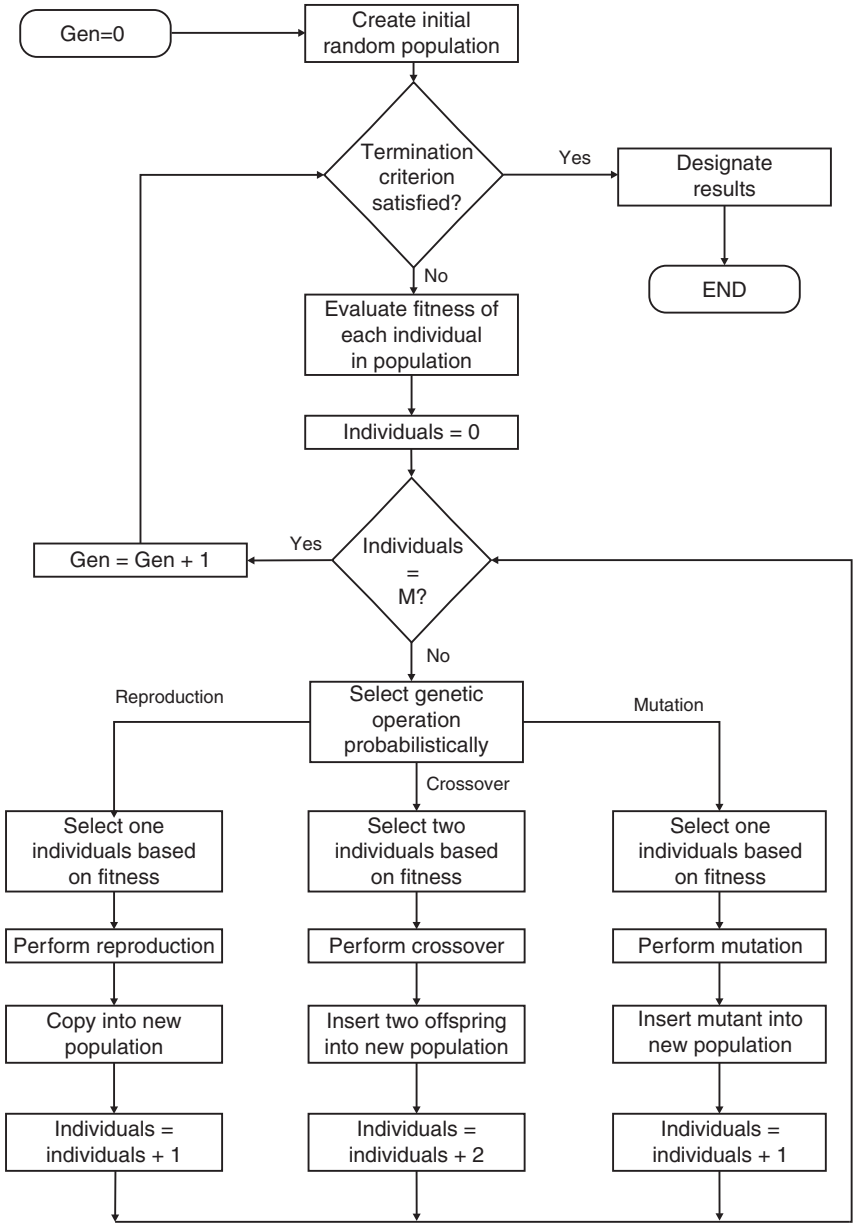
Evolving Game-Playing Strategies

Recently, we have shown that complex and successful game-playing strategies can be attained via genetic programming. We focused on three games (Azaria & Sipper, 2005a,b; Hauptman & Sipper, 2005b, 2007a,b; Shichel et al., 2005; Sipper et al., 2007):

1. *Backgammon.* Evolves a full-fledged player for the non-doubling-cube version of the game (Azaria & Sipper, 2005a,b; Sipper et al., 2007).
2. *Chess* (endgames). Evolves a player able to play endgames (Hauptman & Sipper, 2005b, 2007a,b; Sipper et al., 2007). While endgames typically contain but a few pieces, the problem of evaluation is still hard, as the pieces are usually free to move all over the board, resulting in complex game trees – both deep and with high branching factors. Indeed, in the chess lore much has been said and written about endgames.
3. *Robocode.* A simulation-based game in which robotic tanks fight to destruction in a closed arena (robocode.alphaworks.ibm.com). The programmers implement their robots in the Java programming language, and can test their creations either by using a graphical environment in which battles are held, or by submitting them to a central Web site where online tournaments regularly take place. Our goal here has been to evolve Robocode players able to rank high in the international league (Shichel et al., 2005; Sipper et al., 2007).

A strategy for a given player in a game is a way of specifying which choice the player is to make at every point in the game from the set of allowable choices at that point, given all the information that is available to the player at that point (Koza, 1992). The problem of discovering a strategy for playing a game can be viewed as one of seeking a computer program. Depending on the game, the program might take as input the entire history of past moves or just the current state of the game. The desired program then produces the next move as output. For some games one might evolve a complete strategy that addresses every situation tackled. This proved to work well with Robocode, which is a dynamic game, with relatively few parameters and little need for past history.

In a two-player game, such as chess or backgammon, players move in turn, each trying to win against the opponent according to specific rules (Hong, Huang, &



Evolutionary Games. Figure 2. Generic genetic programming flowchart (based on Koza, 1992). M is the population size, and Gen is the generation counter. The termination criterion can be the completion of a fixed number of generations or the discovery of a good-enough individual

Lin, 2001). The course of the game may be modeled using a structure known as an adversarial game tree (or simply game tree), in which nodes are the positions in the game and edges are the moves. By convention, the two players are denoted as MAX and MIN, where MAX is the player who moves first. Thus, all nodes at odd-numbered tree levels are game positions where MAX moves next (labeled MAX nodes). Similarly, nodes on even levels are called MIN nodes, and represent positions in which MIN (opponent) moves next.

The complete game tree for a given game is the tree starting at the initial position (the root) and containing all possible moves (edges) from each position. *Terminal nodes* represent positions where the rules of the game

determine whether the result is a win, a draw, or a loss. Although the game tree for the initial position is an explicit representation of all possible paths of the game, therefore theoretically containing all the information needed to play perfectly, for most (nontrivial) games it is extremely large, and constructing it is not feasible. For example, the complete chess game tree consists of roughly 10^{43} nodes (Shannon, 1950).

When the game tree is too large to be generated completely, only a partial tree (called a search tree) is generated instead. This is accomplished by invoking a *search algorithm*, deciding which nodes are to be developed at any given time and when to terminate the search (typically at nonterminal nodes due to time constraints). During the search, some nodes are evaluated by means of an *evaluation function* according to given heuristics. This is done mostly at the leaves of the tree. Furthermore, search can start from any position and not just at the beginning of the game.

Because we are searching for a winning strategy, we need to find a good next move for the current player, such that no matter what the opponent does thereafter, the player's chances of winning the game are as high as possible. A well-known method called the *minimax* search (Campbell & Marsland, 1983; Kaindl, 1988) has traditionally been used, and it forms the basis for most methods still in use today. This algorithm performs a depth-first search (the depth is usually predetermined), applying the evaluation function to the leaves of the tree, and propagating these values upward according to the minimax principal: at MAX nodes, select the maximal value, and at MIN nodes – the minimal value. The value is ultimately propagated to the position from which the search had started.

With games such as backgammon and chess one can couple a current-state evaluator (e.g., board evaluator) with a next-move generator. One can then go on to create a minimax tree, which consists of all possible moves, counter moves, counter countermoves, and so on; for real-life games, such a tree's size quickly becomes prohibitive. The approach we used with backgammon and chess is to derive a very shallow, single-level tree, and evolve "smart" evaluation functions. Our artificial player is thus created by combining an evolved board evaluator with a simple program that generates all next-move boards (such programs can easily be written for backgammon and chess).

In what follows, we describe the definition of the six items necessary in order to employ genetic programming, as delineated in the previous section: program architecture, set of terminals, set of functions, fitness measure, control parameters, and manner of designating result and terminating run. Due to lack of space we shall elaborate upon one game – Robocode – and only summarize the major results for backgammon and chess.

Example: Robocode

Program Architecture A Robocode player is written as an event-driven Java program. A main loop controls the tank activities, which can be interrupted on various occasions, called *events*. The program is limited to four lines of code, as we were aiming for the HaikuBot category, one of the divisions of the international league with a four-line code limit. The main loop contains one line of code that directs the robot to start turning the gun (and the mounted radar) to the right. This insures that within the first gun cycle, an enemy tank will be spotted by the radar, triggering a *ScannedRobotEvent*. Within the code for this event, three additional lines of code were added, each controlling a single actuator and using a single numerical input that was supplied by a genetic programming-evolved subprogram. The first line instructs the tank to move to a distance specified by the first evolved argument. The second line instructs the tank to turn to an azimuth specified by the second evolved argument. The third line instructs the gun (and radar) to turn to an azimuth specified by the third evolved argument (Fig. 3).

Terminal and Function Sets We divided the terminals into three groups according to their functionality (Shichel et al., 2005) (Fig. 4):

1. Game-status indicators: A set of terminals that provide real-time information on the game status, such as last enemy azimuth, current tank position, and energy levels.
2. Numerical constants: Two terminals, one providing the constant 0 and the other being an ephemeral random constant (ERC). This latter terminal is initialized to a random real numerical value in the range [-1, 1], and does not change during evolution.

Robocode Player's Code Layout

```
while (true)
    TurnGunRight(INFINITY); //main code loop
...
OnScannedRobot(){
    MoveTank(<GP#1>);
    TurnTankRight(<GP#2>);
    TurnGunRight(<GP#3>);
}
```

Evolutionary Games. Figure 3. Robocode player's code layout (HaikuBot division)

Energy()	Returns the remaining energy of the player
Heading()	Returns the current heading of the player
X()	Returns the current horizontal position of the player
Y()	Returns the current vertical position of the player
MaxX()	Returns the horizontal battlefield dimension
MaxY()	Returns the vertical battlefield dimension
EnemyBearing()	Returns the current enemy bearing, relative to the current player's heading
EnemyDistance()	Returns the current distance to the enemy
EnemyVelocity()	Returns the current enemy's velocity
EnemyHeading()	Returns the current enemy heading, relative to the current player's heading
EnemyEnergy()	Returns the remaining energy of the enemy
Constant()	An ERC (Ephemeral Random Constant) in the range [-1,1]
Random()	Returns a random real number in the range [-1,1]
Zero()	Returns the constant 0

a

Add(F, F)	Add two real numbers
Sub(F, F)	Subtract two real numbers
Mul(F, F)	Multiply two real numbers
Div(F, F)	Divide first argument by second, if denominator non-zero, otherwise return zero
Abs(F)	Absolute value
Neg(F)	Negative value
Sin(F)	Sine function
Cos(F)	Cosine function
ArcSin(F)	Arcsine function
ArcCos(F)	Arccosine function
IfGreater(F, F, F, F)	If first argument greater than second, return value of third argument, else return value of fourth argument
IfPositive(F, F, F)	If first argument is positive, return value of second argument, else return value of third argument
Fire(F)	If argument is positive, execute fire command with argument as fire-power and return 1; otherwise, do nothing and return 0

b

Evolutionary Games. Figure 4. Robocode representation. (a) Terminal set (b) Function set (F: Float)

3. Fire command: This special function is used to curtail one line of code by not implementing the fire actuator in a dedicated line.

Fitness Measure We explored two different modes of learning: using a fixed external opponent as teacher, and coevolution – letting the individuals play against each other; the former proved better. However, not just one but three external opponents were used to measure performance; these adversaries were downloaded from the HaikuBot league (robocode.yajags.com). The fitness value of an individual equals its average fractional score (over three battles).

Control Parameters and Run Termination The major evolutionary parameters (Koza, 1992) were population size – 256, generation count – between 100 and 200, selection method – tournament, reproduction probability – 0, crossover probability – 0.95, and mutation probability – 0.05. An evolutionary run terminates when fitness is observed to level off. Since the game is highly nondeterministic a "lucky" individual might attain a higher fitness value than better overall individuals. In order to obtain a more accurate measure for the evolved players, we let each of them do battle for 100 rounds against 12 different adversaries (one at a time). The results were used to extract the top player – to be submitted to the international league.

Results We submitted our top player to the HaikuBot division of the international league. At its very first tournament it came in third, later climbing to first place of 28 (robocode.yajags.com/20050625/haiku-1v1.html). All other 27 programs, defeated by our evolved strategy, were written by humans. For more details on GP-Robocode see Shichel et al., (2005) and Azaria, Hauptman, and Shichel (2007).

Backgammon and Chess: Major Results

Backgammon We pitted our top evolved backgammon players against *Pubeval*, a free, public-domain board evaluation function written by Tesauro. The program – which plays well – has become the *de facto* yardstick used by the growing community of backgammon-playing program developers. Our top evolved player was able to attain a win percentage of 62.4% in a tournament against Pubeval, about 10% higher (!) than the previous top method. Moreover, several evolved strategies were able to surpass the 60% mark, and most of them outdid all previous works. For more details on GP-Gammon, see Azaria and Sipper (2005a) and Azaria et al. (2007).

Chess (endgames) We pitted our top evolved chess-endgame players against two very strong external opponents: (1) A program we wrote ("Master") based upon consultation with several high-ranking chess players (the highest being Boris Gutkin, ELO 2400, International Master); (2) CRAFTY – a world-class chess program, which finished second in the 2004 World Computer Speed Chess Championship (www.cs.biu.ac.il/games/). Speed chess ("blitz") involves a time-limit per move, which we imposed both on CRAFTY and on our players. Not only did we thus seek to evolve good players, but ones who play well *and fast*. Results are shown in Table 1. As can be seen, GP-EndChess manages to hold its own, and even win, against these top players. For more details on GP-EndChess see Azaria et al., (2007) and Hauptman and Sipper (2005b).

Evolutionary Games. Table 1 Percent of wins, advantages, and draws for the best GP-EndChess player in the tournament against two top competitors

	%Wins	%Advs	%Draws
Master	6.00	2.00	68.00
CRAFTY	2.00	4.00	72.00

Deeper analysis of the strategies developed (Hauptman & Sipper, 2005a) revealed several important shortcomings, most of which stemmed from the fact that they used deep knowledge and little search (typically, they developed only *one* level of the search tree). Simply increasing the search depth would not solve the problem, since the evolved programs examine each board very thoroughly, and scanning many boards would increase time requirements prohibitively. And so we turned to evolution to find an optimal way to overcome this problem: How to add more search at the expense of less knowledgeable (and thus less time-consuming) node evaluators, while attaining better performance. In Hauptman and Sipper (2007b) *we evolved the search algorithm itself*, focusing on the *Mate-In-N* problem: find a key move such that even with the best possible counterplays, the opponent cannot avoid being mated in (or before) move N. We showed that our evolved search algorithms successfully solve several instances of the Mate-In-N problem, for the hardest ones developing 47% less game-tree nodes than CRAFTY. Improvement is thus not over the basic alpha-beta algorithm, but over a world-class program using all standard enhancements (Hauptman & Sipper, 2007b).

Finally, in Hauptman and Sipper (2007a), we examined a strong evolved chess-endgame player, focusing on the player's emergent capabilities and tactics in the context of a chess match. Using a number of methods we analyzed the evolved player's building blocks

and their effect on play level. We concluded that evolution has found combinations of building blocks that are far from trivial and cannot be explained through simple combination – thereby indicating the possible emergence of complex strategies.

Cross References

►Evolutionary Computation
►Genetic Algorithms
►Genetic Programming

Recommended Reading

Azaria, Y., & Sipper, M. (2005a). GP-Gammon: Genetically programming backgammon players. *Genetic Programming and Evolvable Machines, 6*(3), 283–300.

Azaria, Y., & Sipper, M. (2005b). GP-Gammon: Using genetic programming to evolve backgammon players. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, & M. Tomassini (Eds.), *Proceedings of 8th European conference on genetic programming (EuroGP2005), LNCS* (Vol. 3447, pp. 132–142). Heidelberg: Springer.

Campbell, M. S., & Marsland, T. A. (1983). A comparison of minimax tree search algorithms. *Artificial Intelligence, 20*, 347–367.

Epstein, S. L. (1999). Game playing: The next moves. In *Proceedings of the sixteenth National conference on artificial intelligence* (pp. 987–993). Menlo Park, CA: AAAI Press.

Hauptman, A., & Sipper, M. (2005a). Analyzing the intelligence of a genetically programmed chess player. In *Late breaking papers at the 2005 genetic and evolutionary computation conference*, GECCO 2005.

Hauptman, A., & Sipper, M. (2005b). GP-EndChess: Using genetic programming to evolve chess endgame players. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, & M. Tomassini (Eds.), *Proceedings of 8th European conference on genetic programming (EuroGP2005), LNCS* (Vol. 3447, pp. 120–131). Heidelberg: Springer.

Hauptman, A., & Sipper, M. (2007a). Emergence of complex strategies in the evolution of chess endgame players. *Advances in Complex Systems, 10*(Suppl. 1), 35–59.

Hauptman, A., & Sipper, M. (2007b). Evolution of an efficient search algorithm for the mate-in-N problem in chess. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, & A. I. Esparcia-Alcázar (Eds.), *Proceedings of 10th European conference on genetic programming (EuroGP2007), LNCS* (Vol. 4445, pp. 78–89). Heidelberg: Springer.

Hong, T.-P., Huang, K.-Y., & Lin, W.-Y. (2001). Adversarial search by evolutionary computation. *Evolutionary Computation, 9*(3), 371–385.

Kaindl, H. (1988). Minimaxing: Theory and practice. *AI-Magazine, 9*(3), 69–76.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Laird, J. E., & van Lent, M. (2000). Human-level AI's killer application: Interactive computer games. In *AAAI-00: Proceedings of the 17th National conference on artificial intelligence* (pp. 1171–1178). Cambridge, MA: MIT Press.

Shannon, C. E. (1950). Automatic chess player. *Scientific American, 48*, 182.

Shichel, Y., Ziserman, E., & Sipper, M. (2005). GP-Robocode: Using genetic programming to evolve robocode players. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, & M. Tomassini (Eds.), *Proceedings of 8th European conference on genetic programming (EuroGP2005), LNCS* (Vol. 3447, pp. 143–154). Heidelberg: Springer.

Sipper, M. (2002). *Machine nature: The coming age of bio-inspired computing*. New York: McGraw-Hill.

Sipper, M., Azaria, Y., Hauptman, A., & Shichel, Y. (2007). Designing an evolutionary strategizing machine for game playing and beyond. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 37*(4), 583–593.

Tettamanzi, A., & Tomassini, M. (2001). *Soft computing: Integrating evolutionary, neural, and fuzzy systems*. Berlin: Springer.

Evolutionary Grouping

►Evolutionary Clustering

Evolutionary Kernel Learning

Christian Igel
Ruhr-Universität Bochum
Bochum, Institute für Neuroinformatik Germany

Definition

Evolutionary kernel learning stands for using ►evolutionary algorithms to optimize the ►kernel function for a kernel-based learning machine.

Motivation and Background

In kernel-based learning algorithms the kernel function determines the scalar product and thereby the metric in the feature space in which the learning algorithm operates. The kernel is usually not adapted by the ►kernel method itself. Choosing the right kernel function is crucial for the training accuracy and generalization capabilities of the learning machine. It may also influence the runtime and storage complexity during learning and application.

Finding an appropriate kernel is a ►model selection problem. The kernel function is selected from an a

priori fixed class. When a parameterized family of kernel functions is considered, kernel adaptation reduces to finding an appropriate parameter vector. In practice, the most frequently used method to determine these values is grid search. In simple grid search the parameters are varied independently with a fixed step-size through a range of values and the performance of every combination is measured. Because of its computational complexity, grid search is only suitable for the adjustment of a few parameters. Further, the choice of the discretization of the search space may be crucial. Gradient-based approaches are perhaps the most elaborate techniques for adapting real-valued kernel parameters, see the articles by Chapelle, Vapnik, Bousquet, and Mukherjee (2002) and Glasmachers and Igel (2005) and references therein. To use these methods, however, the class of kernel functions must have a differentiable structure. They are also not directly applicable if the score function for assessing the parameter performance is not differentiable. This excludes some reasonable performance measures. Evolutionary kernel learning does not suffer from these limitations. Additionally, it allows for ►multi-objective optimization (MOO).

Structure of Learning System

Canonical evolutionary kernel learning can be described as an evolutionary algorithm (EA) in which the individuals encode kernel functions, see Fig. 1. These individuals are evaluated by determining the task-specific performance of the kernel they represent. Two special aspects must be considered when designing an EA for kernel learning. First, one must decide how to assess the performance (i.e., the fitness) of a particular kernel. That is, model selection criteria have to be defined depending on the problem at hand. Second, one must also specify the subset of possible kernel functions in which the EA should search. This leads to the questions of how to encode these kernels and which variation operators to employ.

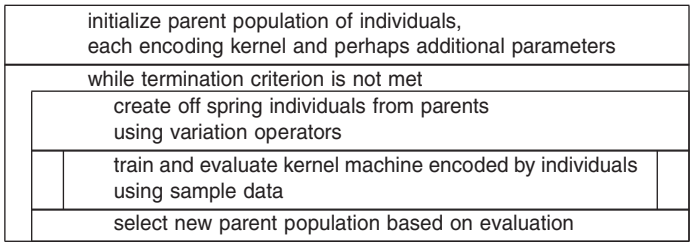
Assessing Fitness: Model Selection Criteria

The following presents some performance indices that have been considered for kernel selection. They can be used alone or in linear combination for single-objective optimization. In MOO a subset of these criteria can be used as different objectives.

It is important to note that, although many of these measures are designed to improve ►generalization, kernel learning can lead to ►overfitting if only limited data is used in the model selection process (e.g., in every generation the same small data sets are used to assess performance). Regularization (e.g., in a Bayesian framework) can be used to prevent overfitting. If enough data are available, it is advisable to monitor the generalization behavior of kernel learning using independent data. For example, external data can be used for the early stopping of evolutionary kernel learning.

Accuracy on Sample Data The most straightforward way to evaluate a model is to consider its performance on sample data. The empirical risk given by the error on the training data could be considered, but it does not measure generalization. To estimate the generalization performance, the accuracy on data not used for training is evaluated. In the simplest case, the available data is split into a training and validation set, with the first used for learning and the second for subsequent performance assessment. A theoretically sound and simple method is ►cross-validation (CV). Cross-validation makes better use of the data, but it is more computationally demanding. In practice, it yields very good results.

If ►classification is considered, it may be reasonable to split the classification error into false negative



Evolutionary Kernel Learning. Figure 1. Canonical evolutionary kernel learning algorithm

and false positive rates and to view ►sensitivity and ►specificity as two separate objectives (Suttorp & Igel, 2006).

Measures Derived from Bounds on the Generalization Performance Statistical learning theory allows one to compute estimates of and bounds on the expected generalization error of learning machines. These values can be utilized as criteria for model selection, although then the assumptions of the underlying theorems from statistical learning theory are typically violated and the terms "bound" and "unbiased estimate" become misleading.

For example, radius-margin bounds were used to evolve kernels for ►support vector machines (SVMs) for classification (Igel, 2005). Furthermore, the number of support vectors (SVs) was optimized in combination with the empirical risk (Igel, 2005). The fraction of SVs is an upper bound on the leave-one-out error (e.g., Chapelle et al., 2002).

Number of Input Variables Variable selection refers to the ►feature selection problem of choosing input variables that are best suited for the learning task. Masking a subset of variables can be viewed as modifying the kernel. By considering only a subset of feature dimensions the computational complexity of the learning machine decreases. When deteriorating feature dimensions are removed, the overall performance may increase. Reducing the number of input variables is therefore a common objective, which can be achieved using single-objective (Eads et al., 2002; Fröhlich, Chapelle, & Schölkopf, 2004; Jong, Marchiori, & van der Vaart, 2004; Miller, Jerebko, Malley, & Summers, 2003) or multi-objective (Pang & Kasabov, 2004; Shi, Suganthan, & Deb, 2004) evolutionary kernel learning.

Space and Time Complexity of the Classifier In some applications, it can be desirable to have fast kernel methods (e.g., for meeting real-time constraints). Thus, the execution time may be considered in the performance assessment during evolutionary kernel learning.

The space and time complexity of SVMs scales with the number of SVs. This is an additional reason to consider minimization of the number of SVs as an objective in evolutionary model selection for SVMs (Igel, 2005; Suttorp & Igel, 2006).

Multi-Objective Optimization The design of a learning machine is usually a MOO problem. For example, accuracy and complexity can be viewed as multiple, and probably conflicting, objectives. The goal of MOO is to approximate a diverse set of Pareto-optimal solutions (i.e., solutions that cannot be improved in one objective without getting worse in another one), which provide insights into the trade-offs between the objectives. Evolutionary multi-objective algorithms have become popular for MOO. Applications of multi-objective evolutionary kernel learning combining some of these performance measures listed above can be found in the work of Igel (2005), Pang and Kasabov (2004), and Shi et al. (2004).

Encoding and Variation Operators

The sheer complexity of the space of possible kernel functions makes it necessary to restrict the search to a particular class of kernel functions. This restriction essentially determines the representation and the operators used in evolutionary kernel learning.

When a parameterized family of mappings is considered, the kernel parameters can be encoded more or less directly in a real-valued EA. This is a frequently used representation, for example for *Gaussian kernel functions*.

For variable selection a binary encoding can be appropriate. One can fix a kernel $k : X \times X \to \mathbb{R}$, where $k(\vec{x}, \vec{z})$ solely depends on some distance measure between $\vec{x}, \vec{z} \in X$. In the binary encoding each bit then indicates whether a particular input variable is considered when computing the distance (Pang and Kasabov, 2004; Shi et al., 2004).

Kernels can be built from other kernels. For example, if k_1 and k_2 are kernel functions on X then $ak_1(\vec{x}, \vec{z}) + bk_2(\vec{x}, \vec{z})$ and $a \exp(-bk_1(\vec{x}, \vec{z}))$ for $\vec{x}, \vec{z} \in X, a, b \in \mathbb{R}^+$ are also kernels on X. This suggests a representation in which the individuals encode expressions that evaluate to kernel functions.

Given these different search spaces, it is not surprising that the aspects of all major branches of evolutionary computation have been used in evolutionary kernel learning: genetic algorithms (Fröhlich et al., 2004), genetic programming (Howley & Madden, 2005), evolution strategies (Igel, 2005), and evolutionary programming (Runarsson & Sigurdsson, 2004).

In general, kernel methods assume that the kernel (or at least the ►Gram matrix ►(kernel matrix) in the training process) is ►positive semidefinite (psd). Therefore, it is advisable to restrict the search space such that only psd functions evolve. Other ways of dealing with the problem of ensuring positive semidefiniteness are to ignore it (Howley & Madden, 2005) or to construct a psd Gram matrix from the matrix $\vec{M}$ induced by the training data and a non-psd "kernel" function. The latter can be achieved by subtracting the smallest eigenvalue of $\vec{M}$ from its diagonal entries.

Gaussian Kernels Gaussian kernel functions are prevalent. Their general form is $k(\vec{x},\vec{z}) := \exp\left(-(\vec{x}-\vec{z})^{\mathrm{T}}\vec{A}(\vec{x}-\vec{z})\right)$ for $\vec{x},\vec{z} \in \mathbb{R}^n$ and symmetric positive definite (pd) matrix $\vec{A} \in \mathbb{R}^{n\times n}$. When adapting $\vec{A}$, the issue of ensuring that the optimization algorithm generates only pd matrices $\vec{A}$ arises. This can be achieved by an appropriate parametrization of $\vec{A}$. Often the search is restricted to matrices of the form $\gamma\vec{I}$, where $\vec{I}$ is the unit matrix and $\gamma \in \mathbb{R}^+$ is the only adjustable parameter. However, allowing more flexibility has proven to be beneficial in certain applications (e.g., see Chapelle et al., 2002; Friedrichs & Igel, 2005; Glasmachers & Igel, 2005). It is straightforward to consider diagonal matrices with positive elements to allow for independent scaling factors weighting the input components. However, only by dropping this restriction one can achieve invariance against both rotation and scaling of the input space. A real-valued encoding that maps onto the set of all symmetric pd matrices can be used such that all modifications of the parameters result in feasible kernels, see the articles by Friedrichs and Igel (2005), Glasmachers and Igel (2005), and Suttorp and Igel (2006) for different parametrizations.

Optimizing Additional Hyperparameters One of the advantages of evolutionary kernel learning is that it can be easily augmented with an optimization of additional hyperparameters of the kernel method. The most prominent example is to encode not only the kernel but also the regularization parameter when doing model selection for SVMs.

Application Example

Notable applications of evolutionary kernel learning include the design of classifiers in bioinformatics (Mersch, Glasmachers, Meinicke, & Igel, 2007; Pang & Kasabov, 2004; Shi et al., 2004). Let us consider the work by Mersch et al. (2007) as an instructive example. Here, the parameters of a sequence kernel are evolved to improve the prediction of gene starts in DNA sequences. The kernel can be viewed as a weighted sum of 64 kernels, each measuring similarity with respect to a particular tri-nucleotide sequence (codon). The 64 weights $w_1,\ldots,w_{64}$ are optimized together with an additional global kernel parameter σ and a regularization parameter C for the SVM. Each individual stores $\vec{x} \in \mathbb{R}^{66}$, where $(w_1,\ldots,w_{64},\sigma,C)^{\mathrm{T}} = (\exp(x_1),\ldots,\exp(x_{64}),|x_{65}|,|x_{66}|)^{\mathrm{T}}$. An evolution strategy is applied, using additive multi-variate Gaussian mutation and weighted global recombination for variation and rank-based selection. The fitness is determined by a 5-fold cross-validation. The evolved kernels lead to higher classification rates and the adapted weights reveal the importance of particular codons for the task at hand.

Cross References

►Neuroevolution

Recommended Reading

Chapelle, O., Vapnik, V., Bousquet, O., & Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine Learning, 46*(1), 131–159.

Eads, D. R., Hill, D., Davis, S., Perkins, S. J., Ma, J., Porter, R. B., et al. (2002). Genetic algorithms and support vector machines for time series classification. In B. Bosacchi, D. B. Fogel, & J. C. Bezdek (Eds.), *Applications and science of neural networks, fuzzy systems, and evolutionary computation V*, Proceedings of the SPIE (Vol. 4787) (pp. 74–85). SPIE–The International Society for Optical Engineering. Bellington, WA

Friedrichs, F., & Igel, C. (2005). Evolutionary tuning of multiple SVM parameters. *Neurocomputing, 64*(C), 107–117.

Fröhlich, H., Chapelle, O., & Schölkopf, B. (2004). Feature selection for support vector machines using genetic algorithms. *International Journal on Artificial Intelligence Tools, 13*(4), 791–800.

Glasmachers, T., & Igel, C. (2005). Gradient-based adaptation of general gaussian kernels. *Neural Computation, 17*(10), 2099–2105.

Howley, T., & Madden, M. (2005). The genetic kernel support vector machine: Description and evaluation. *Artificial Intelligence Review, 24*(3), 379–395.

Igel, C. (2005). Multi-objective model selection for support vector machines. In C. A. Coello Coello, E. Zitzler, & A. Hernandez

Aguirre (Eds.), *Proceedings of the third international conference on evolutionary multi-criterion optimization (EMO 2005)*, LNCS (Vol. 3410) (pp. 534–546). Berlin: Springer.

Jong, K., Marchiori, E., & van der Vaart, A. (2004). Analysis of proteomic pattern data for cancer detection. In G. R. Raidl, S. Cagnoni, J. Branke, D. W. Corne, R. Drechsler, Y. Jin, et al. (Eds.), *Applications of evolutionary computing*, LNCS (Vol. 3005, pp. 41–51). Berlin: Springer.

Mersch, B., Glasmachers, T., Meinicke, P., & Igel, C. (2007). Evolutionary optimization of sequence kernels for detection of bacterial gene starts. *International Journal of Neural Systems, 17*(5), 369–381.

Miller, M. T., Jerebko, A. K., Malley, J. D., & Summers, R. M. (2003). Feature selection for computer-aided polyp detection using genetic algorithms. In A. V. Clough & A. A. Amini (Eds.), *Medical imaging 2003: Physiology and function: Methods, systems, and applications*, Proceedings of the SPIE (Vol. 5031) (pp. 102–110).

Pang, S., & Kasabov, N. (2004). Inductive vs. transductive inference, global vs. local models: SVM, TSVM, and SVMT for gene expression classification problems. In *International joint conference on neural networks (IJCNN 2004)* (Vol. 2, pp. 1197–1202). Washington, DC: IEEE Press.

Runarsson, T. P., & Sigurdsson, S. (2004). Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing – Letters and Reviews, 3*(3), 59–68.

Shi, S. Y. M., Suganthan, P. N., & Deb, K. (2004). Multi-class protein fold recognition using multi-objective evolutionary algorithms. In *IEEE symposium on computational intelligence in bioinformatics and computational biology* (pp. 61–66). Washington, DC: IEEE Press.

Suttorp, T., & Igel, C. (2006). Multi-objective optimization of support vector machines. In Y. Jin (Ed.), *Multi-objective machine learning*. Studies in computational intelligence (Vol. 16, pp. 199–220). Berlin: Springer.

Evolutionary Robotics

Phil Husbands
University of Sussex
Brighton, UK

Synonyms

Embodied evolutionary learning; Evolution of agent behaviors; Evolution of robot control

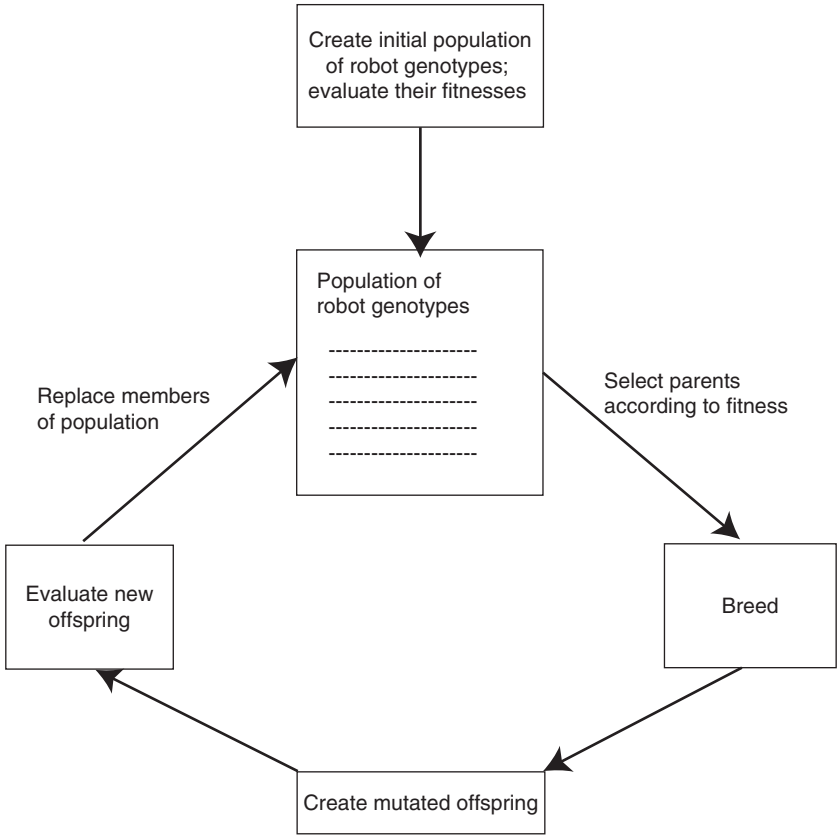
Definition

Evolutionary robotics involves the use of ▶evolutionary computing techniques to automatically develop some or all of the following properties of a robot: the control system, the body morphology, and the sensor and motor properties and layout. Populations of artificial genomes (usually lists of characters and numbers) encode properties of autonomous mobile robots required to carry out a particular task or to exhibit some set of behaviors. The genomes are mutated and interbred creating new generations of robots according to a Darwinian scheme in which the fittest individuals are most likely to produce offspring. Fitness is measured in terms of how good a robot's behavior is according to some evaluation criteria; this is usually automatically measured but may, in the manner of eighteenth century pig breeders, be based on the experimenters' judgment.

Motivation and Background

Turing's (1950) paper, *Computing Machinery and Intelligence*, is widely regarded as one of the seminal works in artificial intelligence. It is best known for what came to be called the Turing test – a proposal for deciding whether or not a machine is intelligent. However, tucked away toward the end of Turing's wide ranging discussion of issues arising from the test is a far more interesting proposal. He suggests that worthwhile intelligent machines should be adaptive, should learn and develop, but concedes that designing, building, and programming such machines by hand is probably completely infeasible. He goes on to sketch an alternative way of creating machines based on an artificial analog of biological evolution. Each machine would have hereditary material encoding its structure, mutated copies of which would form offspring machines. A selection mechanism would be used to favor better adapted machines – in this case, those that learned to behave most intelligently. Turing proposed that the selection mechanism should largely consist of the experimenter's judgment.

It was not until more than 40 years after their publication that Turing's long forgotten suggestions became reality. Building on the development of principled evolutionary search algorithm by, among others, Holland (1975), researchers at CNR, Rome, Case Western University, the University of Sussex, EPFL, and elsewhere independently demonstrated methodologies and practical techniques to evolve, rather than design, the control systems for primitive autonomous intelligent machines (Beer & Gallagher, 1992; Cliff, Harvey, & Husbands, 1993; de Garis, 1990; Floreano & Mondada,



Evolutionary Robotics. Figure 1. General scheme employed in evolutionary robotics

1994; Husbands & Harvey, 1992; Parisi & Nolfi, 1993). Thus, the field of *Evolutionary Robotics* was born in the early 1990s. Initial motivations were similar to Turing's: the hand design of intelligent adaptive machines intended for operation in natural environments is extremely difficult, would it be possible to wholly or partly automate the process?

Today, the field of evolutionary robotics has expanded in scope to take in a wide range of applications, including promising new work on autonomous flying machines (Floreano, Husbands, & Nolfi, 2008), as well as research aimed at exploring specific scientific issues – for instance, principles from neuroscience or questions in cognitive science (Harvey, Di Paolo, Wood, Quinn, & Tuci, 2005; Philippides, Husbands, Smith, & O'Shea, 2005). Such work is able to exploit the fact that evolutionary robotics operates with fewer assumptions about neural architectures and behavior generating mechanisms than other methods; this means that whole general classes of designs and processes can be explored.

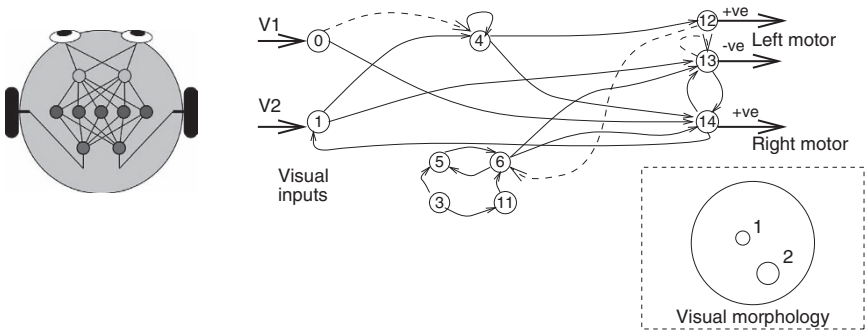
Structure of the Learning System

The key elements of the evolutionary robotics approach are

- An artificial genetic encoding specifying the robot control systems/body plan/sensor properties etc., along with a mapping to the target system
- A method for measuring the fitness of the robot behaviors generated from these genotypes
- A way of applying selection and a set of "genetic" operators to produce the next generation from the current

The structure of the overall evolutionary process is captured in Fig. 1. The general scheme is like that of any application of an evolutionary search algorithm. However, many details of specific parts of the process, particularly the evaluation step, are peculiar to evolutionary robotics.

The more general parts of the evolutionary process (selection, breeding, genetic operators such as mutation and crossover, replacement, and population structure)



Evolutionary Robotics. Figure 2. Evolved neurocontrollers. On the left a simple fixed architecture feedforward network is illustrated. The connection weights, and sometimes the neuron properties, are put under evolutionary control. On the right a more complex architecture is illustrated. In this case, the whole architecture, including the number of neurons and connections, is under evolutionary control, along with connection and neuron properties and the morphology of a visual sensor that feeds into the network

are also found in most other applications of evolutionary computing and, just as in those other applications, there are many well-documented ways of implemented each (De Jong, 2006; Eiben & Smith, 2003). Hence, this section focuses on genetic encoding and evaluation as a route to more evolutionary robotics specific issues. For a much fuller treatment of the subject, see Floreano et al. (2008) and Nolfi and Floreano (2000).

Genetic Encoding

While, as already mentioned, many aspects of the robot design can potentially be under genetic control, *at least* the control system always is. By far the most popular form of controller is some sort of neural network. These range from straightforward feedforward networks of simple elements (Floreano & Mondada, 1994) to relatively complex, dynamic and plastic recurrent networks (Beer & Gallagher 1992; Floreano & Urzelai 2000; Philippides, Husbands, Smith, & O'Shea, 2005), as illustrated in Fig. 2. In the simplest case, a fixed architecture network is used to control a robot whose sensors feed into the network which in turn feeds out to the robot motors. In this scenario, the parameters of the network (connection weights and relevant properties of the units such as thresholds or biases) are coded as a fixed length string of numerical values.

A more complex case, which has been explored since the very early days of evolutionary robotics (Cliff et al., 1993), involves the evolution of the network architecture as well as the properties of the connections and units. Typically, the size of the network (number of units and connections) and its architecture (wiring diagram) are unconstrained and free to evolve. This involves more complex encodings which can grow and shrink, as units and connections are added or lost, while allowing a coherent decoding of connections between units. These range from relatively simple strings employing blocks of symbols that encode a unit's properties and connections relative to other units (Cliff et al.) to more indirect schemes that make use of growth processes in some geometric space (Philippides et al., 2005) or employ genetic programming-like tree representations in which whole subbranches can be added, deleted, or swapped over (Gruau, 1995).

The most general case involves the encoding of control network and body and sensor properties. Various kinds of developmental schemes have been used to encode the construction of body morphologies from basic building blocks, both in simulation and in the real world. The position and properties of sensors can also be put under evolutionary control. Sometimes one complex encoding scheme is used for all aspects of the robot under evolutionary control, and sometimes the different aspects are put on separate genotypes.

Fitness Evaluation

The fitness of members of the population is measured, via an evaluation mechanism, in terms of the robot behaviors produced by the control system, or control system plus robot morphology that it encodes. Fitness evaluation, therefore, consists of translating the genome in question into a robot instantiation and then measuring the aspects of the resulting behavior. In the earliest

work aimed at using evolutionary techniques to develop neurocontrollers for particular physical robots, members of a population were downloaded in turn onto the robot and their behavior was monitored and measured either automatically by clever experimental setups (Floreano & Mondada, 1994; Harvey, Husbands, & Cliff, 1994) or manually by an observer (Gruau & Quatramaran, 1997). The machinery of the evolutionary search algorithm was managed on a host computer while the fitness evaluations were undertaken on the target robot.

One drawback of evaluating fitness on the robot is that this cannot be done any quicker than in real time, making the whole evolutionary process rather slow. However, in the early work in the field this approach was taken because it was felt that it was unlikely that simulations could be made accurate enough to allow proper transfer of evolved behavior onto the real robot. However, a careful study of accurate physics-based simulations of a Khepera robot, with various degrees of noise added, proved this assumption false (Jakobi, Husbands, & Harvey, 1995). This led to the development of Jakobi's minimal simulation methodology (Jakobi, 1998a), whereby computationally very efficient simulations are built by modeling only those aspects of the robot–environment interaction deemed important to the desired behavior and masking everything else with carefully structured noise (so that evolution could not come to rely on any of those features). These ultrafast, ultralean simulations have successfully been used with many different forms of robot and sensing, with very accurate transfer of behavior from simulation to reality. An alternative approach uses plastic controllers that further adapt through self-organization to help smooth out the differences between an inaccurate simulation and the real world (Urzelai & Floreano, 2001). Instead of evolving connection weights, in this approach "learning rules" for adapting connection strengths are evolved – this results in controllers that continually adapt to changes in their environment. For details of further approaches, see Floreano et al. (2008). Much evolutionary robotics work now makes use of simulations; without them it would be impossible to do the most ambitious work on the concurrent evolution of controllers and body morphology (Lipson & Pollack, 2000) (to be briefly described later). However, although simulation packages and techniques have developed rapidly in the past few years, there will still inevitably be discrepancies between simulation and reality, and the lessons and insights of the work outlined above should not be forgotten.

An interesting distinction can be made between implicit and explicit fitness functions in evolutionary robotics (Nolfi & Floreano, 2000). In this context, an explicit fitness function rewards specific behavioral elements – such as traveling in a straight line – and hence shapes the overall behavior from a set of specific behavioral primitives. Implicit fitness functions operate at a more indirect, abstract level – fitness points are given for completing some task but they are not tied to specific behavioral elements. Implicit fitness functions might involve components such as maintaining energy levels or covering as much ground as possible, components that can be achieved in many different ways. In practice, it is quite possible to define a fitness function that has both explicit and implicit elements.

Advantages

Potential advantages of this methodology include

- The ability to explore potentially unconstrained designs that have large numbers of free variables. A *class* of robot systems (to be searched) is defined rather than specific, fully defined robot designs. This means fewer assumptions and constraints are necessary in specifying a viable solution.
- The ability to use the methodology to fine-tune the parameters of an already successful design.
- The ability, through the careful design of fitness criteria and selection techniques, to take into account multiple, and potentially conflicting, design criteria and constraints (e.g., efficiency, cost, weight, power consumption, etc.).
- The possibility of developing highly unconventional and minimal designs.
- The ability to explicitly take into account robustness and reliability as major driving force behind the fitness measure, factors that are particularly important for certain applications.

Applications

For a detailed survey of applications of evolutionary robotics, see Floreano et al. (2008); this section gives a

brief overview of some areas covered by the methodology to give a better idea of the techniques involved and to indicate the scope of the field.

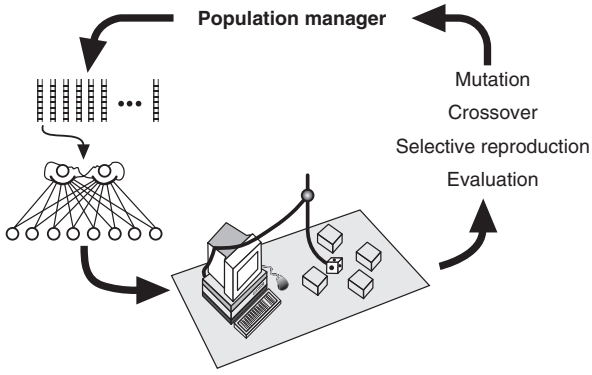
Prominent early centers for research in this area were EPFL and Sussex University, both of which are still very active in the field. Much of the early EPFL work used the miniature Khepera robot (Mondada, Franzi, & Ienne, 1993), which became a popular tool in many areas of robotics research. In its simplest form, it is a two-wheeled cylindrical robot with a ring of IR sensors around its body. The first successful evolutionary robotics experiments at EPFL employed the setup illustrated in Figs. 3 and 4. A population of bit strings encoded the connection weights and node thresholds for a simple fixed architecture feedforward neural network. Each member of the population was decoded into a particular instantiation of a neural network controller which was then downloaded onto the robot (Floreano & Mondada, 1994). This controlled the robot for a fixed period of time as it moved around the environment shown in Fig. 4.

The following simple fitness function was used to evolve obstacle avoidance behaviors:

$$F = V + (1 - \sqrt{DV}) + (1 - I)$$

where V is the average rotation speed of opposing wheels, DV is the difference between signed speed values of opposing wheels, and I is the activation value of the IR sensor with the highest input (readings are high if an obstacle is close to a sensor). Maximizing this function ensures high speed, a tendency to move in straight lines, and avoidance of walls and obstacles in the environment. After about 36 h of real-world evolution using this setup, controllers were evolved that successfully generated efficient motion around the course, avoiding collisions with the walls.

At the same time as this work was going on at EPFL, a series of pioneering experiments on evolving visually guided behaviors were being performed at Sussex University (Cliff et al., 1993; Harvey et al., 1994) in which discrete-time dynamical recurrent neural networks and visual sampling morphologies were concurrently evolved to allow a gantry robot (as well as other more standard mobile robots) to perform various visually guided tasks. An early instantiation of the Sussex gantry robot is shown in Fig. 5.

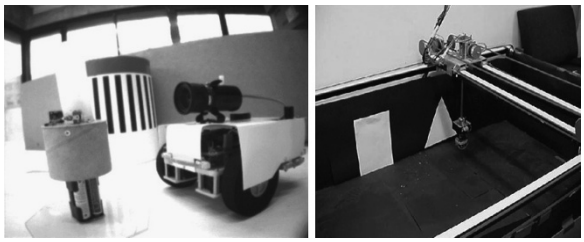


Evolutionary Robotics. Figure 3. Setup for early EPFL evolutionary robotics experiments with the Khepera robot (see text for details). Used with permission

Evolutionary Robotics. Figure 4. The simple environment used for evolving obstacle avoidance behaviors with a Khepera robot. Used with permission

A CCD camera points down toward a mirror angled at 45°. The mirror can rotate around an axis perpendicular to the camera's image plane. The camera is suspended from the gantry allowing motion in the X, Y, and Z dimensions. This effectively provides an equivalent to a wheeled robot with a forward facing camera when only the X and Y dimensions of translation are used (see Fig. 5).

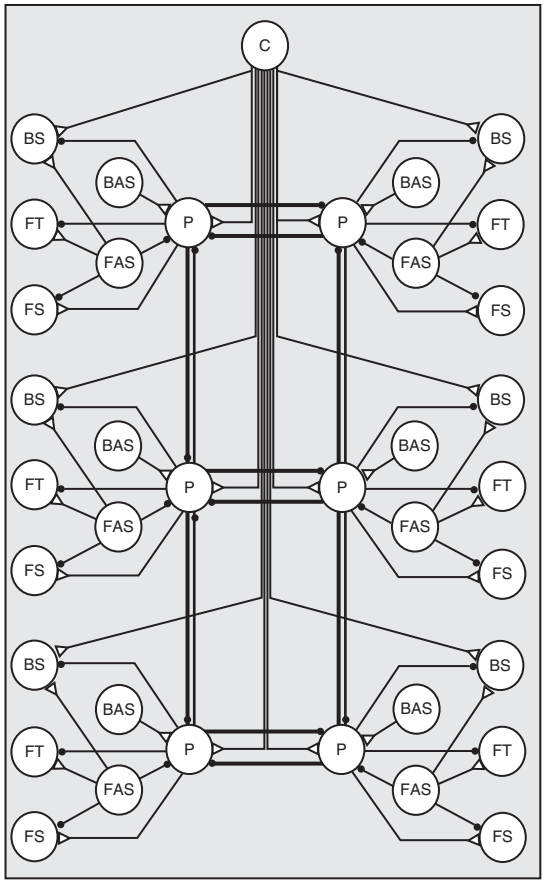
The apparatus was initially used in a manner similar to the real-world EPFL evolutionary robots setup illustrated in Fig. 3. A population of strings encoding robot controllers and visual sensing morphologies are stored on a computer to be downloaded one at a time onto the robot. The exact position and orientation of the camera head can be accurately tracked and used in the fitness evaluations. A number of visually



Evolutionary Robotics. Figure 5. An early version of the Sussex gantry robot (*right*) was a "hardware simulation" of a robot such as that shown on the left. It allowed real-world evolution of visually guided behaviors in an easily controllable experimental setup (see text for further details)

guided navigation behaviors were successfully achieved, including navigating around obstacles and discriminating between different objects. In the experiment illustrated in Fig. 5, starting from a random position and orientation the robot has to move to the triangle rather than the rectangle. This has to be achieved irrespective of the relative positions of the shapes and under very noisy lighting conditions. The architecture and all parameters of recurrent neural network controllers were evolved in conjunction with visual sampling morphologies – only genetically specified patches from the camera image were used (by being fed to input neurons according to a genetic specification), the rest of the image is thrown away. This resulted in extremely minimal systems only using 2 or 3 pixels of visual information, yet still able to very robustly perform the task under highly variable lighting conditions. Behaviors were evolved in an incremental way, with more complex capabilities being evolved from populations of robots that were successful at some simpler task (for details see Harvey et al. (1994) and Harvey, Husbands, Cliff, Thompson, & Jakobi (1997)). The highly minimal yet very robust systems developed highlighted the potential for evolutionary robotics techniques in areas such as space exploration where there is a great pressure to minimize resources while maintaining reliability (Hobbs, Husbands, & Harvey, 1996).

Since this early work, many different behaviors have been successfully evolved on a wide range of robots (Floreano et al., 2008; Nolfi & Floreano, 2000) There is not enough room to give an adequate summary of the whole field, so a few interesting subareas are highlighted below.



Evolutionary Robotics. Figure 6. Schematic diagram of a distributed neural network for the control of locomotion as used by Beer et al. Excitatory connections are denoted by open triangles, and inhibitory connections are denoted by filled circles. C, command neuron; P, pacemaker neuron; FT, foot motor neuron; FS and BS, forward swing and backward swing motor neurons; FAS and BAS, forward and backward angle sensors. Reproduced with permission

Over the past 15 years or so, there has been a growing body of work on evolving controllers for various kinds of walking robots – a nontrivial sensorimotor coordination task. Early work in this area concentrated on evolving dynamical network controllers for simple simulated insects (often inspired by cockroach studies), which were required to walk in uncomplicated environments (e.g., de Garis, 1990; Beer & Gallagher, 1992).

The promise of this work soon led to versions of this methodology being used on real robots. Probably, the first success in this direction was by Lewis, Fagg, and Solidum (1992) who evolved a neural controller for a simple hexapod robot, using coupled oscillators built from continuous-time, leaky-integrator, artificial neurons. The robot was able to execute an efficient tripod gait on flat surfaces. All evaluations were done on the actual robot with each leg connected to its own pair of coupled neurons, leg swing being driven by one neuron and leg elevation by the other. These pairs of neurons were cross-connected, in a manner similar to that used in the neural architecture shown in Fig. 6, to allow coordination between the legs. This architecture for locomotion, introduced by Beer, Chiel, and Sterling (1989), was based on the studies of cockroaches and has been much used ever since. Gallagher, Beer, Espenschiel, and Quinn (1996) used a generalization of it to evolve controllers for generating locomotion in a hexapod robot. This machine was more complex than Lewis et al.'s, with a greater number of degrees of freedom per leg. In this work, each leg was controlled by a fully connected network of five continuous-time, leaky-integrator neurons, each receiving a weighted sensory input from that leg's angle sensor. The connection weights and neuron time constants and biases were under genetic control. This produced efficient tripod gaits for walking on flat surfaces. In order to produce a wider range of gaits operating at a number of speeds such that rougher terrain could be successfully negotiated, a slightly different distributed architecture, more inspired by stick insect studies, was found to be more effective (Beer, Quinn, Chiel, & Ritzmann, 1997).

Jakobi (1998b) successfully used his minimal simulation techniques to evolve controllers for an 8-legged robot. Evolution in simulation took less than 2 h on what would today be regarded as a very slow computer, and then transferred successfully to the real robot. Jakobi evolved modular controllers based on Beer's continuous recurrent network architecture to control the robot as it engaged in walking about its environment, avoiding obstacles and seeking out goals. The robot could smoothly change gait, move backward and forward, and even turn on the spot. More recently, related approaches have been successfully used to evolve controllers for more mechanically sophisticated robots such as the Sony Aibo (Tllez, Angulo, & Pardo, 2006). In the last few years, there has also been successful work on evolving coupled oscillator style neural controllers for the highly unstable dynamic problem of biped walking. Reil and Husbands (2002) showed that accurate physics-based simulations using physics-engine software could be used to develop controllers able to generate successful bipedal gaits. Reil and colleagues have now significantly developed this technology to exploits its commercial possibilities in the animation and games industries (see www.naturalmotion.com for further details). Vaughan has taken related work in another direction. He has successfully applied evolutionary robotics techniques to evolve a simulation of a 3D ten-degree of freedom bipedal robot. This machine demonstrates many of the properties of human locomotion. By using passive dynamics and compliant tendons, it conserves energy while walking on a flat surface. Its speed and gait can be dynamically adjusted and it is capable of adapting to discrepancies in both its environment and its body's construction (Vaughan, Di Paolo & Harvey, 2004). In general, the evolutionary development of neural network walking controllers, with their intricate dynamics, produces a wider range of gaits and generates smoother, more adaptive locomotion than the more standard use of finite state machine based systems employing parameterized rules governing the timing and coordination of individual leg movements.

Early single robot research was soon expanded to handle interactions between multiple robots. Floreano and Nolfi did pioneering work on the coevolution of predator–prey behaviors in physical robots (Floreano & Nolfi, 1997). The fitness of the prey robot was measured by how quickly it could catch the prey; the fitness of the prey was determined by how long it could escape the predator. Two Khepera robots were used in this experiment, each had the standard set of proximity sensors but the predator also has a vision system and the prey was able to move twice as fast as the predator. A series of interesting chasing and evasion strategies emerged. Later Quinn, Smith, Mayley, and Husbands (2003) demonstrated the evolution of coordinated cooperative behavior in a group of robots. A group of robots equipped only with IR proximity sensors were required to move as far as possible as a coordinated group starting from a random configuration. The task was solved by the robots adopting and then maintaining a specific formation.

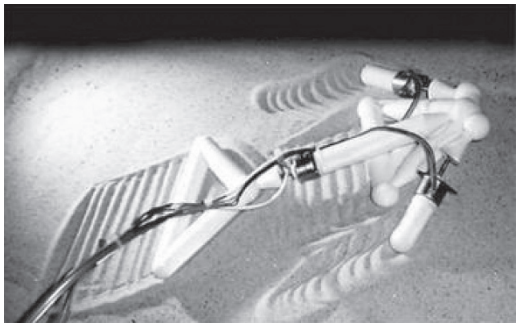
Analysis of the best evolved solution showed that it involved the robots adopting different roles, with the identical robots collectively "deciding" which robot would perform each role. Given the minimal sensing constraints, the evolved system would have proved extremely difficult to have designed by hand. For discussion of other multiple robot behaviors, see Floreano et al. (2008).

In the work described so far, control systems have been evolved for pre-existing robots: the brain is constrained to fit a particular body and set of sensors. Of course in nature, the nervous system evolved simultaneously with the rest of the organism. As a result, the nervous system is highly integrated with the sensory apparatus and the rest of the body: the whole operates in a harmonious and balanced way – there are no distinct boundaries between the control system, the sensors, and the body.

Karl Sims started to explore the concurrent evolution of the brain and the body in his highly imaginative work involving simulated 3D "creatures" (Sims, 1994). In this work, the creatures coevolved under a competitive scenario in which they were required to try and gain control of a resource (a cube) placed in the centre of an arena. Both the morphology of the creatures and the neural system controlling their actuators were under evolutionary control.

Lipson and Pollack (2000), working at Brandeis University, pushed the idea of fully evolvable robot hardware about as far as is reasonably technologically feasible at present. In an important piece of research, directly inspired by Sims' earlier simulation work, autonomous "creatures" were evolved in simulation out of basic building blocks (neurons, plastic bars, and actuators). The bars could connect together to form arbitrary truss structures with the possibility of both rigid and articulated substructures. Neurons could be connected to each other and to the bars whose length they would then control via a linear actuator. Machines defined in this way were required to move as far as possible in a limited time. The fittest individuals were then fabricated robotically using rapid manufacturing technology (plastic extrusion 3D printing) to produce results such as that shown in Fig. 7. They thus achieved autonomy of design and construction using evolution in a "limited universe" physical simulation coupled to automatic fabrication. The highly unconventional designs thus realized performed as well in reality as in simulation. The success of this work points the way to new possibilities in developing energy efficient fault tolerant machines.

Pfeifer and colleagues at Zurich University have explored issues central to the key motivation for fully evolvable robot hardware: the balanced interplay between body morphology, neural processing, and generation of adaptive behavior and have developed a set of design principles for intelligent systems in which these issues take centre stage (Pfeifer & Bongard, 2007).



Evolutionary Robotics. Figure 7. A fully automatically evolved robot developed on the Golem project (see text for details). Used with permission

Future Directions

Major ongoing challenges – methodological, theoretical, and technological – include finding the best way to incorporate development and lifetime plasticity within the evolutionary framework (this involves trends coming from the emerging field of epigenetic robotics), understanding better what the most useful building blocks are for evolved neurocontrollers, and finding efficient ways to scale work on concurrently evolving bodies and brains.

There are very interesting developments in the evolution of group behaviors and the emergence of communication (Di Paolo, 1998; Floreano, Mitri, Magnenat, & Keller, 2007; Quinn, 2001), the use of evolutionary robotics as a tool to illuminate problems in cognitive science (Beer, 2003; Harvey et al., 2005) and neuroscience (Di Paolo, 2003; Philippides et al., 2005;

Seth, 2005), in developing flying behaviors (Floreano, Hauert, Leven, & Zufferey, 2007; Shim & Husbands, 2007), and in robots that have some form of self-model (Bongard, Zykov, & Lipson, 2006), to name but a few.

Cross References

►Co-Evolutionary Learning
►Evolutionary Artificial Neural Networks
►Genetic Algorithms
►Robot Learning

Recommended Reading

Beer, R. D. (2003). The dynamics of active categorical perception in an evolved model agent (with commentary and response). *Adaptive Behavior*, *11*(4), 209–243.

Beer, R. D., Chiel, H. J., & Sterling, L. S. (1989). Heterogeneous neural networks for adaptive behavior in dynamic environments. In D. Touretzky (Ed.), *Neural information processing systems* (vol.1, pp. 577–585). San Francisco, CA: Morgan Kauffman.

Beer, R. D., & Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behaviour. *Adaptive Behaviour*, *1*, 94–110.

Beer, R. D., Quinn, R. D., Chiel, H. J., & Ritzmann, R. E. (1997). Biologically-inspired approaches to robotics. *Communications of the ACM*, *40*(3), 30–38.

Bongard, J., Zykov, V., & Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, *314*, 1118–1121.

Cliff, D., Harvey, I., & Husbands, P. (1993). Explorations in evolutionary robotics. *Adaptive Behavior*, *2*, 73–110.

de Garis, H. (1990). Genetic programming: Evolution of time dependent neural network modules which teach a pair of stick legs to walk. In *Proceedings of the 9th European conference on artificial intelligence* (pp. 204–206). Stockholm, Sweden.

De Jong, K. A. (2006). *Evolutionary computation: A unified approach*. Cambridge, MA: MIT Press.

Di Paolo, E. (1998). An investigation into the evolution of communication. *Adaptive Behavior, 6*(2), 285–324.

Di Paolo, E. A. (2003). Evolving spike-timing dependent plasticity for single-trial learning in robots. *Philosophical Transactions of the Royal Society A*, *361*, 2299–2319.

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing*. Berlin: Springer.

Floreano, D., Hauert, S., Leven, S., & Zufferey, J. C. (2007). Evolutionary swarms of flying robots. In D. Floreano (Ed.), *Proceedings of the international symposium on flying insects and robots*, (pp. 35–36). Monte Verita, Switzerland: EPFL.

Floreano, D., Husbands, P., & Nolfi, S. (2008). Evolutionary robotics. In B. Siciliano, & O. Khatib (Eds.), *Springer handbook of robotics* (Chap. 61). (pp.1423–1451). Berlin: Springer.

Floreano, D., Mitri, S., Magnenat, S., & Keller, L. (2007). Evolutionary conditions for the emergence of communication in robots. *Current Biology*, *17*, 514–519.

Floreano, D., & Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In D. Cliff, P. Husbands, J. Meyer, & S. W. Wilson (Eds.), *From animals to animats III: Proceedings of the third international conference on simulation of adaptive behavior* (pp. 402–410). Cambridge, MA: MIT Press-Bradford Books.

Floreano, D., & Nolfi, S. (1997). Adaptive behavior in competing co-evolving species. In P. Husbands, & I. Harvey (Eds.), *Proceedings of the 4th European conference on artificial life* (pp. 378–387). Cambridge, MA: MIT Press.

Floreano, D., & Urzelai, J. (2000). Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, *13*(4–5), 431–443.

Gallagher, J., Beer, R., Espenschiel, M., & Quinn, R. (1996). Application of evolved locomotion controllers to a hexapod robot. *Robotics and Autonomous Systems*, *19*(1), 95–103.

Gruau, F. (1995). Automatic definition of modular neural networks. *Adaptive Behavior*, *3*(2), 151–183.

Gruau, F., & Quatramaran, K. (1997). Cellular encoding for interactive evolutionary robotics. In P. Husbands, & I. Harvey (Eds.), *Proceedings of the 4th European conference on artificial life*. Cambridge, MA: The MIT Press/Bradford Books

Harvey, I., Di Paolo, E., Wood, R., Quinn, M., & Tuci, E. (2005). Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life*, *11*(1–2), 79–98.

Harvey, I., Husbands, P., & Cliff, D. T. (1994). Seeing the light: Artificial evolution, real vision. In D. T. Cliff, P. Husbands, J. A. Meyer, & S. Wilson (Eds.), *From animals to animats 3: Proceedings of the third international conference on simulation of adaptive behaviour, SAB94* (pp. 392–401). Cambridge, MA: MIT Press.

Harvey, I., Husbands, P., Cliff, D., Thompson, A., & Jakobi, N. (1997). Evolutionary robotics: The Sussex approach. *Robotics and Autonomous Systems*, *20*, 205–224.

Hobbs, J., Husbands, P., & Harvey, I. (1996). Achieving improved mission robustness. In *4th European Space Agency workshop on advanced space technologies for robot applications – ASTRA'96*, Noordwijk, The Netherlands ESTEC.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Husbands, P., & Harvey, I. (1992). Evolution versus design: Controlling autonomous mobile robots. In *Proceedings of 3rd annual conf. on artificial intelligence, simulation and planning in high autonomy systems* (pp. 139–146) Los Alimitos, CA: IEEE Computer Society Press.

Jakobi, N. (1998a). Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behaviour*, *6*, 325–368.

Jakobi, N. (1998b). Running across the reality gap: Octopod locomotion evolved in a minimal simulation. In P. Husbands, & J. A. Meyer, (Eds.), *Evolutionary robotics: First European workshop, EvoRobot98* (pp. 39–58). Berlin: Springer.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulations in evolutionary robotics. In F. Moran et al. (Eds.), *Proceedings of 3rd European conference on artificial life* (pp. 704–720). Berlin: Springer.

Lewis, M. A., Fagg, A. H., & Solidum, A. (1992). Genetic programming approach to the construction of a neural network for a walking robot. In *Proceedings of IEEE international conference on robotics and automation* (pp. 2618–2623). Washington, DC: IEEE Press.

Lipson, H., & Pollack, J. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, *406*, 974–978.

Mondada, F., Franzi, E., & Ienne, P. (1993). Mobile robot miniaturization: A tool for investigation in control algorithms. In T. Yoshikawa, & F. Miyazaki (Eds.), *Proceedings of the third international symposium on experimental robotics* (pp. 501–513). Berlin: Springer.

Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics: The biology, Intelligence, and technology of self-organizing machines*. Cambridge, MA: MIT Press/Bradford Books.

Parisi, D., & Nolfi, S. (1993). Neural network learning in an ecological and evolutionary context. In V. Roberto (Ed.), *Intelligent perceptual systems* (pp. 20–40). Berlin: Springer.

Pfeifer, R., & Bongard, J. (2007). *How the body shapes the way we think: A new view of intelligence*. Cambridge, MA: MIT Press.

Philippides, A., Husbands, P., Smith, T., & O'Shea, M. (2005). Flexible couplings: Diffusing neuromodulators and adaptive robotics. *Artificial Life*, *11*(1&2), 139–160.

Quinn, M. (2001). Evolving communication without dedicated communication channels. In J. Kelemen, & P. Sosik. (Eds.), *Proceedings of the 6th European conference on artificial life*, ECAL'01 (pp. 357–366). Berlin: Springer.

Quinn, M., Smith, L., Mayley, G., & Husbands, P. (2003). Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical Transactions of the Royal Society of London, Series A: Mathematical, Physical and Engineering Sciences*, *361*, 2321–2344.

Reil, T., & Husbands, P. (2002). Evolution of central pattern generators for bipedal walking in real-time physics environments. *IEEE Transactions of Evolutionary Computation*, *6*(2), 10–21.

Seth, A. K. (2005). Causal connectivity analysis of evolved neural networks during behavior. *Network: Computation in Neural Systems*, *16*(1), 35–54.

Shim, Y. S., & Husbands, P. (2007). Feathered flyer: Integrating morphological computation and sensory reflexes into a physically simulated flapping-wing robot for robust flight Manoeuvre. In *Proceedings of ECAL* LNCS (Vol. 4648, pp. 756–765). Berlin: Springer.

Sims, K. (1994). Evolving 3D morphology and behavior by competition. In R. Brooks, & P. Maes, (Eds.), *Proceedings of artificial life IV* (pp. 28–39). Cambridge, MA: MIT Press.

Tllez, R., Angulo, C., & Pardo, D. (2006). Evolving the walking behaviour of a 12 DOF quadruped using a distributed neural architecture. In *2nd International workshop on biologically inspired approaches to advanced information technology (Bio-ADIT'2006)* LNCS (Vol. 385, pp. 5–19). Berlin: Springer.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*, 433–460.

Urzelai, J., & Floreano, D. (2001). Evolution of adaptive synapses: Robots with fast adaptive behavior in new environments. *Evolution Computing*, *9*, 495–524.

Vaughan, E., Di Paolo, E. A., & Harvey, I. (2004). The evolution of control and adaptation in a 3D powered passive dynamic walker. In J. Pollack, M. Bedau, P. Husbands, T. Ikegami, & R. Watson, (Eds.), *Proceedings of the ninth international conference on the simulation and synthesis of living systems artificial life IX*, (pp. 139–145). Cambridge, MA: MIT Press.

Evolving Neural Networks

►Neuroevolution

Example

►Instance

Example-Based Programming

►Inductive Programming

Expectation Maximization Algorithm

►Expectation-Maximization Algorithm

Expectation Maximization Clustering

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

Synonyms

EM Clustering

The EM algorithm (Dempster, Laird, & Rubin 1977) finds maximum likelihood estimates of parameters in probabilistic models. EM is an iterative method which alternates between two steps, expectation (*E*) and maximization (*M*). For clustering, EM makes use of the finite Gaussian mixtures model and estimates a set of parameters iteratively until a desired convergence value is achieved. The mixture is defined as a set of *K* probability distributions and each distribution corresponds to one cluster. An instance is assigned with a membership probability for each cluster.

The EM algorithm for partitional clustering works as follows:

1. Guess initial parameters: mean and standard deviation (if using normal distribution model).

2. Iteratively refine the parameters with *E* and *M* steps. In the *E* step: compute the membership possibility for each instance based on the initial parameter values. In the *M* step: recompute the parameters based on the new membership possibilities.
3. Assign each instance to the cluster with which it has the highest membership possibility.

Cross References

►Expectation-Maximization Algorithm

Recommended Reading

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological), 39*(1), 1–38.

Expectation Propagation

Tom Heskes
Radboud University Nijmegen
Nijmegen, The Netherlands

Synonyms

EP

Definition

Expectation propagation is an algorithm for Bayesian machine learning (see ►Bayesian Methods). It tunes the parameters of a simpler approximate distribution (e.g., a Gaussian) to match the exact posterior distribution of the model parameters given the data. Expectation propagation operates by propagating messages, similar to the messages in (loopy) belief propagation (see ►Graphical Models). Whereas messages in belief propagation correspond to exact belief states, messages in expectation propagation correspond to approximations of the belief states in terms of expectations, such as means and variances. It is a deterministic method especially well-suited to large databases and dynamic systems, where exact methods for Bayesian inference fail and ►Monte Carlo methods are far too slow.

Motivation and Background

One of the main problems for ►Bayesian methods are their computational expense: computation of the exact posterior, given the observed data, typically requires the solution of high-dimensional integrals that have no analytical expressions. Approximation algorithms are needed to approximate this posterior as accurately as possible. These techniques for approximate inference can be subdivided in two categories: deterministic approaches and stochastic sampling (Monte Carlo) methods. Having the important advantage that (under certain conditions) they give exact results in the limit of an infinite number of samples, *Monte Carlo methods* are the method of choice in Bayesian statistics. However, in particular when dealing with large databases, the time needed for stochastic sampling to obtain a reasonably accurate approximation of the exact posterior can be prohibitive. This explains the need for faster, deterministic approaches, such as the Laplace approximation, *variational approximations*, and expectation propagation.

Expectation propagation was first described by Thomas Minka in his thesis (Minka, 2001). It can be viewed as a generalization and reformulation of the earlier ADATAP algorithm of Manfred Opper and Ole Winther (2001). Expectation propagation quickly became one of the most popular deterministic approaches for approximate Bayesian inference. Expectation propagation improves upon assumed density filtering, a classical method from stochastic control, by iteratively refining local approximations instead of computing them just once. Furthermore, it encompasses loopy belief propagation, a popular method for approximate inference in probabilistic ►graphical models, as a special case. Where loopy belief propagation is restricted to models of discrete variables only, expectation propagation applies to a much wider class of probabilistic graphical models with discrete and continuous variables and complex interactions between them.

Structure of Learning System

Bayesian Machine Learning

In the Bayesian framework for machine learning, you should enumerate all reasonable models of the data and assign a prior belief $P(w)$ to each of these models w. In the discrete case, the w are the different models, in the continuous case, the w are the continuous valued parameters (usually vectors). Then, upon observing the data D, you compute the likelihood $P(D|w)$ to evaluate how probable the data was under each of these models.

Expectation Propagation. Figure 1. (left-hand side) A so-called factor graph corresponding to the i.i.d. assumption in Bayesian machine learning. Each box corresponds to a factor or term. A circle corresponds to a variable. Factors are connected to the variables that they contain. Ψ_0 corresponds to the prior, $\Psi_1 \ldots \Psi_n$ are the likelihood terms for the n data points. (right-hand side) Factor graph of the approximating distribution. The original terms have been replaced by term approximations

The product of the prior and the likelihood gives you, up to a normalization constant, the posterior probability $P(w|D)$ over models given the data:

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)},$$

where the normalization term $P(D)$ is called the probability of the data or "evidence." This posterior probability incorporates all you have learned from the data D regarding the models w under consideration. As indicated above, exact calculation of this posterior probability is often infeasible, because the normalization term requires the solution of intractable sums or integrals.

In its simplest setting, the data D consists of n observations, $x_1, \ldots, x_n$, which are assumed to be independent and identically-distributed (i.i.d.). The posterior probability then factorizes into $n + 1$ terms, one for each observation and one for the prior. With definitions $\Psi_0(w) \equiv P(w)$ and $\Psi_i(w) \equiv P(x_i|w)$, we can rewrite

$$P(w|D) = \frac{P(w)\prod_{i=1}^{n} P(x_i|w)}{P(D)} \equiv \frac{\prod_{i=0}^{n} \Psi_i(w)}{P(D)}.$$

This factorization is visualized in the so-called factor graph in Fig. 1. We use it as a running example in the following section.

Assumed Density Filtering

Expectation propagation can be interpreted as an iterative refinement of assumed density filtering. In assumed density filtering, we add terms one-by-one and project in each step back to the "assumed density." For example, suppose that our prior probability $P(w) = \Psi_0(w)$ is a (known) Gaussian distribution over model parameters w, the terms corresponding to the data points are non-Gaussian, and we aim to find an appropriate Gaussian approximation $Q(w)$ to the exact (non-Gaussian) posterior $P(w|D)$. Our first approximation is the prior itself. Assumed-density filtering now proceeds by adding terms one at a time, where at each step we approximate the resulting distribution as closely as possible by a Gaussian. The pseudo-code is given in Algorithm 1, where $Q_{0:i}(w)$ denotes the approximation obtained after incorporating the prior and the first i observations.

If we use the Kullback–Leibler divergence as the distance measure from the non-Gaussian (but normalized) product of $Q_{0:i-1}(w)$ and $\Psi_i(w)$ and the Gaussian approximation, projection becomes "moment matching"; the result of the projection is the Gaussian that has the same mean and covariance matrix as the non-Gaussian product.

Expectation Propagation

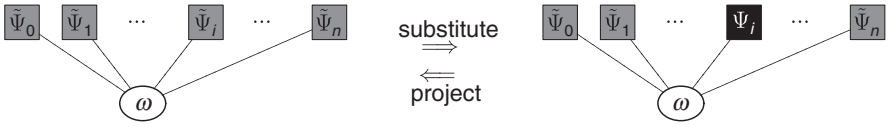
When in assumed density filtering, we add the term $\Psi_i(w)$, the Gaussian approximation changes from $Q_{0:i-1}(w)$ to $Q_{0:i}(w)$. We will call the quotient of the two the *term approximation* (here and in the following we ignore normalization constants):

$$\tilde{\Psi}_i(w) = \frac{Q_{0:i}(w)}{Q_{0:i-1}(w)}.$$

In our running example, term approximations are quotients between two different Gaussian densities and therefore have a Gaussian form themselves. Since the prior $\Psi_0(w)$ is a Gaussian density, $\tilde{\Psi}_0(w) = \Psi_0(w)$. The approximation $Q_{0:n}(w)$ is equal to the product of all

Algorithm 1 Assumed density filtering

1: $Q_0(w) = \Psi_0(w)$
2: **for** $i = 1$ to n **do**
3: $Q_{0:i}(w) = \text{Project\_to\_Gaussian}(Q_{0:i-1}(w)\Psi_i(w))$
4: **end for**



Expectation Propagation. Figure 2. Visualization of expectation propagation when recomputing the term approximation for observation *i*

term approximations and is visualized on the righthand side of Fig. 1. In assumed density filtering, the resulting approximation depends on the ordering in which the terms have been added. For example, if the terms had been added in reverse order, their term approximations might have been (slightly) different.

Expectation propagation now generalizes assumed density filtering by iteratively refining these term approximations. When successful, the final approximation will be independent of the ordering. Pseudo-code of expectation propagation is given in Algorithm 2. In step 1 through 5, the term approximations are initialized; in step 6 through 12, these term approximations are iteratively refined until they no longer change. In step 8, we take out the previous term approximation from the current approximation. In step 9, we put back in the exact term and project back to a Gaussian, like we did in assumed density filtering. It is easy to check that the approximation $Q(w)$ after the first loop equals the approximation $Q_{0:n}(w)$ obtained with assumed density filtering. The recalculation of the term approximation corresponding to observation i is visualized in Fig. 2.

Computational Aspects

With expectation propagation, we have to do a little more bookkeeping than with assumed density filtering: we have to keep track of the term approximations. One loop of expectation propagation is about as expensive as running assumed density filtering. Typically, about five iterations are sufficient for convergence.

The crucial operation is in step 3 of Algorithm 1 and step 9 of Algorithm 2. Here we have to compute the moments of the (non-Gaussian) probability distribution on the right-hand side. In most cases, we do not have analytical expressions for these moments and have to compute them numerically, e.g., using Gaussian quadrature. We then obtain the moments (mean and covariance matrix) of the new approximation $Q(w)$. Divisions and multiplications correspond to a simple subtraction and addition of so-called canonical parameters. For the Gaussian, these canonical parameters are the inverse of the covariance matrix (precision matrix) and the product of the precision matrix and the mean. The bottom-line is that we go back and forth between distributions in terms of moments and in terms of canonical parameters. For a Gaussian, this requires computing the inverse of the covariance matrix, which is roughly on the order of d^3, where d is the dimension of w. A practical point of concern is that matrix inversion is numerically instable, in particular for matrices that are close to singular, which can lead to serious round-off errors.

Convergence Issues

Sadly enough, expectation propagation is not guaranteed to converge to a fixed point. If it does, this fixed point can be shown to correspond to an extremum of the so-called Bethe free energy, an approximation of the "evidence" $\log P(D)$, under particular consistency and normalization constraints (Heskes, Opper, Wiegerinck,

Algorithm 2 Expectation propagation

1: $\tilde{\Psi}_0(w) = \Psi_0(w)$
2: **for** $i = 1$ to n **do**
3: $\quad\tilde{\Psi}_i(w) = 1$
4: **end for**
5: $Q(w) = \prod_{i=0}^{n} \tilde{\Psi}_i(w)$
6: **while** not converged **do**
7: $\quad$**for** $i = 1$ to n **do**
8: $\quad\quad Q_{-i}(w) = \dfrac{Q(w)}{\tilde{\Psi}_i(w)}$
9: $\quad\quad Q(w) = \text{Project\_to\_Gaussian}(Q_{-i}(w)\Psi_i(w))$
10: $\quad\quad \tilde{\Psi}_i(w) = \dfrac{Q(w)}{Q_{-i}(w)}$
11: $\quad$**end for**
12: **end while**

Winther, & Zoeter, 2005; Heskes & Zoeter, 2002; Minka, 2001, 2005). These constraints relate to the projection step in Algorithm 2: after convergence, the moments of $Q(w)$ should be equal to the moments of the distribution obtained by taking out a term approximation and putting back the corresponding exact term. This should hold for all i.i.d. observations $i = 1, \ldots, n$ in the factor graph of Fig. 1: so we conclude that, after convergence, the moments ("expectations") of all distributions constructed in this way should be the same. Expectation consistent approximations are based on the exact same idea and indeed turn out to be equivalent to expectation propagation (Heskes et al., 2005).

When expectation propagation does not converge, we can try "damping": instead of replacing the old term approximation by the new one, we replace it by a log-convex combination of the old and the new one. In many cases, damping with a step size 0.1 makes expectation propagation converge, at the expense of requiring more iterations. However, even damping with an infinitesimally small step size is not guaranteed to lead to convergence. In those cases, we can try to minimize the Bethe free energy more explicitly with a so-called double-loop algorithm (Heskes & Zoeter, 2002): in the outer loop we compute a convex bound on the Bethe free energy, which we then minimize in the inner loop with an algorithm very similar to standard expectation propagation. Double-loop algorithms are an order of magnitude slower than standard expectation propagation.

Generalizations

The running example above serves to illustrate the main idea, but is of course rather restrictive. Expectation propagation can be applied with any member of the exponential family as approximating distribution (Minka, 2001; Seeger, 2005). The crucial operations are the projection step and the transformation from moment to canonical form: if these can be performed efficiently and robustly, expectation propagation is into play.

In many interesting cases, the model to be learned (here represented as a single variable w) contains a lot of structure. This structure can be exploited by expectation propagation to make it more efficient. For example, when a term only contains a subset of the elements of w, so does its term approximation. Also, we might take as the approximating distribution a distribution that factorizes over the elements of w, instead of a "full" distribution coupling all elements. For a Gaussian, this would amount to a diagonal instead of a full covariance matrix. Such a factorization will lead to lower memory requirements and faster computation, perhaps at the expense of reduced accuracy. More advanced approximations include Tree-EP, where the approximating structure is a tree, and generalized expectation propagation, which generalizes expectation propagation to include higher-order interactions in the same way as generalized belief propagation generalizes loopy belief propagation (Welling, Minka, & Teh, 2005).

Power expectation propagation (Minka, 2005) generalizes expectation propagation by considering a different distance measure in the projection step. Instead of taking the Kullback–Leibler divergence, we can take any so-called α-divergence. $\alpha = 1$ corresponds to the Kullback–Leibler divergence, $\alpha = -1$ to the Kullback–Leibler divergence with the two probabilities interchanged. In the latter case, we obtain a variational method called variational Bayes.

Programs and Data

Code for expectation propagation applied for Gaussian process classification can be found at http://www.kyb.tuebingen.mpg.de/bs/people/csatol/ogp/, and http://www.gaussianprocess.org/gpml/code/matlab/doc/classification.html. Kevin Murphy's Bayes Net toolbox (http://bnt.sourceforge.net) can provide a good starting point to write your own code for expectation propagation.

Applications

Expectation propagation has been applied for, among others, Gaussian process classification (Csató, 2002), inference in Bayesian networks and Markov random fields, text classification with Dirichlet models and processes (Minka & Lafferty, 2002), ▶logistic regression models for rating players (Herbrich & Graepel, 2006), and inference and learning in hybrid and nonlinear dynamic Bayesian networks (Heskes & Zoeter, 2002).

Future Directions

From an application point of view, expectation propagation will probably become one of the standard techniques for approximate Bayesian machine learning, much like the Laplace approximation and Monte Carlo methods. Future research may involve questions like

- When does expectation propagation converge? Can we design variants that are guaranteed to converge?
- What "power" to use in power expectation propagation for what kind of purposes?
- Can we adapt expectation propagation to handle approximating distributions that are not part of the exponential family?

Cross References

►Gaussian Distribution
►Gaussian Process
►Graphical Models

Recommended Reading

Csató, L. (2002). *Gaussian processes – iterative sparse approximations*. PhD thesis, Aston University, Birmingham, UK.

Herbrich, R., & Graepel, T. (2006). *TrueSkill: A Bayesian skill rating system*. (Tech. Rep. No. MSR-TR-2006-80). Cambridge, UK: Microsoft Research.

Heskes, T., Opper, M., Wiegerinck, W., Winther, O., & Zoeter, O. (2005). Approximate inference with expectation constraints. *Journal of Statistical Mechanics: Theory and Experiment, 11*, P11015-1–P11015-24.

Heskes, T., & Zoeter, O. (2002). Expectation propagation for approximate inference in dynamic Bayesian networks. In A. Darwiche & N. Friedman (Eds.), *Proceedings of the 18th conference on uncertainty in artificial intelligence* (pp. 216–223). San Francisco: Morgan Kaufmann.

Minka, T. (2001). *A family of algorithms for approximate Bayesian inference*. PhD thesis, Cambridge, MA: MIT.

Minka, T. (2005). *Divergence measures and message passing*. (Tech. Rep. NO. MSR-TR-2005-173), Cambridge, UK: Microsoft Research.

Minka, T., & Lafferty, J. (2002). Expectation-propogation for the generative aspect model. In A. Darwiche & N. Friedman (Eds.), *Proceedings of the 18th conference on uncertainty in artificial intelligence* (pp. 352–359). San Francisco: Morgan Kaufmann.

Opper, M., & Winther, O. (2001). Tractable approximations for probabilistic models: The adaptive Thouless-Anderson-Palmer mean field approach. *Physical Review Letters, 86*, 3695–3699.

Seeger, M. (2005). *Expectation propagation for exponential families* (Tech. Rep.). Berkeley, CA: University of California.

Welling, M., Minka, T., & Teh, Y. (2005). Structured region graphs: Morphing EP into GBP. In F. Bacchus & T. Jaakkola (Eds.), *Proceedings of the 21st conference on uncertainty in artificial intelligence (UAI)* (pp. 609–614). Arlington, VA: AUAI Press.

Expectation-Maximization Algorithm

Synonyms

EM Algorithm; Expectation Maximization Algorithm

Expectation-Maximization (EM) was described by Arthur Dempster, Nan Laird, and Donald Rubin in a classic 1977 paper in the Journal of the Royal Statistical Society. The EM algorithm is used for finding maximum likelihood estimates of parameters in stochastic models, where the model depends on unobserved latent or hidden variables. EM iterates between performing expectation (E) and maximization (M) steps. Each expectation step involves the computation of the expectation of the likelihood of all model parameters by including the hidden variables as if they were observed. Each maximization step involves the computation of the maximum likelihood estimates of the parameters by maximizing the expected likelihood found during the expectation step. The parameters produced by the maximization step are then used to begin another expectation step, and the process is repeated.

It can be shown that an EM iteration will not decrease the observed data likelihood function. However, there is no guarantee that the iteration converges to a maximum likelihood estimator.

"Expectation-maximization" has developed to be a general recipe and umbrella term for a class of algorithms that iterates between a type of expectation and maximization step. The Baum–Welch algorithm is an example of an EM algorithm specifically suited to HMMs.

Experience Curve

►Learning Curves in Machine Learning

Experience-Based Reasoning

►Case-Based Reasoning

Explanation

In ►Minimum Message Length, an *explanation* is a code with two parts, where the first part is an *assertion* code and the second part is a *detail* code.

Explanation-Based Generalization for Planning

►Explanation-Based Learning for Planning

Explanation-Based Learning

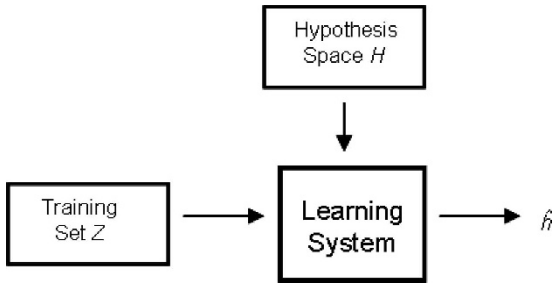
Gerald DeJong[1], Shiau Hong Lim[2]
[1]University of Illinois at Urbana
Urbana, IL, USA
[2]University of Illinois,
IL, USA

Synonyms

Analytical learning; Deductive learning; EBL; Utility problem

Definition

Explanation-Based Learning (EBL) is a principled method for exploiting available domain knowledge to improve ►supervised learning. Improvement can be in speed of learning, confidence of learning, accuracy of the learned concept, or a combination of these. In modern EBL the domain theory represents an expert's approximate knowledge of complex systematic world behavior. It may be imperfect and incomplete. Inference over the domain knowledge provides *analytic* evidence that compliments the empirical evidence of the training data. By contrast, in original EBL the domain theory is required to be much stronger; inferred properties are guaranteed. Another important aspect of modern EBL is the interaction between domain knowledge and



Explanation-Based Learning. Figure 1. Conventional learner

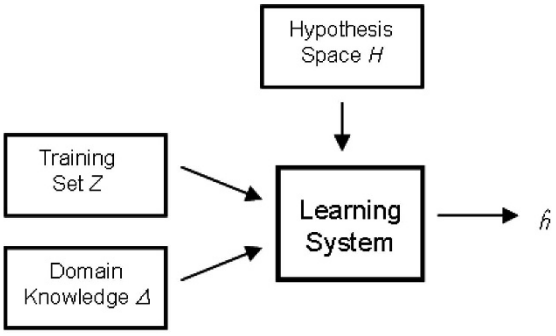
labeled training examples afforded by explanations. Interaction allows the nonlinear combination of evidence so that the resulting information about the target concept can be much greater than the sum of the information from each evidence source taken independently.

Motivation and Background

A conventional machine learning system is illustrated in Fig. 1. A hypothesis $\hat{h}$ is selected from a space of candidates H using a training set of labeled examples Z as evidence. It is common to assume that the examples are drawn from some space of well-formed inputs X according to some fixed but unknown distribution $\mathcal{D}$. The quality of $\hat{h}$ is to be judged against different examples similarly selected and labeled. The correct label for an example is specified by some ideal *target concept*, c^*. This is typically some complex world process whose outcome is of interest. The target concept, c^*, will generally not be a member of space of acceptable candidates, H. Rather, the learner tries to find some $\hat{h}$ which is acceptably similar to c^* over $X_{\mathcal{D}}$ and can serve as a computationally tractable stand-in.

Of course, good performance of $\hat{h}$ on Z (its training performance) alone is insufficient. The learner must achieve some statistical guarantee of good performance on the underlying distribution (test performance). If H is too rich and diverse or if Z is too impoverished, a learner is likely to ►overfit the data; it may find a pattern in the training data that does not hold in the underlying distribution $X_{\mathcal{D}}$. Test performance will be poor despite good training performance.

An Explanation-Based Learner employs its domain theory, Δ (Fig. 2) as an additional source of information. This domain theory must not be confused with ►learning bias, which is present in all learners. Determinations (Russell & Grosof, 1987) provide an extreme



Explanation-Based Learning. Figure 2. EBL learner

illustration. These are logical expressions that make strong claims about the world but only after seeing a training example. EBL domain theories are used only to explain. An inferred expression is not guaranteed to hold but only provides analytic evidence.

An explanation for some $z \in Z$ is immediately and easily generalized: The structure of the explanation accounts for why *z*s assigned classification label should follow from its features. All other examples that meet these conditions are assigned the same classification by the generalized explanation for the same reasons.

Early approaches to EBL (e.g., DeJong & Mooney, 1986; Mitchell, 1997; Mitchell, Keller, & Kedar-Cabelli, 1986; Russell & Norvig, 2003) were undone by two difficult problems: (1) unavoidable imperfections in the domain theory and (2) the utility problem. The former stems from assuming a conventional semantics for the domain theory. It results in a brittleness and an underreliance on the training data. Modern EBL is largely a reaction to this difficulty. The utility problem is a consequence of an ill-defined hypothesis space and, as will be discussed later, can be avoided in a straightforward manner.

Structure of Learning System

Explanations and Their Generalization

An *explanation* for a training example is any causal structure, derivable from Δ, which justifies why this training example might merit its teacher-assigned classification label. A *generalized explanation* is the structure of an explanation without the commitment to any particular example. The explanation and generalization processes are relatively straightforward and not significantly different from the original EBL algorithms.

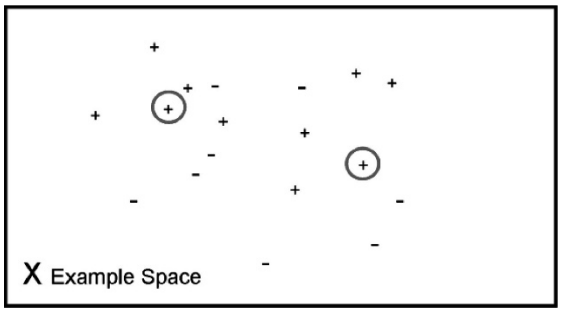
The weakness of early EBL is in viewing the components of Δ as constraints. This leads to a view of explanations and their generalizations as *proofs*. Real-world brittleness due to the qualification problem (McCarthy, 1980) follows inevitably. In modern EBL, Δ is seen as approximating the underlying world constraints (DeJong, 2006; Kimmig, De Raedt, & Toivonen, 2007). The domain theory is fundamentally a statistical device. Its analytic evidence and the empirical evidence of the training examples both provide a bridge to the real world.

The domain theory introduces new predicates and specifies their significant potential interactions. From a statistical point of view, these are named latent (hidden) features together with a kind of grammar for constructing alternative estimators for them. In short, the domain theory compactly specifies a large set of conceptual structures that an expert believes may be useful making sense of the domain. If the expert is correct, then patterns of interest will become computationally much more accessible via analytic inference.

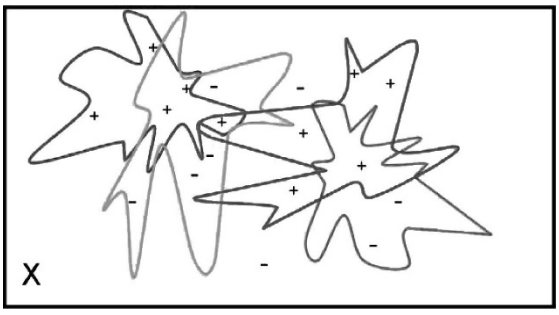
One flexible and useful form of a domain theory is sound inference over a set of first-order symbolic logic sentences. In such domain theories, the explanation mechanism can be identical to logical deduction although using a paraconsistent inference mechanism; inference must be well behaved despite inconsistencies in the theory. Generalized explanations are simply "theorems" of Δ that relate a classification label to the values of observable example features. But since the sentences of the theory only approximate world constraints, derivation alone, even via sound inference, is not sufficient evidence to believe a conclusion. Thus, a generalized explanation is only a conjecture. Additional training examples beyond those used to generate each explanation help to estimate the utility of these generalizations.

But analytic mechanisms need not be limited to symbolic logic-like inference. For example, one EBL approach is to distinguish handwritten Chinese characters (Lim, Wang, & DeJong, 2007) employing a Hough transform as a component of the domain theory. There, an explanation conjectures (hidden) glyph "strokes" to explain how the observed pixels of the training images may realize the image's character label.

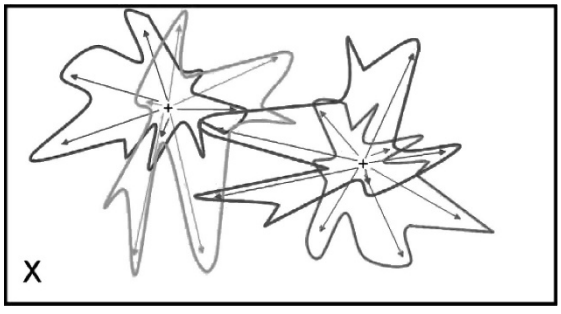
Whatever the form of the analytic inferential mechanism, multiple, quite incompatible explanations can be generated; the same training label can be explained



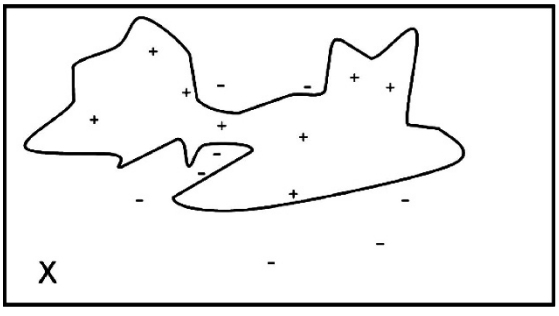
Explanation-Based Learning. Figure 3. An example space with two designated positive training items



Explanation-Based Learning. Figure 5. Explanations are evaluated with other training examples



Explanation-Based Learning. Figure 4. Four constructed explanations are sufficient to cover the positive examples



Explanation-Based Learning. Figure 6. An element from *H* that approximates the weighted explanations

using different input features and postulating different interactions. Such explanations will generalize to cover quite different subsets of X. Figure 3 shows a small training set with two positive examples highlighted. While the explanation process can be applied to all examples both positive and negative, these two will be used to illustrate. In this illustration, just two explanations are constructed for each of the highlighted training examples. Figure 4 shows the generalized extensions of these four explanations in the example space. The region enclosed by each contour is meant to denote the subset of X conjectured to merit the same classification as the explained example. Explanations make no claim about the labels for examples outside their extension.

Evaluation and Hypothesis Selection

Additional training examples that fall within the extension of a generalized explanation help to evaluate it empirically. This is shown in Fig. 5. The estimated utility of a generalized explanation reflects (1) the generalized explanation's empirical accuracy on these training examples, (2) the inferential effort required to derive the explanation (see DeJong, 2006), and (3) the redundancies and interactions with other generalized explanations (higher utility is estimated if its correct predictions are less commonly shared by other generalized explanations).

The estimated utilities define an EBL classifier as a mixture of the generalized explanations each weighted by its estimated utility:

$$\hat{c}_{\mathrm{EBL}}(x) = \sum_{g \in GE(Z,\Delta)} u_g \cdot g(x),$$

where $GE(Z, \Delta)$ denotes the generalized explanations for Z from Δ and u_g is the estimated utility for g. This corresponds to a voting scheme where each generalized explanation that claims to apply to an example

casts a vote in proportion to its estimated utility. The votes are normalized over the utilities of voting generalized explanations. The mixture scheme is similar to that of sleeping experts (Freund, Schapire, Singer, & Warmuth, 1997). This EBL classifier approximates the target concept c^*. But unlike the approximation chosen by a conventional learner, $\hat{c}_{\mathrm{EBL}}$ reflects the information of Δ in addition to Z.

The final step is to select a hypothesis $\hat{h}$ from H. The EBL concept $\hat{c}_{\mathrm{EBL}}$ is used to guide this choice. Figure 6 illustrates the selection of a $\hat{h} \in H$, which is a good approximation to a utility-blended mixture of Fig. 5. This final step, selecting a hypothesis from H, is important but was omitted in original EBL. These systems employed generalized explanations directly. Unfortunately, such classifiers suffer from a difficulty known as the *utility problem* (Minton, 1990). Note this is a slightly different use of the term *utility*, referring to the performance of an application system. This system can be harmed more than helped by concepts such as $\hat{c}_{\mathrm{EBL}}$, even if these concepts provide highly accurate classification. Essentially, the average cost of evaluating an EBL concept may outweigh the average benefit that it provides to the application system. It is now clear that this utility problem is simply the manifestation of a poorly structured hypothesis space. Note that, in general, an EBL classifier itself will not be an element of the space of acceptable hypotheses H. Previous approaches to the utility problem (Etzioni, 1993; Gratch & DeJong, 1992; Greiner & Jurisica, 1992; Minton, 1990) identify and disallow offending EBL concepts. However, the root cause is addressed by employing the EBL concept as a guidance in selecting a $\hat{h} \in H$ rather than using $\hat{c}_{\mathrm{EBL}}$ directly. Without this last step, H is completely ignored. But H embodies all of the information in the learning problem about what makes an acceptable hypothesis. The "utility problem" is simply the manifestation of leaving out this important information.

Literature

The roots and motivation for EBL extend at least to the MACROPs of STRIPS (Fikes, Hart, & Nilsson, 1972). The importance of explanations of training examples was first suggested in DeJong (1981). The standard references for the early EBL work are Mitchell et al. (1986) and DeJong and Mooney (1986). When covering EBL, current textbooks give somewhat refined versions of this early approach (Mitchell, 1997; Russell & Norvig, 2003). Important related ideas include determinations (Russell & Grosof, 1987), chunking (Laird, Rosenbloom, & Newell, 1986), and knowledge compilation (Anderson, 1986). EBL's ability to employ first-order theories make it an attractive compliment to learning Horn theories with ►Inductive Logic Programming (Bruynooghe, De Raedt, & De Schreye, 1989; Hirsh, 1987; Pazzani & Kibler, 1992; Zelle & Mooney, 1993). The problem of imperfect domain theories was recognized early, and there have been many approaches (Cohen, 1992; Flann & Dietterich, 1989; Genest, Matwin, & Plante, 1990; Ourston & Mooney, 1994; Thrun & Mitchell, 1993; Towell, Craven, & Shavlik, 1991). But with modern statistical learning ascending to the dominant paradigm of the field, interest in analytic approaches waned. The current resurgence of interest is largely driven by placing EBL in a modern statistically sophisticated framework that nonetheless is still able to exploit a first-order expressiveness (DeJong, 2006; Kimmig et al., 2007; Lim et al., 2007; Sun & DeJong, 2005)

Cross References

►Explanation-Based Learning for Planning
►Speedup Learning

Recommended Reading

Anderson, J. (1986). Knowledge compilation: The general learning mechanism. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine learning II* (pp. 289–310). San Mateo, CA: Morgan Kaufmann.

Bruynooghe, M., De Raedt, L., & De Schreye, D. (1989). Explanation based program transformation. In *IJCAI* (pp. 407–412).

Cohen, W. W. (1992). Abductive explanation-based learning: A solution to the multiple inconsistent explanation problem. *Machine Learning, 8*, 167–219.

DeJong, G. (1981). Generalizations based on explanations. In *IJCAI'81, the seventh international joint conference on artificial intelligence* (pp. 67–69). Vancover, BC.

DeJong, G. (2006). Toward robust real-world inference: A new perspective on explanation-based learning. In *ECML06, the seventeenth European conference on machine learning* (pp. 102–113). Heidelberg: Springer.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1*(2), 145–176.

Etzioni, O. (1993). A structural theory of explanation-based learning. *Artificial Intelligence, 60*(1), 93–139.

Fikes, R., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence, 3*(1–3), 251–288.

Flann, N. S., & Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning, 4*, 187–226.

Freund, Y., Schapire, R. E., Singer, Y., & Warmuth, M. K. (1997). Using and combining predictors that specialize. In *Twenty-ninth annual ACM symposium on the theory of computing* (pp. 334–343). El Paso, TX.

Genest, J., Matwin, S., & Plante, B. (1990). Explanation-based learning with incomplete theories: A three-step approach. In proceedings of the seventh international conference on machine learning (pp. 286–294).

Gratch, J., & DeJong, G. (1992). Composer: A probabilistic solution to the utility problem in speed-up learning. In *AAAI* (pp. 235–240).

Greiner, R., & Jurisica, I. (1992). A statistical approach to solving the EBL utility problem. In *National conference on artificial intelligence* (pp. 241–248). San Jose, CA.

Hirsh, H. (1987). Explanation-based generalization in a logic-programming environment. In *IJCAI* (pp. 221–227). Milan, Italy.

Kimmig, A., De Raedt, L., & Toivonen, H. (2007). Probabilistic explanation based learning. In *ECML'07, the eighteenth European conference on machine learning* (pp. 176–187).

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning, 1*(1), 11–46.

Lim, S. H., Wang, L.-L., & DeJong, G. (2007). Explanation-based feature construction. In *IJCAI'07, the twentieth international joint conference on artificial intelligence* (pp. 931–936)

McCarthy, J. (1980). Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence, 13*, 27–39.

Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence, 42*(2–3), 363–391.

Mitchell, T. (1997). *Machine learning*. New York: McGraw-Hill.

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*(1), 47–80.

Ourston, D., & Mooney, R. J. (1994). Theory refinement combining analytical and empirical methods. *Artificial Intelligence, 66*(2), 273–309.

Pazzani, M. J., & Kibler, D. F. (1992). The utility of knowledge in inductive learning. *Machine Learning, 9*, 57–94.

Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.

Russell, S. J., & Grosof, B. N. (1987). A declarative approach to bias in concept learning. In *AAAI* (pp. 505–510). Seattle, WA.

Sun, Q., & DeJong, G. (2005). Feature kernel functions: Improving svms using high-level knowledge. In *CVPR (2)* (pp. 177–183)

Thrun, S., & Mitchell, T. M. (1993). Integrating inductive neural network learning and explanation-based learning. In *IJCAI* (pp. 930–936). Chambery, France.

Towell, G. G., Craven, M., & Shavlik, J. W. (1991). Constructive induction in knowledge-based neural networks. In proceedings of the eighth international conference on machine learning (pp. 213–217)

Zelle, J. M., & Mooney, R. J. (1993). Combining Foil and EBG to speed-up logic programs. In *IJCAI* (pp. 1106–1113). Chambery, France.

Explanation-Based Learning for Planning

Subbarao Kambhampati[1], Sungwook Yoon[2]
[1]Arizona State University, Tempe, AZ, USA
[2]Palo Alto, CA, USA

Synonyms

Explanation-based generalization for planning; Speedup learning for planning

Definition

►Explanation-based learning (EBL) involves using prior knowledge to explain ("prove") why the training example has the label it is given, and using this explanation to guide the learning. Since the explanations are often able to pinpoint the features of the example that justify its label, EBL techniques are able to get by with much fewer number of training examples. On the flip side, unlike general classification learners, EBL requires prior knowledge (aka "domain theory/model") in addition to labeled training examples – a requirement that is not easily met in some scenarios. Since many planning and problem solving agents do start with declarative domain theories (consisting at least descriptions of actions along with their preconditions and effects), EBL has been a popular learning technique for planning.

Dimensions of Variation

The application of EBL in planning varies along several dimensions: whether the learning was for improving the speed and quality of the underlying planner (Etzioni, 1993; Kambhampati, 1994; Kambhampati, Katukam, & Qu, 1996; Minton et al., 1989; Yoon, Fern, & Givan, 2008) or acquire the domain model (Levine & DeJong, 2006); whether it was was done from successes (Kambhampati, 1994; Yoon et al. 2008) or failures (Ihrig

& Kambhampati, 1997; Minton et al., 1989); whether the explanations were based on complete/correct (Kambhampati et al., 1996; Minton et al., 1989) or partial domain theories (Yoon et al.), whether learning is based on single (Kambhampati, 1994; Kambhampati et al.; Minton et al., 1989) or multiple examples (Estlin & Mooney, 1997; Flann & Dietterich, 1989) (where, in the latter case, inductive learning is used in conjunction with EBL) and finally whether the planner whose performance EBL aims to improve is a means-ends analysis one (Minton et al., 1989), partial-order planner (Estlin & Mooney, 1997) or a heuristic search planner (Yoon et al.).

EBL techniques have been used in planning both to improve search and to reduce domain modeling burden (although the former has received more attention by far). In the former case, EBL is used to learn "control knowledge" to speedup the search process (Kambhampati et al., 1996; Minton et al., 1989), or to improve the quality of the solutions found by the search process (Estlin & Mooney, 1997). In the latter case EBL is used to develop domain models (e.g., action models) (Levine & DeJong, 2006).

EBL for search improvement involves either remembering and reusing successful plans, or learning search control rules to avoid failing search branches. Other variations include learning effective indexing of stored cases from retrieval failures (Ihrig & Kambhampati, 1997) and learning "adjustments" to the default heuristic used by the underlying search.

Another important issue is the degree of completeness/correctness of the underlying background theory used to explain examples. If the theory is complete and correct, then learning is possible from a single example. This type of EBL has been called "analytical learning." When the theory is partia, EBL still is effective in narrowing down the set of potentially relevant features of the training example. These features can then be used within an inductive learner. Within planning, EBL has been used in the context of complete/correct as well as partial domain models.

A final dimension of variation that differentiated a large number of research efforts is the type of underlying planner. Initially, EBL was used on top of means-ends analysis planners (cf. PRODIGY, Minton et al., 1989). Later work focused on partial order planners (e.g., Estlin & Mooney, 1997; Kambhampati et al., 1996). More recently, the focus has been on forward search state-space planners (Yoon et al., 2008).

Learning from Success: Explanation-Based Generalization

When learning from successful cases (plans), the training examples comprise of successful plans, and the explanations involve proofs showing that the plan, as it is given, is able to support the goals. Only the parts of the plan that take part in this proof are relevant for justifying the success of the plan. The plan is thus "generalized" by removing extraneous actions that do not take part in the proof. Object identifiers and action orderings are also generalized as long as the generalization doesn't affect the proof of correctness (Kambhampati, 1994). The output of the learning phase is thus a variablized plan containing a subset of the constraints (actions, orderings, object identity constraints) of the original plan. This is then typically indexed and used as a macro-operator to speed-up later search.

For example, given a planning problem of starting with an initial state where five blocks, A, B, C, D and E are on table, and the problem requires that in the goal state A must be on B and C must be on D, and a plan P that is a sequence of actions *pickup A, stack A on B, pickup E, putdown E, Pickup C, stack C on D*, the explanation-based learner might output the generalization "*do in any order* { *pickup x, stack x on y*} { *pick up z, stack z on w*}" for the generalized goals *on* (x, y) *and on* (w, z), starting from a state where x, y, z and w are all on table and clear, and each of them denotes a distinct block.

One general class of such proof schema involves showing that every top level goal of the planning problem as well as the precondition of every action are established and protected. Establishment requires that there is an action in the plan that gives that condition, and protection requires that once established, the condition is not deleted by any intervening action.

A crucial point is that the extent of generalization depends on the flexibility of the proof strategy used. Kambhampati and Kedar (1994) discuss a spectrum of generalization strategies associated with a spectrum of proof strategies, while Shavlik (1990) discusses how the number of actions in the plan can also be generalized.

Learning from Failure

When learning from the failure of a search branch, EBL starts by analyzing the plans at the failing nodes and constructing an explanation of failure. The failure explanation is just a subset of constraints in the plan at the current search node, which, in conjunction with domain theory ensures that no successful solution can be reached by further refining this plan. The explanations can range from direct constraint inconsistencies (e.g., ordering cycles), to indirect violation of domain axioms (e.g., the plan requiring both clear(B) and On(A,B) to be satisfied at the same time point). The explanations at the leaf nodes are "regressed" over the decisions in the search tree to higher level nodes to get explanations of (implicit) failures in these higher level nodes. The search control rules can then essentially recommend pruning any search node which satisfies a failure explanation.

The deep affinity between EBL from search failures and the idea of ►nogood learning and dependency-directed backtracking in CSP is explored in Kambhampati (1998). As in dependency directed backtracking, the more succinct the explanation, the higher the chance of learning effective control rules. Note that effectiveness here is defined in terms of the match costs involved in checking whether the rule is applicable, and the search reductions provided when it is applicable. Significant work has been done to identify classes of failure explanation that are expected to lead to ineffective rules (Etzioni, 1993). In contrast to CSP that has a finite depth search tree, one challenge in planning is that often an unpromising search node might not exhibit any direct failure with a succinct explanation, and is abandoned by the search for heuristic reasons (such as the fact that the node crosses a depth limit threshold). Strategies for finding implicit explanations of failure (using domain axioms), as well as getting by with incomplete explanations of failure are discussed in Kambhampati et al. (1996). EBL from failures has also been applied to retrieval (rather than search) failures. In this case, the failure of extending a plan retrieved from the library to solve a new problem is used to learn new indexing schemes that inhibit that case from being retrieved in such situations (Ihrig & Kambhampati, 1997).

Learning Adjustments to Heuristics

Most recent work in planning has been in the context of heuristic search planners, where learning from failures doesn't work as well (since the heuristic search may change directions much before a given search branch ends in an explainable failure). One way of helping such planners is to improve their default heuristic (Yoon et al., 2008). Given a heuristic $h(s)$ that gives the heuristic estimate of state s, the aim in Yoon et al. is to learn an adjustment $\delta(s)$ that is added to $h(s)$ to get a getter estimate of $h^*(s)$ – the true cost of state s. The system has access to actual plan traces (which can be obtained by having the underlying planner solve some problems from scratch). For each state s on the trace, we know the true distance of state s from the goal state, and we can also compute the $h(s)$ value with respect to the default heuristic. This gives the learner a set of training examples which are pairs of states and the adjustments they needed to make to the default heuristic meet the true distance. In order to learn the $\delta(s)$ from this training data, we need to enumerate the features of state s that are relevant to it needing the specific adjustment. This is where EBL come in. Specifically, one way of enumerating the relevant features is to explain why s has the default heuristic value it does. This, in turn, is done by taking the features of the relaxed plan for state s. Since the relaxed plan is a plan that assumes away all negative interactions between the actions, relaxed plan features can be seen as features of the explanation of the label for state s in terms of a *partial domain theory* (one which ignores all the deletes of all actions).

EBL from Incomplete Domain Theories

While most early efforts for speed-up focused on complete and correct theories, several efforts also looked at speed-up learning from incomplete theories. The so-called Lazy EBL approaches (Chien, 1989; Tadepalli, 1989) work by first constructing partial explanations, and subsequently refine the over-general rules learned. Other approaches that use similar ideas outside planning include Flann and Dietterich (1989) and Cohen (1992). As we noted above, the work by Yoon et al. (2008) can also be seen as basing learning (in their case of adjustments to a default heuristic function) w.r.t. a partial domain theory.

EBL to Learn Domain Knowledge

Although most work in EBL for planning has been focused on speedup, there has also been some work aimed at learning domain knowledge (rather than control knowledge). Of particular interest is "operationalizing" a complex, if opaque, domain model by learning from it a simplified domain model that is adequate to efficiently solve an expected distribution of problems. The recent work by Levine and DeJong (2006) is an example of such an effort.

EBL and Knowledge-Level Learning

Although the focus of this article is on EBL as applied to planning, we need to foreground one general issue: whether EBL is capable of knowledge-level learning or not. A popular misconception of EBL is that since it depends on a complete and correct domain theory, no knowledge-level learning is possible, and speedup learning is the only possibility. (The origins of this misconception can be traced back to the very beginning. The two seminal articles on EBL in the very first issue of the Machine Learning journal differed profoundly in their interpretations of EBL. While Mitchell, Keller, and Kedar-Cabelli (1986) assumed that EBL by default works with complete and correct theories (thus precluding any knowledge-level learning), DeJong (2006) provide a more general view of EBL that uses background knowledge – whether or not it is complete – to focus the generalization (and as such can be seen as a knowledge-based feature-selection step for a subsequent inductive learner)). As we noted at the outset however, EBL is not required to depend on complete and correct domain theories, and when it doesn't, knowledge level learning is indeed possible.

Utility Problem and its Non-Exclusive Relation to EBL

As we saw above, much early work in EBL for planning focused on speed-up for the underlying planner. Some of the knowledge learned for speedup – especially control rules and macro-operators – can also adversely affect the search by increasing either the search space size (macros) and/or per-node cost (matching control rules). Clearly, in order for the net effect to be positive, care needs to be exercised as to which control rules and/or macros are stored. This has been called the "utility problem" (Minton, 1990) and significant attention has been paid to develop strategies that either dynamically evaluate the utility of the learned control knowledge (and forget useless rules) (Markovitch & Scott, 1988; Minton, 1990), or select the set of rules that best serve a given distribution of problem instances (Gratch, Chien, & DeJong, 1994).

Despite the prominent attention given to the utility problem, it is important to note the non-exclusive connection between EBL and utility problem We note that *any* strategy that aims to provide/acquire control knowledge will suffer from the utility problem. For example, utility problem also holds for inductive learning techniques that were used to learn control knowledge (cf. Leckie & Zukerman, 1993). In other words, it is not special to EBL but rather to the specific application task. We note that it is both possible to do speedup learning that is less suceptible to the utility problem (e.g., learn adjustments to heuristics, Yoon et al., 2008), and possible to to use EBL for knowledge-level learning (Levine & DeJong, 2006).

Current Status

EBL for planning was very much in vogue in late eighties and early nineties. However, as the speed of the underlying planners increased drastically, the need for learning as a crutch to improve search efficiency reduced. There has however been a recent resurgence of interest, both in further speeding up the planners, and in learning domain models. Starting 2008, there is a new track in the International Planning Competition devoted to learning methods for planning. In the first year, the emphasis was on speedup learning. ObtuseWedge, a system that uses EBL analysis to learn adjustments to the default heuristic, was among the winners of the track. The DARPA integrated learning initiative, and interest in model-lite planning have also brought focus back to EBL for planning – this time with partial domain theories.

Additional Reading

The tutorial (Yoon & Kambhampati, 2007) provides an up-to-date and broader overview of learning techniques applied to planning, and contains significant discussion of EBL techniques. The paper (Zimmerman & Kambhampati, 2003) provides a survey of machine learning techniques used in planning, and includes a more comprehensive listing of research efforts that applied EBL in planning.

Cross References

►Explanation-Based Learning
►Speedup Learning

Recommended Reading

Bhatnagar, N., & Mostow, J. (1994). On-line learning from search failures. *Machine Learning, 15*(1), 69–117.

Borrajo, D., & Veloso, M. M. (1997). Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *Artificial Intelligence Review, 11*(1–5), 371–405.

Chien, S. A. (1989). Using and refining simplifications: Explanation-based learning of plans in intractable domains. In *IJCAI* (pp. 590–595).

Cohen, W. W. (1992). Abductive explanation-based learning: A solution to the multiple inconsistent explanation problem. *Machine Learning, 8*, 167–219.

DeJong, G., & Mooney, R. J. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1*(2), 145–176.

Estlin, T. A., & Mooney, R. J. (1997). Learning to improve both efficiency and quality of planning. In *IJCAI 1997* (pp. 1227–1233).

Etzioni, O. (1993). A structural theory of explanation-based learning. *Artificial Intelligence, 60*(1), 93–139.

Flann, N. S., & Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning, 4*, 187–226.

Gratch, J., Chien, S. A., & DeJong, G. (1994). Improving learning performance through rational resource allocation. In *AAAI 1994* (pp. 576–581).

Ihrig, L. H., & Kambhampati, S. (1997). Storing and indexing plan derivations through explanation-based analysis of retrieval failures. *Journal of Artificial Intelligence Research, 7*, 161–198.

Kambhampati, S. (1994). A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence, 67*(1), 29–70.

Kambhampati, S. (1998). On the relations between intelligent backtracking and failure-driven explanation-based learning in constraint satisfaction and planning. *Artificial Intelligence, 105*(1–2), 161–208.

Kambhampati, S., Katukam, S., & Qu, Y. (1996). Failure driven dynamic search control for partial order planners: An explanation based approach. *Artificial Intelligence, 88*(1–2), 253–315.

Leckie, C., & Zukerman, I. (1993). An inductive approach to learning search control rules for planning. In *IJCAI 1993* (pp. 1100–1105)

Levine, G., & DeJong, G. (2006). Explanation-based acquisition of planning operators. In *ICAPS 2006* (pp. 152–161).

Markovitch, S., & Scott, P. D. (1988). The role of forgetting in learning. In *ML 1988* (pp. 459–465).

Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence, 42*(2–3), 363–391.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D., Etzioni, O., & Gil, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence, 40*(1–3), 63–118.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*(1), 47–80.

Shavlik, J. W. (1990). Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning, 5*, 39–40.

Tadepalli, P. (1989). Lazy explanation based learning: A solution to the intractable theory problem. In *IJCAI 1989* (pp. 694–700).

Yoon, S., Fern, A., & Givan, R. (2008). Learning control knowledge for forward search planning. *Journal of Machine Learning Research, 9*, 683–718.

Yoon, S., & Kambhampati, S. (2007). Learning for planning. Tutorial delivered at ICAPS 2007. http://rakaposhi.eas.asu.edu/learn-plan.html

Zimmerman, T., & Kambhampati, S. (2003). Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine, 24*(2), 73–96.

F

F_1-Measure

The F_1-measure is used to evaluate the accuracy of predictions in two-class (binary) ►classification problems. It originates in the field of information retrieval and is often used to evaluate ►document classification models and algorithms. It is defined as the harmonic mean of ►precision (i.e., the ratio of ►true positives to all instances predicted as positive) and ►recall (i.e., the ratio of true positives to all instances that are actually positive). As such, it lies between precision and recall, but is closer to the smaller of these two values. Therefore a system with high F_1 has both good precision and good recall. The F_1-measure is a special case of the more general family of evaluation measures:

$$F_\beta = (\beta^2 + 1)precisionrecall/(\beta^2 precision + recall)$$

Thus using $\beta >$ increases the influence of precision on the overall measure, while using $\beta < 1$ increases the influence of recall. Some authors use an alternative parameterization,

$$F_\alpha = 1/(\alpha/precision + (1t\alpha)/recall)$$

which, however, leads to the same family of measures; conversion is possible via the relationship $\alpha = 1/(\beta^2 + 1)$.

False Negative

In a two-class problem, a ►classification ►model makes two types of error: ►false positives and false negatives. A **false negative** is an example of positive class that has been incorrectly classified as negative. See ►confusion matrix for a complete range of related terms.

False Positive

In a two-class problem, a ►classification ►model makes two types of error: false positives and ►false negatives. A **false positive** is an example of negative class that has been incorrectly classified as positive. See ►confusion matrix for a complete range of related terms.

Feature

►Attribute

Feature Construction

►Data Preparation

Feature Construction in Text Mining

Janez Brank, Dunja Mladenić,
Marko Grobelnik
Jožef Stefan Insitute
Ljubljana, Slovenia

Synonyms

Feature generation in text mining

Definition

Feature construction in ►text mining consists of various techniques and approaches which convert textual data into a feature-based representation. Since traditional machine learning and data mining techniques are generally not designed to deal directly with textual data, feature construction is an important preliminary step in text mining, converting source documents

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

into a representation that a data mining algorithm can then work with. Various kinds of feature construction approaches are used in text mining depending on the task that is being addressed, the data mining algorithms used, and the nature of the dataset in question.

Motivation and Background

Text mining is the use of machine learning and data mining techniques on textual data. This data consists of natural language documents that can be more or less structured, ranging from completely unstructured plain text to documents with various kinds of tags containing machine-readable semantic information. Furthermore, documents may sometimes contain hyperlinks that connect them into a graph. Since most traditional machine learning and data mining techniques are not directly equipped to deal with this kind of data, an important first step in text mining is to extract or construct features from the input documents, thereby obtaining a feature-based representation which is suitable for handling with machine learning and data mining algorithms. Thus, the task of feature construction in text mining is inextricably connected with text mining itself and has evolved alongside it. An important trend over the years has been the development of techniques that do not process each document in isolation but make use of a corpus of documents as a whole, possibly even involving external data or background knowledge in the process.

Documents and text data provide for valuable sources of information and their growing availability in electronic form naturally led to application of different analytic methods. One of the common ways is to take a whole vocabulary of the natural language in which the text is written as a feature set, resulting in several tens of thousands of features. In a simple setting, each feature gives a count of the word occurrences in a document. In this way text of a document is represented as a vector of numbers. The representation of a particular document contains many zeros, as most of the words from the vocabulary do not occur in a particular document. In addition to the already mentioned two common specifics of text data, having a large number of features and a sparse data representation, it was observed that frequency of words in text in general follows Zipf's law – a small subset of words occur very frequently in texts while a large number of words occur only rarely. ▶Document classification takes these and some other data specifics into account when developing the appropriate classification methods.

Structure of Learning System

In a learning or mining system that deals with textual data, feature construction is usually one of the first steps that is often performed alongside typical preprocessing tasks such as data cleaning. A typical output of feature construction are feature vectors representing the input documents; these vectors themselves then form the input for a machine learning or data mining algorithm. On the other hand, sometimes feature construction is more closely integrated into the learning algorithm itself, and sometimes it can be argued that the features themselves are the desired output that is the goal of the text mining task.

Solutions

At the lowest level, text is represented as a sequence of bytes or other elementary units of information. How these bytes are to be converted into a sequence of characters depends on the *character encoding* of the text. Many standard encodings exist, such as UTF-8, the ISO-8859 family, and so on. Often, all the texts that appear as input for a specific text mining task are in the same encoding, or if various encodings are used they are specified clearly and explicitly (e.g., via the Content-Type header in the HTTP protocol), in which case the problem of conversion is straightforward. In the case of missing or faulty encoding information, various heuristics can be used to detect the encoding and convert the data to characters; it is best to think of this as a data cleaning and preprocessing step.

Word-Based Features

When we have our text represented as a sequence of characters, the usual next step is to convert it into a sequence of words. This is usually performed with heuristics which depend to some extent on the language and underlying character set; for the purposes of segmentation of text into words, a word is thought of as a sequence of alphabetic characters delimited by

whitespace and/or punctuation. Some efforts to standardize word boundary detection in a way that would work reasonably well with a large set of natural languages have also been made (see, e.g., the Unicode Standard Annex #29, *Unicode Text Segmentation*). For many (but not all) text mining tasks, the distinction between upper and lower case (if it is present in the underlying natural language) is largely or entirely irrelevant; hence, all text is often converted into lower case at this point. Another frequently used preprocessing step is *stemming*, whereby each word is replaced by its stem (e.g., *walking* → *walk*). The details of stemming depend on the natural language involved; for English, a relatively simple set of heuristics such as Porter's stemmer is sufficient. Instead of stemming, where the ending is chopped off the word, one can apply a more sophisticated transformation referred to as lemmatization that replaces the word by its normalized form (lemma). Lemmatization is especially relevant for natural languages that have many different forms of the same word (e.g., several cases, gender influence on verb form etc.). Efforts have also been made to discover stemming rules or lemmatization rules automatically using machine learning techniques (Plisson, Lavrač, Mladenić, & Erjavec, 2008).

The individual words can themselves be thought of as features of the document. In the feature-vector representation of a document d, the feature corresponding to the word w would tell something about the presence of the word w in this document: either the frequency (number of occurrences) of w in d, or a simple binary value (1 if present, 0 if absent), or it can further be modified by, e.g., the ▶TF-IDF weighting. In this kind of representation, all information about the word order in the original document is lost; hence, it is referred to as the "bag of words" model. For many tasks, the loss of word order information is not critical and the bag of words model is a staple of information retrieval, document classification, and many other text-related tasks. A downside of this approach (and many other word-based feature construction techniques) is that the resulting number of features can be very large (there are easily tens of thousands of different words in a mid-sized document corpus); see ▶Feature Selection in Text Mining.

Clearly, ignoring the word order completely can sometimes lead to the loss of valuable information. Multi-word phrases sometimes have a meaning that is not adequately covered by the individual words of the phrase (e.g., proper names, technical terms, etc.). Various ways of creating multi-word features have been considered. Let d be a document consisting of the sequence of words $(w_1, w_2, \ldots, w_m)$ (note that, this sequence might already be an output of some preprocessing operations, e.g., the removal of stopwords and of very infrequent words.). Then an *n-gram* is defined as a sequence of n adjacent words from the document, i.e., $(w_i, w_{i+1}, \ldots, w_{i+n-1})$. We can use n-grams as features in the same way as individual words, and indeed a typical approach is to use n-grams for all values of n from 1 to a certain upper limit (e.g., 5). Many of the resulting n-grams will be incidental and irrelevant, but some of them may be valuable and informative phrases; whether the text mining algorithm will be able to profit from them depends a lot on the algorithm used, and feature selection might be even more necessary than in the case of individual words. A related problem is the explosion of the number of features; if the number of different words in a corpus grows approximately with the square root of the length of the corpus (Heaps' law), the number of different n-grams is more likely to grow nearly linearly with the length of the corpus. The use of n-grams as features has been found to be beneficial, e.g., for the classification of very short documents (Mladenić & Grobelnik, 2003).

Further generalization of n-grams is possible by removing the requirement that the words of the n-gram must appear adjacently; we can allow them to be separated by other words. The weight of an occurrence of the n-gram is often defined as decreasing exponentially with the number of intervening separator words. Another direction of generalizing n-gram is to ignore the order of words within the n-gram; in effect one treats n-grams as bags (multisets) instead of sequences. This results in features sometimes called *loose phrases* or *proximity features* (i.e., every bag of words up to a certain size, occurring in sufficiently close proximity to each other, is considered to be a feature). These generalizations greatly increase the feature space as well as the number of features present in any individual document, so the risk of computational intractability is greatly increased; this can sometimes be alleviated through the use of kernels (see below).

Character-Based Features

Instead of treating the text as a sequences of words, we might choose to treat it as a sequence of characters. A sequence of n characters is also known as an n-graph. We can use n-graphs as features in the representation of text in a way analogous to the use of n-grams in the previous subsection. The weight of the feature corresponding to a particular n-graph in the feature vector of a particular document d will typically depend on the number of occurrences of that n-graph in the text of d. Sometimes noncontiguous occurrences of the n-graph are also counted (i.e., occurrences where characters from the n-graph are separated by one or more other characters), although with a lower weight; this is can be done very elegantly with kernel methods (see below). Feature selection and TF-IDF style weighting schemes can also be used as in the case of n-grams. Whether an n-graph-based representation offers any benefits compared to an n-gram-based one depends largely on the dataset and task in question. For example, the classification of English documents the usefulness of n-graphs has been found to be dubious, but they can be beneficial in highly agglutinative languages where an individual word can consist of many morphemes and it is not really useful to treat a whole word as an individual unit of information (as would be the case in a word-based feature representation). In effect, the use of n-graphs provides the learner with cheap access to the sort of information that would otherwise require more sophisticated NLP technologies (stemming, parsing, morpheme analysis, etc.); the downside is that a lot of the n-graph features are merely noise (Lodhi, Saunders, Shawe-Taylor, Cristianini, & Watkins, 2002). For some application, word suffixes can be particularly useful features, e.g., to learn lemmatization rules (Mladenić, 2002; Plisson et al., 2008).

Kernel Methods

Let ϕ be a function which assigns, to a given document d, a feature vector $\phi(d)$ from some feature space F. Assume furthermore that a dot product (a.k.a. inner product) is defined over F, denoted by $\langle \cdot, \cdot \rangle_F$. Then the function K defined by $K(d_1, d_2) = \langle \phi(d_1), \phi(d_2) \rangle_F$ is called a *kernel function*. It turns out that many machine learning and data mining methods can be described in a way such that the only operation they need to do with the data is to compute dot products of their feature vectors; in other words, they only require us to be able to compute the kernel function over our documents. These approaches are collectively known as ►*kernel methods*; a well-known example of this is the ►support vector machine (SVM) method for supervised learning, but the same principle can be used in ►clustering as well. An important advantage of this approach is that it is often possible to compute the kernel function K directly from the documents $d_{1,2}$ without explicitly generating the feature vectors $\phi(d_{1,2})$. This is especially valuable if the feature space is untractably large. Several families of kernel functions for textual data have been described in the literature, corresponding to various kinds of n-graph and n-gram based features (Brank, 2006; Lodhi et al., 2002).

Linear Algebra Methods

Assume that a corpus of n documents have already been represented by d-dimensional real feature vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in R^d$. If we select some direction $\mathbf{y} \in R^d$ and project a vector $\mathbf{x}_i$ in this direction, the resulting value $\mathbf{y}^T \mathbf{x}_i / ||\mathbf{y}||$ is in effect a new feature describing the document i. In other words, we have constructed a new feature as a linear combination of the existing features. This leads to the question of how to select one or more suitable directions $\mathbf{y}$; various techniques from linear algebra and statistics have been proposed for this.

A well-known example of this is *principal component analysis* (PCA) in which one or more new coordinate axes $\mathbf{y}$ are selected in such a way that the variance of the original vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in the directions of the new coordinate axes is maximized. As it turns out, this problem is equivalent to computing the principal eigenvectors of the covariance matrix of the original dataset.

Another technique of this sort is *latent semantic indexing* (LSI) (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990). Let X be a $d \times n$ matrix with $\mathbf{x}_1, \ldots, \mathbf{x}_n$ as its columns (a.k.a. the *term-document matrix*). LSI uses singular value decomposition (SVD) to express X as the product of three matrices, $T \cdot S \cdot D$, where T is a $d \times r$ orthonormal matrix, D is a $r \times n$ orthonormal matrix, and S is a $r \times r$ diagonal matrix containing the singular values of X. Here, r denotes the rank of the original matrix X. Let $T^{(m)}$ be the matrix consisting of the left m columns of T, let $D^{(m)}$ be the matrix consisting of the top m rows of D, and let $S^{(m)}$ be

the top left $m \times m$ submatrix of S. Then it turns out that $X^{(m)} = T^{(m)}S^{(m)}D^{(m)}$ is the best rank-m approximation of the original X (best in the sense of minimizing the Frobenius norm of $X - X^{(m)}$). Thus, the i-th column of $D^{(m)}$ can be seen as a vector of m new features representing the i-th document of our original dataset, and the product $T^{(m)}S^{(m)}$ can be seen as a set of m new coordinate axes. The new feature vectors (columns of $D^{(m)}$) can be used instead of the original vectors $\mathbf{x}_i$.

Canonical correlation analysis (CCA): Sometimes several vector representations are available for the same document d_i; for example, we might have the same text in two different languages, giving rise to two feature vectors, e.g., $\mathbf{x}_i \in R^d$ and $\mathbf{y}_i \in R^{d'}$. Given such a "parallel corpus" of pairs $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1 \ldots n$, it is sometimes desirable to convert both types of representations to a "common denominator." In other words, we want to find a set of r new coordinate axes in $\mathbf{x}$-space (say the columns of $U \in R^{d \times r}$) and a set of r new coordinate axes in $\mathbf{y}$-space (say the columns of $V \in R^{d' \times r}$) such that the j-th column of U has a similar role in $\mathbf{x}$-space as the j-th column of V has in $\mathbf{y}$-space, for all j. This can be formulated as an optimization problem: find U and V such that the correlation between $U^T\mathbf{x}_i$ and $V^T\mathbf{y}_i$ (i.e., the projections of $\mathbf{x}_i$ and $\mathbf{y}_i$ onto the new sets of axes) is maximized. Once we have suitable matrices U and V, we can convert any feature vector from the original $\mathbf{x}$-space or $\mathbf{y}$-space into a common new r-dimensional space. This makes it easier to deal with multi-lingual corpora, allowing us, e.g., to retrieve documents in language $\mathbf{x}$ as a response to a query in language $\mathbf{y}$, or vice versa. The same techniques are applicable in multimodal scenarios (i.e., $\mathbf{x}_i$ and $\mathbf{y}_i$ can be any two representations of the same instance d_i from two substantially different perspectives, not necessarily textual). This method is often used in combination with kernels, in which case it is known as *kernel canonical correlation analysis* (KCCA) (Hardoon, Szedmak, & Shawe-Taylor, 2004).

Miscellaneous

There are many other ways to extract or construct features from text, depending on the use that the features are intended for. For example, a *dual representation* of a corpus may be considered, in which features are used to represent terms and not documents. The feature vector for a term t contains one feature for each document, and its value is related to the frequency of t in that document. This representation can be used to analyze which words co-occur frequently and may therefore be related in meaning. Feature construction can also utilize methods from *information extraction*, such as identifying various kinds of named entities (names of persons, places, organizations, etc.) or other interesting bits of information and introducing features which indicate the presence of particular names or other tagged entities in the document.

Cross References

►Document Classification
►Feature Selection in Text Mining
►Kernel Methods
►Support Vector Machine
►Text Mining

Recommended Reading

Brank, J. (2006). Loose phrase string kernels. In *Proceedings of SiKDD, Ljubljana, Slovenia*. Jozef Stefan Institute.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, *41*, 391–407.

Hardoon, D. R., Szedmak, S. R., & Shawe-Taylor, J. R. (2004). Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, *16*(12), 2639–2664.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research, 2*, 419–444.

Mladenić, D. (2002). Learning word normalization using word suffix and context from unlabeled data. *Proceedings of the 19th ICML 1*(8), 427–434.

Mladenić, D., & Grobelnik, M. (2003). Feature selection on hierarchy of web documents. *Decision Support Systems, 35*(1), 45–87.

Plisson, J., Lavrač, N., Mladenić, D., & Erjavec, T. (2008). Ripple down rule learning for automated word lemmatization. *AI Communications, 21*(1), 15–26.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.

Feature Extraction

►Dimensionality Reduction

Feature Reduction

►Feature Selection

Feature Selection

Huan Liu
Arizona State University
Tempe, AZ, USA

Synonyms

Attribute selection; Feature reduction; Feature subset selection; Variable selection; Variable subset selection

Definition

Feature selection is the study of algorithms for reducing dimensionality of data to improve *machine learning* performance. For a dataset with N features and M dimensions (or features, attributes), feature selection aims to reduce M to M' and $M' \leq M$. It is an important and widely used approach to ►dimensionality reduction. Another effective approach is ►*feature extraction*. One of the key distinctions of the two approaches lies at their outcomes. Assuming we have four features F_1, F_2, F_3, F_4, if both approaches result in 2 features, the 2 *selected* features are a subset of 4 original features (say, F_1, F_3), but the 2 *extracted* features are some combination of 4 original features (e.g., $F'_1 = \sum a_i F_i$ and $F'_2 = \sum b_i F_i$, where a_i, b_i are some constants). Feature selection is commonly used in applications where original features need to be retained. Some examples are document categorization, medical diagnosis and prognosis, gene-expression profiling. We focus our discussion on feature selection. The benefits of feature selection are multifold: it helps improve machine learning in terms of predictive accuracy, comprehensibility, learning efficiency, compact models, and effective data collection.

The objective of feature selection is to remove irrelevant and/or redundant features and retain only relevant features. *Irrelevant features* can be removed without affecting learning performance. *Redundant features* are a type of irrelevant features. The distinction is that a redundant feature implies the copresence of another feature; individually, each feature is relevant, but the removal of either one will not affect learning performance.

Motivation and Background

The rapid advance of computer technology and the ubiquitous use of Internet have provided unparalleled opportunities for humans to expand the capabilities in production, services, communications, and research. In this process, immense quantities of high-dimensional data are accumulated, challenging the state-of-the-art machine learning techniques to efficiently produce useful results. Machine learning can benefit from using only relevant data in terms of learning performance (e.g., better predictive accuracy and shortened training time) and learning results such as improved comprehensibility to gain insights and to facilitate validation. At first glimpse, one might think a powerful machine learning algorithm can automatically identify the useful features in its model building process. In effect, removing irrelevant and/or redundant features can affect machine learning. First, let us look at what constitutes effective learning. In essence, the ►hypothesis space is largely constrained by the number of features. Learning can be viewed as searching for a "correct" hypothesis in the hypothesis space. A hypothesis is correct if it is consistent with the training data or the majority of it in the presence of noise, and is expected to perform equally well for the unseen data (or instances that are not present in the training data). In some sense, the more instances we have in the training data, the more constraints there are in helping guide the search for a correct hypothesis.

Broadly speaking, two factors matter most for effective learning: (1) the number of features (M), and (2) the number of instances (N). For a fixed M, a larger N means more constraints and the resulting correct hypothesis is expected to be more reliable. For a fixed N, a decreased M is tantamount to a significantly increased number of instances. Consider the following *thought experiment* for a binary domain of a binary classification problem: F_1, F_2, F_3, F_4 are binary and class C is also binary (e.g., positive or negative). If the training data consists of 4 instances ($N = 4$), it is only a quarter of the total number of possible instances ($2^4 = 16$). The size of the hypothesis space is $2^{2^4} = 65,536$. If only two features are relevant, the size of the hypothesis

space becomes $2^{2^2} = 16$, an exponential reduction of the hypothesis space. Now, the only available 4 instances might suffice for perfect learning if there is no duplicate instance in the reduced training data with two features. And a resulting model of 4 features can also be more complex than that of 2 features. Hence, feature selection can effectively reduce the hypothesis space, or virtually increase the number of training instances, and help create a compact model.

An unnecessarily complex model subjects itself to oversearching an excessively large hypothesis space. Its consequence is that the learned hypothesis *overfits* the training data and is expected to perform poorly when applying the learned model to the unseen data. Another way of describing the relationship between N and M in the context of learning is the so-called *curse of dimensionality*, the need for the *exponential* increase in data size associated with *linearly* adding additional dimensions to a multidimensional space; or the concept of proximity becomes blurry in a high-dimensional space, resulting in degrading learning performance. Theoretically, the reduction of dimensionality can eventuate the exponential shrinkage of hypothesis space.

Structure of the Learning System

The structure of a feature selection system consists of four basic components: input, search, evaluation, and output. The output of any feature selection system can be either a *ranked list* of features or a *subset* of features. For the former, one can select top k highly ranked features depending on the need. In the context of learning, the *input* to a feature selection system is the data which can be (1) supervised – all instances are associated with class labels as in supervised learning; (2) unsupervised – no class labels are available as in unsupervised learning; and (3) some instances have class labels and the rest do not as in semi-supervised learning. To rank the features or select a feature subset can be phrased as a *search* problem in which various search strategies can be employed. Depending on how a feature selection system is working together with a learning system, we can study different *models* of feature selection (Kohavi & John, 1997) such as wrapper, filter, or embedded. An inevitable question about feature selection is whether the removal of features can help machine learning. This necessitates the *evaluation* of feature selection. We will review these aspects of feature selection research next.

Categories of Feature Selection

Feature selection algorithms can be categorized into supervised, unsupervised, and semi-supervised, corresponding to different types of learning algorithms. There has been a substantial gamut of research on *supervised feature selection*. As in supervised machine learning, the data available for feature selection contains class labels. The class information is used as a dominant factor in determining the feature quality. For example, we can simply measure the correlation between a feature (or a subset of features) and the class and select those features with highest correlations. Another way of using the class information is to see if a feature can help differentiate two neighboring instances with different classes: obviously, a relevant feature can, but an irrelevant feature cannot.

Unsupervised feature selection has gained much attention in the recent years. Most data collected are without class labels since labeling data can incur huge costs. The basic principle of unsupervised learning is to cluster data such that similar objects (instances) are grouped together and dissimilar objects are separated. In other words, if we had all similar objects in their corresponding designated clusters, we would have the minimum intracluster distances and the maximum intercluster distances among all objects. For data of high-dimensionality, distance calculation can be a big problem due to the ►curse of dimensionality. One idea is to find features that can promote the data separability. In Dy and Brodley (2004), the goal of unsupervised feature selection is defined as finding the smallest feature subset that best uncovers "interesting natural" clusters from data according to the chosen criterion. A variant of unsupervised feature selection is subspace clustering. It explores the fact that in a high-dimensional space, clusters can often be found in various subspaces of very low dimensionality. Some subspace clustering algorithms are reviewed in Parson, Haque, and Liu (2004).

Unsupervised feature selection is a more loosely constrained problem than supervised feature selection. When a large number of labeled instances are infeasible to obtain, could we use a small number of labeled instances? *Semi-supervised feature selection* attempts to take advantage of both the size of unlabeled data and the labeling information of a small number of labeled instances. In the same spirit of ►semi-supervised learning, semi-supervised feature selection takes advantage of the two biases inherited in labeled

and unlabeled data, respectively, in the hope that the problem of feature selection becomes more constrained. The basic idea is to find features that can not only help group the data, but also encourage to find, among many equally good groups, those groups in which instances of different classes are not in the same group (Zhao & Liu, 2007b).

Searching for Relevant Features

The search for relevant features can be realized in two ways: (1) *feature ranking* – features are ranked according to the intrinsic properties of the data so that top k features can be chosen according to the need or a given threshold; and (2) *subset selection* – a subset of feature is selected from the full set of features, and there is no relevant difference between the features in the selected subset. Subset selection can be carried out in various ways: forward selection – starting with an empty feature subset and adding more iteratively, backward elimination – beginning with a full set of features and eliminating some gradually, and random – the starting subset can be any number which is then adjusted: if this number of features suffices according to some quality measure, it may be decreased, otherwise it should be increased. The exhaustive search is usually too expensive for either forward or backward search. Hence, a sequential search strategy is often adopted. Sequential forward search (SFS) selects one feature at a time; once a feature is selected, it will always be in the selected feature subset and also helps determine which feature should be selected next within the already selected features. Sequential backward search eliminates one feature at a time; once it is ruled out, it will never be considered for selection or inclusion in the selected set of features.

Here we briefly illustrate two effective and efficient algorithms with disparate ideas. One is ReliefF (Robnik-Sikonja & Kononenko, 2003). It selects a feature by checking how effectively it can differentiate the neighboring data points with different classes. A feature's weight is increased if it is effective in doing so. The features are then ranked according to their weights, and the top ranked features are deemed relevant. The second is FCBF (Yu & Liu, 2004), which adds the feature-feature correlation into the selection process with feature-class correlations. The basic idea is that for two features and the class, we can consider not only each feature's correlation with the class, but also the correlation between the two features. If the feature-feature correlation is greater than the smaller of the two feature-class correlations, the feature with smaller feature-class correlation is redundant and can thus be removed.

Models of Feature Selection

Three classic models of feature selection are filter, wrapper, and embedded. Research shows that, generally speaking, even for a classifier with embedded feature selection capability, it can benefit from feature selection in terms of learning performance. A filter model relies on measures about the intrinsic data properties. Mutual information and data consistency are two examples of measures about data properties. A wrapper model involves a learning algorithm (e.g., a classifier, or a clustering algorithm) in determining the feature quality. For instance, if removing a feature does not affect the classifier's accuracy, the feature can be removed. Obviously, this way feature selection is adapted to improving a particular classification algorithm. To determine if the feature should be selected or removed, it needs to build a classifier every time when a feature is considered. Hence, the wrapper model can be quite costly. An embedded model embeds feature selection in the learning of a classifier. A best example can be found in decision tree induction in which, at each branching point, a feature has to be selected first. When feature selection is performed for data preprocessing, filter and wrapper models are often employed. When the purpose of feature selection goes beyond improving learning performance (e.g., classification accuracy), the most applied is the filter model.

Evaluation of Feature Selection

The efficacy of feature selection can be validated via empirical evaluation. Two natural questions related to classification learning are (1) whether using selected features can do as well as using the full set of features, and (2) how to compare two feature selection algorithms when one wants to figure out which is more effective. The first question can be considered as a special form of the second one if we assume the full set of features is selected by a dummy feature selection algorithm that simply selects all given features. We therefore address

only the second question here. When we need to compare two feature selection algorithms (A_1, A_2), if the relevant features are known (the ground truth) as in the experiments using synthetic data, we can directly compare the selected features by A_1 and A_2, respectively, and check which result is closer to the ground truth. In practice, one seldom knows what the relevant features are. A conventional way of evaluating two algorithms is to evaluate the effect of selected features on classification accuracy. It is a two-step procedure: first, selecting features from data D to form D'_i with reduced dimensionality; and second, obtaining estimated predictive accuracy of a classifier on D and D'_i, respectively. Which algorithm is superior can be statistically measured by accuracy difference between A_1 and A_2. If there is no significant difference, one cannot tell which one of the two feature selection algorithms is better; otherwise, the algorithm resulting in better predictive accuracy is better.

Another issue arising from feature selection evaluation is about *feature selection bias*. Using the *same* training data in both feature selection and classification learning can result in this selection bias. According to statistical theory based on regression research, this bias can exacerbate data overfitting and negatively affect classification performance. A recommended practice is to use separate data for feature selection and for learning. In reality, however, separate datasets are rarely used in the selection and learning steps. This is because we often want to use as much data as possible in both selection and learning. It is against this intuition to divide the training data into two datasets leading to the reduced data in either task. The work presented in Singhi and Liu (2006) convincingly demonstrates that in regression, feature selection bias caused by using the same data for feature selection and classification learning does not negatively impact classification as expected.

Feature Selection Development and Applications

The advancement of feature selection research enables us to tackle new challenges. Feature interaction presents a challenge to feature selection. If we define relevance using correlation, a feature by itself might have little correlation with the target concept as in classification learning, but can be very relevant if it is combined with some other features, because the subset can be strongly correlated with the target concept. The unintentional removal of these features can eventuate poor learning performance. It is, in general, computationally intractable to handle feature interaction. In Zhao and Liu (2007a), it is shown that it is feasible to identify interacting features, in the case of using data consistency as a feature quality measure, by designing a special data structure for linear-time backward elimination in terms of M features and by employing an information-theoretic feature ranking heuristic. The authors also point out that the key challenge of employing the ranking heuristic is the *feature order problem* – a lowly ranked feature is more likely to be eliminated first.

Data fusion of multiple data sources presents another challenge for feature selection. Multiple data sources, each with its own features, need to be integrated in order to perform an inference task with the same objective optimally. Instead of selecting the most relevant features from each data source, one now needs to consider selecting *complementary features*. It is also very likely that performing conventional feature selection on the single aggregated dataset by combining all the data sources cannot accomplish the task. The problem of complementary feature selection seems related to that of finding interacting features. It will be worthwhile to examine how both research efforts can bootstrap each other in attacking the two recent challenges.

The recent developments in feature selection witness many new efforts on studying *causal relationships* among features (Guyon, Aliferis, & Elisseeff, 2007) to distinguish actual features and experimental artifacts; on *text feature selection* (Forman, 2007) that were widely employed in thwarting spam emails, automatic sorting of news articles, Web content management, and customer support; on *small data sample* problems that present challenges to reliable estimation of feature quality and detection of feature interactions, and on connecting feature selection and feature extraction. Both feature selection and extraction aim to reduce the dimensionality of the data by removing the nonessential (redundant or noisy) information, but the two areas have been researched largely independently. They can, however, indeed complement each other. On the one hand, feature extraction approaches, such as linear discriminant analysis, are effective for reducing data dimensionality, but suffer from the high computational complexity, especially for high-dimensional data; on the other hand, feature selection algorithms can

handle large-scale data and also lead to easy interpretation of the resulting learning model, but they may not select interacting features. The challenges of high-dimensional data suggest a need for the two to work together.

There is little work in the literature discussing about selecting structural features and sequential features. When data evolve, the variety of data types increases. Semi-structural or structural data now become increasingly common. Consequently, some features in these data may contain a structure (e.g., a hierarchy that defines the relationships between some atomic features). It can be commonly seen in the data with meta data. Clearly, extant feature selection algorithms have to evolve in order to handle structural feature selection. Another area that requires more research attention is the study of sequential features for data streams and for ►time series. They are very different from the types of data that are well studied. Data streams are massive, transient, and often from multiple sources. Time series data present their continuous temporal patterns.

Feature selection research has found its application in many fields where the presence of large (either row-wise or column-wise) volumes of data presents challenges to effective data analysis and processing. High-throughput technologies allow for parallel measurements of massive biological variables describing biological processes. The inordinate number of the biological measurements can contain noise, irrelevance, and redundancy. Feature selection can help focus on relevant biological variables in genomics and proteomics research. The pervasive use of Internet and Web technologies has been bringing about a great number of new services and applications, ranging from recent Web 2.0 applications to traditional Web services where multimedia data are ubiquitous and abundant. Feature selection is widely applied to find topical terms, establish group profiles, assist categorization, simplify descriptions, facilitate personalization and visualization, among others.

Cross References

►Classification
►Clustering
►Cross Validation
►Curse of Dimensionality
►Dimensionality Reduction
►Semi-Supervised Learning

Recommended Reading

Dy, J. G., & Brodley, C. E. (2004). Feature selection for unsupervised learning. *Journal of Machine Learning Research, 5*, 845–889.

Forman, G. (2007). Feature selection for text classification. In H. Liu & Motoda, H. (Eds.), *Computational methods of feature selection*. Boca Raton, FL: Chapman and Hall/CRC Press.

Guyon, I., Aliferis, C., & Elisseeff, A. (2007). Causal feature selection. In *(LM07)*, A longer technical report is available: http://clopinet.com/isabelle/Papers/causalFS.pdf.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research, 3*, 1157–1182.

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence, 97*(1–2), 273–324.

Liu, H., & Motoda, H. (1998). *Feature Selection for knowledge discovery & data mining*. Boston, MA: Kluwer Academic Publishers.

Liu, H., & Motoda, H. (Eds.). (2007). *Computational methods of feature selection*. Boca Raton, FL: Chapman and Hall/CRC Press.

Liu, H., & Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering, 17*(3), 1–12.

Parson, L., Haque, E., & Liu, H. (2004). Subspace clustering for high dimensional data – a review. *ACM SIGKDD Explorations Newsletter Archive special issue on learning from imbalanced datasets*, 6(1): 90–105. ISSN: 1931–0145

Robnik-Sikonja, M., & Kononenko, I. (2003). Theoretical and empirical analysis of Relief and ReliefF. *Machine Learning, 53*, 23–69.

Singhi, S., & Liu, H. (2006). Feature subset selection bias for classification learning. In *Proceeding of the 23rd international conference on machine learning ACM international conference proceeding series*, (Vol. 148, pp. 849–856). Pittsburg, PA. ISBN: 1-59593-383-2

Yu, L., & Liu, H. (2004). Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research, 5*(October), 1205–1224.

Zhao, Z., & Liu, H. (2007a). Searching for interacting features. In *Proceedings of the 20th international joint conference on artificial intelligence*, (pp. 1156–1161). Hydrabad, India.

Zhao, Z., & Liu, H. (2007b). Semi-supervised feature selection via spectral analysis. In *Proceedings of 24th international conference on machine learning* (1151–1157). Corvalis, OR. ISBN: 978-1-59593-793-3

Feature Selection in Text Mining

Dunja Mladenić
Jožef Stefan Insitute
Ljubljana, Slovenia

Synonyms

Dimensionality reduction on text via feature selection

Definition

The term *feature selection* is used in machine learning for the process of selecting a subset of features

(dimensions) used to represent the data (see ►Feature Selection, and ►Dimensionality Reduction). Feature selection can be seen as a part of data pre-processing potentially followed or coupled with feature construction ►Feature Construction in Text Mining, but can also be coupled with the learning phase if embedded in the learning algorithm. An Assumption of feature selection is that we have defined an original feature space that can be used to represent the data, and our goal is to reduce its dimensionality by selecting a subset of original features. The original feature space of the data is then mapped onto a new feature space. *Feature selection in text mining* is addressed here separately due to the specificity of textual data compared to the data commonly addressed in machine learning.

Motivation and Background

Tasks addressed in machine learning on text are often characterized by a high number of features used to represent the data. However, these features are not necessarily all relevant and beneficial for the task, and may slow down the applied methods giving similar results as a much smaller feature set. The main reasons for using feature selection in machine learning are Mladenić 2006: to improve performance, to improve learning efficiency, to provide faster models possibly requesting less information on the original data, and to reduce the complexity of the learned results and enable better understanding of the underlying process.

Feature selection in text mining was applied in a simple form from the start of applying machine learning methods on text data; for instance, feature selection by keeping the most frequent features and learning decision rules ►Rule Learning proposed in Apte, Damerau, and Weiss (1994) or keeping the most informative features for learning decision trees ►Decision Trees or ►Naïve Bayes ►Bayes Rule proposed in Lewis and Ringuette (1994). The reason is that the number of features used to represent text data for machine learning tasks is high, as the basic approach of learning on text defines a feature for each word that occurs in the given text. This can easily result in several tens of thousands of features, compared to several tens or hundreds of features, as commonly observed on most machine learning tasks at the time.

Most methods for feature subset selection that are used on text are very simple compared to the feature selection methods developed in machine learning. They perform a filtering of features assuming feature independence, so that a score is assigned to each feature independently and the features with high scores are selected. However, there are also more sophisticated methods for feature selection on text data that take into account interactions between the features. Embedded feature selection methods were successfully used on text data, either by applying a learning algorithm that has feature selection embedded (pre-processing step) or by inspecting a model generated by such an algorithm to extract feature scores. On the other hand, approaches to feature selection that search a space of all possible feature subsets can be rather time consuming when dealing with a high number of features, and are rarely used on text data.

Structure of Learning System

Feature selection in text mining is mainly used in connection with applying known machine learning and statistical methods on text when addressing tasks such as ►Document Clustering or ►Document Classification. This is also the focus of this chapter. However, we may need to perform some kind of feature selection on different text mining tasks where features are not necessary words or phrases, in which case we should reconsider the appropriate feature selection methods in the light of the task properties, including the number and type of features.

As already pointed out, the common way of document text representation is by defining a feature for each word in the document collection and feature selection by assuming feature independence, assigning score to the features, and selecting features with high scores. Scoring of individual features is performed either in an unsupervised way, ignoring the class information, or in a supervised way, taking into account class information. Surprisingly both kind of approaches have been shown to perform comparably on document classification tasks, even though supervised scoring uses more information. Here we discuss several feature scoring measures and their performance on document classification, as reported in different researcher papers.

One of the first scoring measures used on text data is scoring by the number of documents that contain a particular word. This was applied after removing very frequent words, as given in a standard "stop-list" for English. An alternative is scoring by frequency – that is, by the number of times a feature occurs in a document collection. Both were shown to work well in document classification Mladenić & Grobelnik (2003); Yang & Pedersen (1997).

Information gain is commonly used in decision tree induction (Quinlan, 1993). It was reported to work well as a feature scoring measure on text data (Yang & Pedersen, 1997) in some domains (news articles of in a collection named Reuters-22173, abstracts of medical articles in a subset of the MEDLINE collection), where a multiclass problem was addressed using the nearest neighbor algorithm ►Nearest Neighbor. The same feature scoring almost completely failed when using Naïve Bayes ►Bayes Rule on a binary classification problem on a hierarchical topic taxonomy of Web pages (Mladenić & Grobelnik, 2003). This difference in performance can be partially attributed to the classification algorithm and domain characteristics.

It is interesting to notice that information gain takes into account all values for each feature. In the case of document classification, these are two values: occurs or does not occur in a document. On the other hand, expected cross entropy as used on text data (Koller & Sahami, 1997; Mladenić & Grobelnik, 2003) is similar in nature to information gain, but only uses the situation when the feature occurred in a document. Experiments on classifying document into a hierarchical topic taxonomy (Mladenić & Grobelnik, 2003) have show that this significantly improves performance. Expected cross entropy is related to information gain as follows: InfGain(F) = CrossEntropyTxt(F) + CrossEntropyTxt(F), where F is a binary feature (usually representing a word's occurrence).

The odds ratio was reported to outperform many other measures (Mladenić & Grobelnik, 2003) in combination with Naïve Bayes, used for document classification on data with highly imbalanced class distribution. A characteristic of Naïve Bayes used for text classification is that, once the model has been generated, the classification is based on the features that occur in a document to be classified. This means that an empty document will be classified into the majority class. Consequently, having a highly imbalanced class distribution, if we want to identify documents from the under-represented class value, we need to have a model sensitive to the features that occur in such documents. If most of the selected features are representative for the majority class value, the documents from other classes will be almost empty when represented using the selected features.

Experimental comparison of different feature selection measures in combination with the support vector machines ►Support Vector Machines classification algorithm (SVM) on news articles from the Reuters-2000 collection (Brank, Grobelnik, Milič-Frayling, & Mladenić, 2002) has shown that using all or almost all the features yields the best performance. The same finding was confirmed in experimental evaluation of different feature selection measures on a number of text classification problems (Forman, 2003). In addition, in Forman (2003) a new feature selection measure was introduced: Bi-Normal Separation, which was reported to improve the performance of SVM, especially with problems where the class distribution is highly imbalanced. Interestingly, they also report that information gain is outperforming the other tested measures in the situation when using only a small number of selected features (20–50 features).

Another feature scoring measure for text data, called the Fisher Index, was proposed as part of a document retrieval system based on organizing large text databases into hierarchical topic taxonomies (Chakrabarti, Dom, Agrawal, & Raghavan, 1998). Similar to Mladenić (1998), for each internal node in the topic taxonomy, a separate feature subset is used to build a Naïve Bayes model for that node. This is sometimes referred to as local feature selection or, alternatively, context sensitive feature selection. The feature set used in each node is relatively small and tuned to the node context.

What follows are formulas of the described scoring measures as given in Mladenić and Grobelnik (2003).

$$\begin{aligned}\text{InfGain}(F) = {} & P(F)\sum_i P(C_i|F)\log(P(C_i|F)/P(C_i)) \\ & + P(\neg F)\sum_i P(C_i|\neg F) \\ & \times \log P(C_i|\neg F)/P(C_i))\end{aligned}$$

$$\begin{aligned}\text{CrossEntropyTxt}(F) = {} & P(F)\sum_i P(C_i|F) \\ & \log(P(C_i|F)/P(C_i))\end{aligned}$$

$$\text{MutualInfoTxt}(F) = \sum_i P(C_i)\log(P(F|C_i)/P(F))$$
$$\text{OddsRatio}(F) = \log(P(F|C_{\text{pos}})(1-P(F|C_{\text{neg}})))$$
$$-\log((1-P(F|C_{\text{pos}}))P(F|C_{\text{neg}}))$$
$$\text{Bi-NormalSeparation}(F) = Z^{-1}(P(F|C_{\text{pos}}))$$
$$-Z^{-1}(P(F|C_{\text{neg}}))$$
$$\text{FisherIndexTxt}(F) = (\sum_{\text{pos,neg}}(P(F|C_{\text{pos}})$$
$$-P(F|C_{\text{neg}}))^2)/\sum_{\text{Ci}\varepsilon\text{pos,neg}}|C_i|^{-1}$$
$$\times\sum_{\text{d}\varepsilon\text{Ci}}(n(F,d)$$
$$-P(F|C_i))2$$

where $P(F)$ is the probability that feature F occurred, $\neg F$ means that the feature does not occur, $P(C_i)$ is the probability of the ith class value, $P(C_i|F)$ is the conditional probability of the ith class value given that feature F occurred, $P(F|C_i)$ is the conditional probability of feature occurrence given the ith class value, $P(F|C_{\text{pos}})$ is the conditional probability of feature F occurring given the class value "positive," $P(F|C_{\text{neg}})$ is the conditional probability of feature F occurring given the class value "negative," $Z^{-1}(x)$ is the standard Normal distribution's inverse cumulative probability function (z-score), $|C_i|$ is the number of documents in class C_i, and $n(F, d)$ is 1 if the document d contains feature F and 0 otherwise.

As already highlighted in text classification, most of the feature selection methods evaluate each feature independently. A more sophisticated approach is proposed in Brank et al. (2002), where a linear SVM is first trained using all the features, and the induced model is then used to score the features (weight assigned to each feature in the normal to the induced hyper plane is used as a feature score). Experimental evaluation using that feature selection in combination with SVM, Perceptron, and Naïve Bayes has shown that the best performance is achieved by SVM when using almost all the features. The experiments have confirmed the previous findings on feature subset selection improving the performance of Naïve Bayes, but the overall performance is lower than using SVM on all the features.

Much the same as in Brank et al. (2002), feature selection was performed using a linear SVM to rank the features in Bi, Bennett, Embrechts, Breneman, and Song (2003). However, the experiments in Bi et al. (2003) were performed on a regression problem, and the final model was induced using a nonlinear SVM. The feature selection was shown to improve performance.

Distributional clustering of words with an agglomerative approach (words are viewed as distributions over document categories) is used for dimensionality reduction via feature construction (Bekkerman, El-Yaniv, Tishby, & Winter, 2003) that preserves the mutual information between the features as much as possible. This representation was shown to achieve comparable or better results than the bag-of-words document representation using feature selection based on Mutual information for text; a linear SVM was used as the classifier. A related approach, also based on preserving the mutual information between the features (Globerson & Tishby, 2003), finds new dimensions by using an iterative projection algorithm instead of clustering. It was shown to achieve performance comparable to the bag-of-words representation with all the original features, using significantly less features (e.g., on one dataset, four constructed features achieved 98% of performance of 500 original features) using the linear SVM classifier.

Divisive clustering for feature construction (Dhillon, Mallela, & Kumar, 2003) was shown to outperform distributional clustering when used for dimensionality reduction on text data. The approach uses the Kullback-Leibler divergence as a distance function, and minimizes within-cluster divergence while maximizing between-cluster divergence. Experiments on two datasets have shown that this dimensionality reduction slightly improves the performance of Naïve Bayes (compared to using all the original features), outperforming the agglomerative clustering of words combined with Naïve Bayes and achieving considerably higher classification accuracy for the same number of features than feature subset selection using information gain or mutual information (in combination with Naïve Bayes or SVM).

Recommended Reading

Apte, C., Damerau, F., & Weiss, S. M. (1994). Toward language independent automated learning of text categorization models. In *Proceedings of the 17th annual International ACM SIGIR conference on research and development in Information Retrieval*, pp. 23–30, Dublin, Ireland, 1994.

Brank, J., Grobelnik, M., Milič-Frayling, N., & Mladenić, D. (2002). Feature selection using support vector machines. In A. Zanasi (Ed.), *Data mining III* (pp. 261–273). Southampton, UK: WIT.

Bi, J., Bennett, K. P., Embrechts, M., Breneman, C. M., & Song, M. (2003). Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, *3*, 1229–1243.

Bekkerman, R., El-Yaniv, R., Tishby, N., & Winter, Y. (2003). Distributional word clusters vs. words for text categorization. *Journal of Machine Learning Research*, *3*, 1183–1208.

Chakrabarti, S., Dom, B., Agrawal, R., & Raghavan, P. (1998). Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *The VLDB Journal*, *7*, 163–178.

Dhillon, I., Mallela, S., & Kumar, R. (2003). A divisive information-theoretic feature clustering algorithm for text classification. *Journal of Machine Learning Research*, *3*, 1265–1287.

Forman, G. (2003). An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, *3*, 1289–1305.

Globerson, A., & Tishby, N. (2003). Sufficient dimensionality reduction. *Journal of Machine Learning Research*, *3*, 1307–1331.

Koller, D., & Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Proceedings of the 14th international conference on machine learning ICML'97* (pp. 170–178). Nashrille, TN.

Lewis, D. D., & Ringuette, M. (1994). Comparison of two learning algorithms for text categorization. In *Proceedings of the 3rd annual symposium on document analysis and information retrieval SDAIR-1994*. Las Vegas, NV.

Mladenić, D. (1998). Feature subset selection in text-learning. In *Proceedings of the 10th European conference on machine learning ECML'98*. Chemnitz, Germany.

Mladenić, D. (2006). Feature selection for dimensionality reduction. In C. Saunders, S. Gunn, J. Shawe-Taylor, & M. Grobelink (Eds.), *Subspace, Latent Structure and Feature Selection: Statistical and Optimization Perspectives Workshop: Lecture notes in computer science* (Vol. 3940, pp. 84–102). Berlin, Heidelberg: Springer.

Mladenić, D., & Grobelnik, M. (2003). Feature selection on hierarchy of web documents. *Journal of Decision Support Systems*, *35*, 45–87.

Quinlan, J. R. (1993). Constructing decision tree. In *C4.5: Programs for machine learning*. San Francisco: Morgan Kaufman Publishers.

Yang, Y., & Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Proceedings of the 14th international conference on machine learning ICML'97* (pp. 412–420). Las Vegas, NV.

Feature Subset Selection

►Feature Selection

Feedforward Recurrent Network

►Simple Recurrent Network

Finite Mixture Model

►Mixture Model

First-Order Logic

Peter A. Flach
University of Bristol
Bristol, UK

Synonyms

First-order predicate calculus; First-order predicate logic; Predicate calculus; Predicate logic; Resolution

Definition

First-order predicate logic – first-order logic for short – is the logic of properties of, and relations between, objects and their parts. Like any logic, it consists of three parts: *syntax* governs the formation of *well-formed formulae*, *semantics* ascribes meaning to well-formed formulae and formalizes the notion of *deductive consequence*, and *proof procedures* allow the inference of deductive consequences by syntactic means. A number of variants of first-order logic exist, mainly differing in their syntax and proof systems. In machine learning, the main use of first-order logic is in ►learning from structured data, ►inductive logic programming and ►relational data mining.

Motivation and Background

The interest in logic arises from a desire to formalize human, mathematical and scientific reasoning, and goes back to at least the Greek philosophers. Aristotle devised a form of propositional reasoning called *syllogisms* in the fourth century BC. Aristotle was held in very high esteem by medieval scholars, and so further significant advances were not made until after the Middle Ages. Leibniz wrote of an "algebra of thought"

and linked reasoning to calculation in the late seventeenth century. Boole and De Morgan developed the algebraic point of view in the mid-nineteenth century.

Universally quantified variables, which form the main innovation in first-order logic as compared to ▶propositional logic, were invented by Gottlob Frege in his *Begriffsschrift* ("concept notation") from 1879, and independently by Charles Sanders Peirce in 1885, who introduced the notation $\prod_x$ and $\sum_x$ for universal and existential quantification. Frege's work went largely unnoticed until it was developed further by Alfred North Whitehead and Bertrand Russell in their *Principia Mathematica* (1903). Seminal contributions were made, among many others: by Giuseppe Peano, who axiomatized number theory and introduced the notation (x) and $\exists x$; by Kurt Gödel, who established the completeness of first-order logic as well as the incompleteness of any system incorporating Peano arithmetic; by Alonzo Church, who proved that first-order logic is undecidable, and who introduced λ-calculus, a form of ▶higher-order logic that allows quantification over predicates and functions (as opposed to first-order logic, which only allows quantification over objects); and by Alfred Tarski, who pioneered logical semantics through model theory, and the notion of logical consequence. The now universally accepted notation $\forall x$ was introduced by Gerhard Gentzen.

Logic plays an important role in any approach to symbolic AI that employs a formal language for knowledge representation and inference. A significant, relatively recent development was the introduction of logic programming languages such as ▶Prolog, which turn logical inference into computation. In machine learning, the use of a first-order language is essential in order to handle domains in which objects have inherent structure; the availability of Prolog as a common language and programming platform gave rise to the field of inductive logic programming.

Theory

Syntax

A first-order logical language is built from *constant symbols*, *variable symbols*, *predicate symbols* and *function symbols*; the latter two kinds of symbols have an associated *arity*, which is the number of arguments they take. *Terms* are either constant symbols, variable symbols, or of the form $f(t_1, \ldots, t_n)$ where f is a function symbol with arity n, and $t_1, \ldots, t_n$ is a sequence of n terms. Using the logical connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction) and $\rightarrow$ (material implication) and the quantifiers $\forall$ (universal quantifier) and $\exists$ (existential quantifier), well-formed formulae or *wffs* are defined recursively as follows: (1) if P is a predicate symbol with arity n, and $t_1, \ldots, t_n$ is a sequence of n terms, then $P(t_1, \ldots, t_n)$ is a wff, also referred to as an atomic formula or *atom*; (2) if ϕ_1 and ϕ_2 are wff's, then $(\neg\phi_1)$, $(\phi_1 \wedge \phi_2)$, $(\phi_1 \vee \phi_2)$ and $(\phi_1 \rightarrow \phi_2)$ are wffs; (3) if x is a variable and ϕ is a wff, then $(\forall x : \phi)$ and $(\exists x : \phi)$ are wffs; (4) nothing else is a wff. Brackets are usually dropped as much as it is possible without causing confusion.

Example 1 *Let "man," "single," and "partner" be two unary and one binary predicate symbol, respectively, and let "x" and "y" be variable symbols, then the following is a wff ϕ expressing that men who are not single have a partner:*

$$(\forall x : (man(x) \wedge (\neg single(x))) \rightarrow (\exists y : partner(x, y))).$$

Assuming that $\neg$ binds strongest, then $\wedge$, then $\rightarrow$, the brackets can be dropped:

$$\forall x : man(x) \wedge \neg single(x) \rightarrow \exists y : partner(x, y).$$

A *propositional language* is a special case of a predicate-logical language, built only from predicate symbols with arity 0, referred to as *proposition symbols* or propositional atoms, and connectives. So, for instance, assuming the proposition symbols "*man*," "*single*" and "*has_partner*," the following is a propositional wff: $man \wedge \neg single \rightarrow has\_partner$. The main difference is that in propositional logic references to objects cannot be expressed and therefore have to be understood implicitly.

Semantics

First-order wffs express statements that can be true or false and so a first-order semantics consists in constructing a mapping from wffs to truth-values, given an interpretation, which is a possible state of affairs in the domain of discourse, mapping constant, predicate and

function symbols to elements, relations and functions in and over the domain. To deal with variables, a valuation function is employed. Once this mapping is defined, the meaning of a wff consists in the set of interpretations in which the wff maps to true, also called its models. The intuition is that the more "knowledge" a wff contains, the fewer models it has. The key notion of logical consequence is then defined in terms of models: one wff is a logical consequence of another if the set of models of the first contains the set of models of the second; hence the second wff contains at least the same, if not more, knowledge than the first.

Formally, a *predicate-logical interpretation*, or *interpretation* for short, is a pair (D, i), where D is a non-empty *domain* of individuals, and i is a function assigning to every constant symbol an element of D, to every function symbol with arity n a mapping from D^n to D, and to every predicate symbol with arity n a subset of D^n, called the *extension* of the predicate. A *valuation* is a function v assigning to every variable symbol an element of D.

Given an interpretation $I = (D, i)$ and a valuation v, a mapping i_v from terms to individuals is defined as follows: (1) if t is a constant symbol, $i_v(t) = i(t)$; (2) if t is a variable symbol, $i_v(t) = v(t)$; (3) if t is a term $f(t_1, \ldots, t_n)$, $i_v(t) = i(f)(i_v(t_1), \ldots, i_v(t_n))$. The mapping is extended to a mapping from wffs to truthvalues as follows: (4) if ϕ is an atom $P(t_1, \ldots, t_n)$, $i_v(\phi) = i(P)(i_v(t_1), \ldots, i_v(t_n))$; (5) $i_v(\neg\phi) = T$ if $i_v(\phi) = F$, and F otherwise; (6) $i_v(\phi_1 \wedge \phi_2) = T$ if $i_v(\phi_1) = T$ and $i_v(\phi_2) = T$, and F otherwise; (7) $i_v(\forall x : \phi) = T$ if $i_{v_{x \to d}}(\phi) = T$ for all $d \in D$, and F otherwise, where $v_{x \to d}$ is v except that x is assigned d. The remaining connectives and quantifier are evaluated by rewriting: (8) $i_v(\phi_1 \vee \phi_2) = i_v(\neg(\neg\phi_1 \wedge \neg\phi_2))$; (9) $i_v(\phi_1 \to \phi_2) = i_v(\neg\phi_1 \vee \phi_2)$; (10) $i_v(\exists x : \phi) = i_v(\neg\forall x : \neg\phi)$.

An interpretation I *satisfies* a wff ϕ, notation $I \models \phi$, if $i_v(\phi) = T$ for all valuations v; we say that I is a *model* of ϕ, and that ϕ is *satisfiable*. If all models of a set of wffs Σ are also models of ϕ, we say that Σ *logically entails* ϕ or ϕ is a *logical consequence* of Σ, and write $\Sigma \models \phi$. If $\Sigma = \varnothing$, ϕ is called a *tautology* and we write $\models \phi$. A wff ψ is a *contradiction* if $\neg\psi$ is a tautology. Contradictions do not have any models, and consequently $\psi \models \alpha$ for any wff α. The *deduction theorem* says that $\Sigma \models \alpha \to \beta$ if and only if $\Sigma \cup \{\alpha\} \models \beta$. Another useful fact is that, if $\Sigma \cup \{\neg\gamma\}$ is a contradiction, $\Sigma \models \gamma$; this gives rise to a proof technique known as *Reductio ad absurdum* or *proof by contradiction* (see below).

Example 2 *We continue the previous example. Let* $D = \{Peter, Paul, Mary\}$*, and let the function* i *be defined as follows:* $i(man) = \{Peter, Paul\}$*;* $i(single) = \{Paul\}$*;* $i(partner) = \{(Peter, Mary)\}$*. We then have that the interpretation* $I = (D, i)$ *is a model for the wff* ϕ *above. On the other hand,* I *doesn't satisfy* $\psi = \forall x : \exists y : partner(x, y)$*, and therefore* $\phi \not\models \psi$*. However, the reverse does hold: there is no interpretation that satisfies* ψ *and not* ϕ*, and therefore* $\psi \models \phi$*.*

In case of a propositional logic this semantics can be considerably simplified. Since there are no terms the domain D plays no role, and an interpretation simply assigns truth-values to proposition symbols. Wffs can then be evaluated using rules (5–6) and (8–9). For example, if $i(man) = T$, $i(single) = T$ and $i(has\_partner) = T$, then $i(man \wedge \neg single \to has\_partner) = T$ (if this seems counter-intuitive, this is probably because the reader's knowledge of the domain suggests another wff $\neg(single \wedge has\_partner)$, which is false in this particular interpretation).

Proofs

A *proof procedure* consists of a set of *axioms* and a set of *inference rules*. Given a proof procedure P, we say that ϕ is *provable* from Σ and write $\Sigma \vdash_P \phi$ if there exists a finite sequence of wffs $\phi_1, \phi_2, \ldots, \phi_{n-1}, \phi$ which is obtained by successive applications of inference rules to axioms, *premisses* in Σ, and/or previous wffs in the sequence. Such a sequence of wffs, if it exists, is called a *proof* of ϕ from Σ. A proof procedure P is *sound*, with respect to the semantics established by predicate-logical interpretations, if $\Sigma \models \phi$ whenever $\Sigma \vdash_P \phi$; it is *complete* if $\Sigma \vdash_P \phi$ whenever $\Sigma \models \phi$. For a sound and complete proof procedure for first-order predicate logic, see e.g., Turner, 1984, p. 15.

A set of wffs Σ is *consistent*, with respect to a proof procedure P, if not both $\Sigma \vdash_P \phi$ and $\Sigma \vdash_P \neg\phi$ for some wff ϕ. Given a sound and complete proof procedure, the proof-theoretic notion of consistency coincides with the semantic notion of satisfiability. In particular, if we can prove that $\Sigma \cup \{\neg\gamma\}$ is inconsistent, then we know that $\Sigma \cup \{\neg\gamma\}$ is not satisfiable, hence a contradiction, and thus $\Sigma \models \gamma$. This still holds if the proof procedure

is only complete in the weaker sense of being able to demonstrate the inconsistency of arbitrary sets of wffs (see the resolution inference rule, below).

Example 3 *One useful inference rule for predicate logic replaces a universally quantified variable with an arbitrary term, which is called* universal elimination. *So, if "c" is a constant symbol in our language, then we can infer*

$$man(c) \wedge \neg single(c) \rightarrow \exists y : partner(c, y)$$

from ϕ above by universal elimination. Another inference rule, which was called Modus Ponens *by Aristotle, allows us to infer β from α and $\alpha \rightarrow \beta$. So, if we additionally have $man(c) \wedge \neg single(c)$, then we can conclude*

$$\exists y : partner(c, y)$$

by Modus Ponens. This rule is also applicable to propositional logic. An example of an axiom is $c = c$ for any constant symbol c (strictly speaking this is an axiom schema, giving rise to an axiom for every constant symbol in the language).

Programming in Logic

Syntax, semantics and proof procedures for first-order logic can be simplified and made more amenable to computation if we limit the number of ways of expressing the same thing. This can be achieved by restricting wffs to a normal form called *prenex conjunctive normal form* (PCNF). This means that all quantifiers occur at the start of the wff and are followed by a conjunction of disjunctions of atoms and negated atoms, jointly called *literals*. An example of a formula in PCNF is

$$\forall x : \exists y : \neg man(x) \vee single(x) \vee partner(x, y).$$

This formula is equivalent to the wff ϕ in Example 1, in the sense that it has the same set of models, and so either one logically entails the other. Every first-order wff can be transformed into a logically equivalent formula in PCNF, which is unique up to the order of conjuncts and disjuncts. A transformation procedure can be found in Flach (1994).

PCNF can be further simplified if we use function symbols instead of existential quantifiers. For instance, instead of $\exists y : partner(x, y)$, we can say $partner(x, partner\_of(x))$, where *partner_of* is a unary function symbol called a *Skolem function*, after the Norwegian logician Thoralf Skolem. The two statements are not logically equivalent, as the second entails the first but not vice versa, but this difference is of little practical consequence. Since all variables are now universally quantified the quantifiers are usually omitted, leading to *clausal form*:

$$\neg man(x) \vee single(x) \vee partner(x, partner\_of(x)).$$

To sum up, a wff in clausal form is a conjunction of disjunctions of literals, of which the variables are implicitly universally quantified. The individual disjunctions are called clauses.

Further simplifications include dispensing with equality, which means that terms involving function symbols, such as $partner\_of(c)$, are not evaluated and in effect treated as names of objects (in this case, the function symbols are called *functors* or *data constructors*). Under this assumption each *ground term* (a term without variables) denotes a different object, which means that we can take the set of ground terms as the domain D of an interpretation; this is called a *Herbrand interpretation*, after the French logician Jacques Herbrand.

The main advantage of clausal logic is the existence of a proof procedure consisting of a single inference rule and no axioms. This inference rule, which is called *resolution*, was introduced by Robinson (1965). In propositional logic, given two clauses $P \vee Q$ and $\neg Q \vee R$ containing *complementary* literals Q and $\neg Q$, resolution infers the *resolvent* $P \vee R$ (P and/or R may themselves contain several disjuncts). For instance, given $\neg man \vee single \vee has\_partner$ and $man \vee woman$, we can infer $woman \vee single \vee has\_partner$ by resolution. In first-order logic, Q and $\neg Q'$ are complementary if Q and Q' are *unifiable*, i.e., there exists a *substitution* θ of terms for variables such that $Q\theta = Q'\theta$, where $Q\theta$ denotes the application of substitution θ to Q; in this case, the resolvent of $P \vee Q$ and $\neg Q' \vee R$ is $P\theta \vee R\theta$. For instance, from the following two clauses:

$$\neg man(x) \vee single(x) \vee partner(x, partner\_of(x))$$

$$\neg single(father\_of(c))$$

we can infer

$$\neg man(father\_of(c)) \vee partner(father\_of(c), partner\_of(father\_of(c))).$$

The resolution inference rule is sound but not complete: for instance, it is unable to produce tautologies such as $man(c) \vee \neg man(c)$ if no clauses involving the predicate *man* are given. However, it is *refutation-complete*, which means it can demonstrate the unsatisfiability of any set of clauses by deriving the *empty clause*, indicated by $\square$. For instance, $man(c) \wedge \neg man(c)$ is a wff consisting of two clauses which are complementary literals, so by resolution we infer the empty clause in one step.

Refutation by resolution is the way in which queries are answered in the logic programming language Prolog. Prolog works with a subset of clausal logic called *Horn logic*, named after the logician Alfred Horn. A *Horn clause* is a disjunction of literals with at most one positive (un-negated) literal; Horn clauses can be further divided into *definite clauses*, which have one positive literal, and *goal clauses* which have none A Prolog program consists of definite clauses, and a goal clause functions as a procedure call. Notice that resolving a goal clause with a definite clause result in another goal clause, because the positive literal in the definite clause (also called its *head*) must be one of the complementary literals. The idea is that the resolution step reformulates the original goal into a new goal that is one step closer to the solution. A refutation is then a sequence of goals $G, G_1, G_2, \ldots, G_n$ such that G is the original goal, each G_i is obtained by resolving G_{i-1} with a clause from the program P, and $G_n = \square$. Such a refutation demonstrates that $P \cup \{G\}$ is inconsistent, and therefore $P \models \neg G$.

Finding a refutation amounts to a search problem, because there are typically several program clauses that could be resolved against the current goal. Virtually all Prolog interpreters apply a depth-first search procedure, searching the goal literals left-to-right and the program clauses top-down. Once a refutation is found the substitutions collected in all resolution steps are composed to obtain an *answer substitution*. One unique feature of logic programming is that a goal may have more than one (or, indeed, less than one) refutation and answer substitution from a given program.

Example 4 *Consider the following Prolog program:*

```
peano_sum(0,Y,Y).
peano_sum(s(X),Y,s(Z)):-
    peano_sum(X,Y,Z).
```

This program defines addition in Peano arithmetic. We follow Prolog syntax: variables start with an uppercase letter, and `:-` *stands for reversed implication* $\leftarrow$ *or "if." The unary functor* `s` *represents the successor function. So the first rule reads "the sum of 0 and an arbitrary number y is y," and the second rule reads "the sum of* $x + 1$ *and* y *is* $z + 1$ *if the sum of* x *and* y *is* z*."*

The goal `:-peano_sum(s(0),s(s(0)),Q)` *states "there are no numbers q such that* $1 + 2 = q$*." We first resolve this goal with the second program clause to obtain* `:-peano_sum(0,s(s(0)),Z)` *under the substitution* {`Q/s(Z)`}*. This new goal states "there are no numbers z such that* $0 + 2 = z$*." It is resolved with the first clause to yield the empty clause under the substitution* {`Y/s(s(0))`, `Z/s(s(0))`}*. The resulting answer substitution is* {`Q/s(s(s(0)))`}*, i.e.,* $q = 3$*.*

As another example, goal `:-peano_sum(A,B,s(s(0)))` *states "there are no numbers a and b such that* $a + b = 2$*." This goal has three refutations: one involving the first clause only, yielding the answer substitution* {`A/0`, `B/s(s(0))`}*; one involving the second clause then the first, resulting in* {`A/s(0)`, `B/s(0)`}*; and the third applying the second clause twice followed by the first, yielding* {`A/s(s(0))`, `B/0`}*. Prolog will return these three answers in this order.*

Induction in first-order logic amount to reconstructing a logical theory from some of its logical consequences. For techniques to induce a Prolog program given examples such as `peano_sum(s(0),s(0),s(s(0)))`, see inductive logic programming.

For general introductions to logic and its use in Artificial Intelligence, see Genesereth and Nilsson (1987) and Turner (1984). Kowalski's classic text *Logic for problem solving* focusses on clausal logic and resolution theorem proving (Kowalski, 1979). For introductions to Prolog programming, see Bratko (2001) and Flach (1994).

Cross References

►Abduction
►Entailment
►Higher-Order Logic
►Hypothesis Language
►Inductive Logic Programming
►Learning from Structured Data
►Logic Program
►Propositionalization
►Relational Data Mining

Recommended Reading

Bratko, I. (2001). *Prolog programming for artificial intelligence* (3rd ed.). Boston: Addison Wesley.
Flach, P. (1994). *Simply logical: Intelligent reasoning by example*. New York: Wiley.
Genesereth, M., & Nilsson, N. (1987). *Logical foundations of artificial intelligence*. San Francisco: Morgan Kaufmann.
Kowalski, R. (1979). *Logic for problem solving*. New York: North-Holland.
Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM, 12*(1), 23–41.
Turner, R. (1984). *Logics for artificial intelligence*. Chichester: Ellis Horwood.

First-Order Predicate Calculus

►First-Order Logic

First-Order Predicate Logic

►First-Order Logic

First-Order Regression Tree

Synonyms

Logical regression tree; Relational regression tree

Definition

A first-order regression tree can be defined as follows:

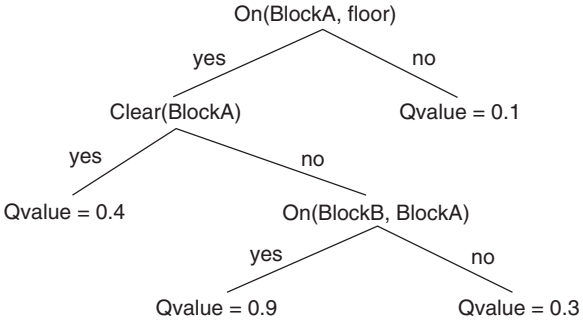
Definition 1 (First-Order Regression Tree) *A first-order regression tree is a binary tree in which*

- *Every internal node contains a test which is a conjunction of first-order literals.*
- *Every leaf (terminal node) of the tree contains a real valued prediction.*

An extra constraint placed on the first-order literals that are used as tests in internal nodes is that a variable that is introduced in a node (i.e., it does not occur in higher nodes) does not occur in the right subtree ofl the node.

Figure 1 gives an example of a first-order regression tree. The test in a node should be read as the existentially quantified conjunction of all literals in the nodes in the path from the root of the tree to that node. In the left subtree of a node, the test of the node is added to the conjunction, for the right subtree, the negation of the test should be added. For the example state description of Fig. 2, the tree would predict a *Qvalue* = 0.9, since there exists no block that is both on the floor and clear, but there is a block which is on the floor and has another block on top of it. To see this, substitute BlockA in the tree with 2 (or 4) and BlockB with 1 (or 4).

The constraint on the use of variables stems from the fact that variables in the tests of internal nodes are existentially quantified. Suppose a node introduces a new variable X. Where the left subtree of a node corresponds to the fact that a substitution for X has been found to make the conjunction true, the right side corresponds to the situation where no substitution for X exists, i.e.,



First-Order Regression Tree. Figure 1. A relational regression tree

```
on(1,2).        clear(1).
on(2,floor).    clear(3).
on(3,4).        clear(floor).
on(4,floor).
```

First-Order Regression Tree. Figure 2. State description

there is no such X. Therefore, it makes no sense to refer to X in the right subtree.

Cross References

►First-Order Rule
►Inductive Logic Programming
►Relational Reinforcement Learning

F-Measure

A measure of information retrieval performance. See ►Precision and Recall.

Foil

►Rule Learning

Formal Concept Analysis

Gemma C. Garriga
Universite Pierre et Marie Curie
Paris, France

Definition

Formal concept analysis is a mathematical theory of concept hierarchies that builds on order theory; it can be seen as an unsupervised machine learning technique and is typically used as a method of knowledge representation. The approach takes an input binary relation (binary matrix) specifying a set of objects (rows) and a set of attributes for those objects (columns), finds the natural concepts described in the data, and then organizes the concepts in a partial order structure or Hasse diagram. Each concept in the final diagram is a pair of sets of objects and attributes that are maximally contained one in each other.

Theory

The above intuition can be formalized through a Galois connection as follows. Let R be the binary relation between a set of objects and a set of attributes, that is, $R \subseteq \mathcal{O} \times \mathcal{A}$. Two mappings $\alpha : \mathcal{O} \mapsto \mathcal{A}$ and $\beta : \mathcal{A} \mapsto \mathcal{O}$ are defined so that the operator $\alpha(O)$, for some $O \subseteq \mathcal{O}$, returns the maximal set of attributes common to all objects in O; dually, the operator $\beta(A)$, for some $A \subseteq \mathcal{A}$, returns the maximal set of objects containing all attributes in A. These two mappings induce a Galois connection between the powerset of objects and the powerset of attributes, that is, they satisfy $O \subseteq \beta(A) \Leftrightarrow A \subseteq \alpha(O)$ for a set of objects O and a set of attributes A.

From here, a formal concept is a pair of sets of objects and attributes (O, A) from the binary relation that satisfy $\alpha(O) = A$ and $\beta(A) = O$. Typically, O is called the extent of the concept and A the intent of the concept. Note that concepts can be interpreted from the geometrical point of view, they are maximal rectangles of ones (not necessarily consecutive) in the input binary table R. The organization of all the formal concepts in a Hasse diagram is called the concept lattice. This lattice corresponds to a partial order structure of concepts where edges between concepts correspond to the standard inclusion of the sets.

A small toy example in Figs. 1 and 2 illustrates the formal concepts and their organization in a Hasse diagram.

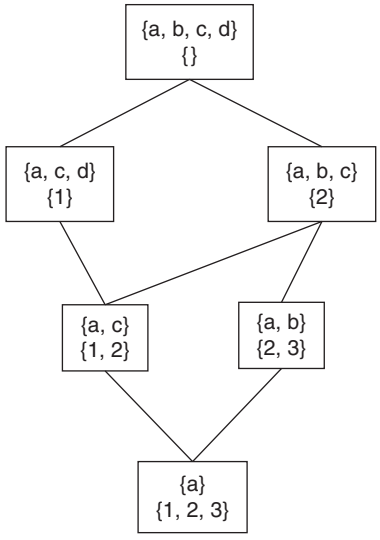
Motivation and Background

Formal concept analysis has been applied to a variety of disciplines, from psychology, sociology, biology, medicine, linguistics, or industrial engineering, to cite some, for the interactive exploration of implicit and explicit structures in the data.

From the point of view of machine learning and data mining, the connection between the formal concepts of the lattice and the so-called, closed sets of items is remarkable. Closed sets of items appear in the context of ►constraint-based mining, in which the user provides

	a	b	c	d
1	1	0	1	1
2	1	1	1	0
3	1	1	0	0

Formal Concept Analysis. Figure 1. A binary relation $R \subseteq \{1, 2, 3\} \times \{a, b, c, d\}$



Formal Concept Analysis. Figure 2. Concepts of the relation *R* organized in a Hasse diagram

restraints that guide a search of patterns in the data. They are maximal sets of attributes occuring frequently in the data; they correspond to a compacted representation of the frequent sets from ►frequent itemset mining. It is well known that closed sets correspond exactly to the intents of the concepts derived via formal concept analysis, and therefore, from the formal concepts it is possible to construct bases of minimal nonredundant sets of association rules from which all other rules holding in the data can be derived.

Also, formal concept analysis has been typically seen as a type of conceptual ►clustering. Each concept or groups of concepts form a cluster of objects sharing similar properties. The diagrams obtained from this sort of clustering can then be used in class discovery and class prediction. Although a diagram of concepts can become large and complex, different approaches have worked toward reducing the complexity of concept lattices via conceptual scaling.

We refer the reader to Ganter & Wille (1998) for a general reference on formal concept analysis, and to Davey & Priestly (2002) for the basic concepts on order theory. For more thorough descriptions of different applications of formal concept analysis in the computer science field, see Carpineto & Romano (2004).

Cross References

►Clustering
►Constraint-Based Mining
►Frequent Itemset Mining

Recommended Reading

Carpineto, C., & Romano, G. (2004). *Concept data analysis. Theory and applications*. New York: Wiley.
Davey, B. A., & Priestly, H. A. (2002). *Introduction to lattices and order*. Cambridge: Cambridge University Press.
Ganter, B. & Wille, R. (1998). *Formal concept analysis. Mathematical foundations*. Heidelberg: Springer.

Frequent Itemset

Hannu Toivonen
University of Helsinki
Helsinki, Finland

Synonyms

Frequent set

Definition

Frequent itemsets (Agrawal et al., 1993, 1996) are a form of ►frequent pattern. Given examples that are sets of items and a minimum frequency, any set of items that occurs at least in the minimum number of examples is a frequent itemset.

For instance, customers of an on-line bookstore could be considered examples, each represented by the set of books he or she has purchased. A set of books, such as {"*Machine Learning*," "*The Elements of Statistical Learning*," "*Pattern Classification*,"} is a frequent itemset if it has been bought by sufficiently many customers. Given a frequency threshold, perhaps only 0.1 or 0.01% for an on-line store, *all* sets of books that have been bought by at least that many customers are called frequent. Discovery of all frequent itemsets is a typical data mining task. The original use has been as part of ►association rule discovery. ►Apriori is a classical algorithm for finding frequent itemsets.

The idea generalizes far beyond examples consisting of sets. The pattern class can be re-defined, e.g., to be (frequent) subsequences rather than itemsets; or original data can often be transformed to a suitable representation, e.g., by considering each discrete attribute-value pair or an interval of a continuous attribute as an individual item. In such more general settings, the term ►frequent pattern is often used. Another direction to generalize frequent itemsets is to consider other conditions than frequency on the patterns to be discovered; see ►constraint-based mining for more details.

Cross References

►Apriori Algorithm
►Association Rule
►Constraint-Based Mining
►Frequent Pattern

Recommended Reading

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC* (pp. 207–216). New York: ACM.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307–328). Menlo Park: AAAI Press.

Frequent Pattern

Hannu Toivonen
University of Helsinki
Finland

Definition

Given a set $\mathcal{D}$ of examples, a language $\mathcal{L}$ of possible patterns, and a minimum frequency *min_fr*, every pattern $\theta \in \mathcal{L}$ that occurs at least in the minimum number of examples, i.e., $|\{e \in \mathcal{D} \mid \theta \text{ occurs in } e\}| \geq min\_fr$, is a frequent pattern. Discovery of all frequent patterns is a common data mining task. In its most typical form, the patterns are ►frequent itemsets. A more general formulation of the problem is ►constraint-based mining.

Motivation and Background

Frequent patterns can be used to characterize a given set of examples: they are the most typical feature combinations in the data.

Frequent patterns are often used as components in larger data mining or machine learning tasks. In particular, discovery of ►frequent itemsets was actually first introduced as an intermediate step in ►association rule mining (Agrawal, Imieliński & Swami, 1993) ("frequent itemsets" were then called "large"). The frequency and confidence of every valid association rule $X \rightarrow Y$ are obtained simply as the frequency of $X \cup Y$ and the ratio of frequencies of $X \cup Y$ and X, respectively.

Frequent patterns can be useful as ►features for further learning tasks. They may capture shared properties of examples better than individual original features, while the frequency threshold gives some guarantee that the constructed features are not so likely just noise. However, other criteria besides frequency are often used to choose a good set of candidate patterns.

Structure of Problem

A frequent pattern often is essentially a set of binary ►features. Given a set $\mathcal{I}$ of all available features, the pattern language $\mathcal{L}$ then is the power set of $\mathcal{I}$. An example in data $\mathcal{D}$ covers a pattern $\theta \in \mathcal{L}$ if it has all the features of θ. In such cases, the frequent pattern discovery task reduces to the task of discovering ►frequent itemsets. Therefore, the structure of the frequent pattern discovery problem is best described using the elementary case of frequent itemsets.

Let $\mathcal{I}$ be the set of all items (or binary features); subsets of $\mathcal{I}$ are called itemsets (or examples or patterns, depending on the context). The input to the frequent itemset mining problem is a multiset $\mathcal{D}$ of itemsets (examples described by their features), and a frequency threshold. The task is to output *all* frequent itemsets (patterns) and their frequencies, i.e., all subsets of $\mathcal{I}$ that exceed the given frequency threshold in the given data $\mathcal{D}$.

Example 1 *Assume the following problem specification:*

- *Set of all items* $\mathcal{I} = \{A, B, C, D\}$.
- *Data* $\mathcal{D} = \{\{A, B, C\}, \{A, D\}, \{B, C, D\}, \{A, B, C\}, \{C, D\}, \{B, C\}\}$.
- *Frequency threshold is 2.*

All possible itemsets and their frequencies:

Itemset	Frequency
$\{A\}$	**3**
$\{B\}$	**4**
$\{C\}$	**5**
$\{D\}$	**3**
$\{A,B\}$	**2**
$\{A,C\}$	**2**
$\{A,D\}$	1
$\{B,C\}$	**4**

Itemset	Frequency
$\{B,D\}$	1
$\{C,D\}$	**2**
$\{A,B,C\}$	**2**
$\{A,B,D\}$	0
$\{A,C,D\}$	0
$\{B,C,D\}$	1
$\{A,B,C,D\}$	0

The frequent itemsets are $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{A,B\}$, $\{A,C\}$, $\{B,C\}$, $\{C,D\}$, $\{A,B,C\}$.

The ▶hypothesis space for itemsets obviously is the power set of $\mathcal{I}$, and it has an exponential size ($2^{|\mathcal{I}|}$) in the number of items. Since all frequent itemsets are output, this is also the size of the output in the worst case (e.g., if the frequency threshold is zero, or if all examples in $\mathcal{D}$ equal $\mathcal{I}$), as well as the worst case time complexity.

In practical applications of frequent itemset mining, the size of the output as well as the running times are much smaller, but they strongly depend on the properties of the data and the frequency threshold. The useful range of thresholds varies enormously among different datasets. In many applications – such as ▶basket analysis – the number $|\mathcal{I}|$ of different items can be in thousands, even millions, while the typical sizes of examples are at most in dozens. In such sparse datasets a relatively small number of frequent itemsets can reveal the most outstanding co-occurrences; e.g., there are not likely to be very large sets of books typically bought by the same customers. In dense datasets, in turn, the number of frequent patterns can be overwhelming and also relatively uninformative. E.g., consider the dense dataset of books that have *not* been purchased by a customer: there are a huge number of sets of books that have not been bought by the same customers.

Theory/solutions

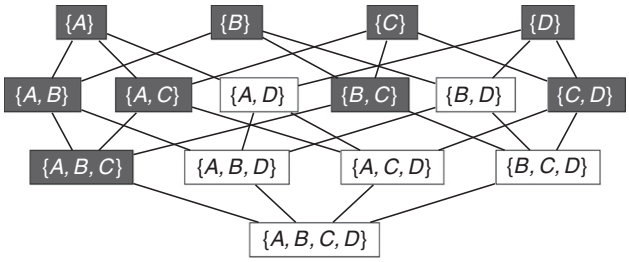
The most widely known solution for finding all frequent itemsets is the ▶Apriori algorithm (Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996). It is based on the monotonicity of itemset frequencies (a ▶generalization relation): the frequency of a set is at most as high as the frequency of any of its subsets. Conversely, if a set is known to be infrequent, then none of its supersets can be frequent.

Apriori views the ▶hypothesis space of itemsets as a (refinement) lattice defined by set containment, and performs a ▶general-to-specific search using ▶breadth-first search. In other words, it starts with singleton itemsets, the most general and frequent sets, and proceeds to larger and less frequent sets. The search is pruned whenever a set does not reach the frequency threshold: all supersets of such sets are excluded from further search. Apriori deviates from standard breadth-first search by evaluating all sets of equal size in a single batch, i.e., it proceeds in a levelwise manner. This has no effect on the search structure or results, but can reduce disk access considerably for large databases. See the entry ▶Apriori Algorithm for an outline of the method.

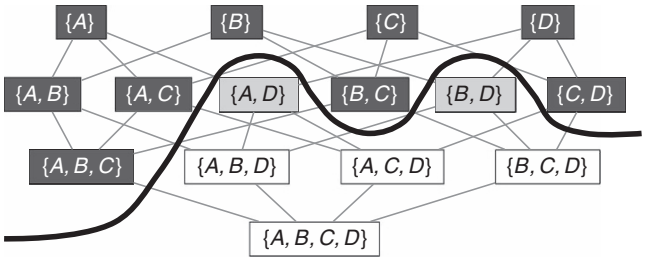
Example 2 *Figure 1 illustrates the search space for the data* $\mathcal{D}$ *of Example 1. Dark nodes represent frequent itemsets, i.e., the answer to the frequent itemset mining problem. Apriori traverses the space a level at a time. For instance, on the second level, it finds out that* $\{A,D\}$ *and* $\{B,D\}$ *are not frequent. It therefore prunes all their supersets, i.e., does not evaluate sets* $\{A,B,D\}$, $\{A,C,D\}$, *and* $\{B,C,D\}$ *on the third level.*

Other search strategies have also been applied. A ▶depth-first search without the subset check allows faster identification of candidates, at the expense of having more candidates to evaluate and doing that without natural batches (e.g., Zaki, 2000). FP-growth (Han, Pei, Yin, & Mao, 2004) uses a tree structure to store the information in the dataset, and uses it to recursively search for frequent itemsets.

The search strategy of Apriori is optimal in a certain sense. Consider the number of sets evaluated, and assume that for any already evaluated set we know whether it was frequent or not but do not consider its frequency. Apriori evaluates the frequencies of all frequent itemsets plus a number of candidates that turn out to be infrequent. It turns out that every infrequent candidate must actually be evaluated under the given assumptions: knowing which other sets are frequent and which are not does not help, regardless of the search



Frequent Pattern. Figure 1. The search space of frequent itemsets for data *D* of the running example. Dark nodes: frequent itemsets; white nodes: infrequent itemsets



Frequent Pattern. Figure 2. The positive border ($\{A,B,C\}$, $\{C,D\}$) and negative border ($\{A,D\}$, $\{B,D\}$) of frequent itemsets

order. This observation leads to the concept of *border*: the border consists of all those itemsets whose all proper subsets are frequent and whose all proper supersets are infrequent (Gunopulos et al., 2003; Mannila & Toivonen, 1997). The border can further be divided into two: the positive border contains those itemsets in the border that are frequent, the negative border contains those that are not. The positive border thus consists of the most specific patterns that are frequent, and corresponds to the "S" set of ▶version spaces.

Example 3 *Continuing our running example, Figure 2 illustrates the border between the frequent and infrequent sets. Either the positive or the negative border can alone be used to specify the collection of frequent itemsets: every frequent itemset is a subset of a set in the positive border* ($\{A,B,C\}$, $\{C,D\}$)*, while every infrequent itemset is a superset of a set in the negative border* ($\{A,D\}$, $\{B,D\}$)*.*

One variant of frequent itemset mining is to output the positive border only, i.e., to find the *maximal frequent itemsets* (Bayardo, 1998). This can be implemented with search strategies that do not need to evaluate the whole space of frequent patterns. This can be useful especially if the number of frequent itemsets is very large, or if the maximal frequent itemsets are large (in which case the number of frequent itemsets is large, too, since the number of subsets is exponential in the length of the maximal set). As a trade-off, the result does not directly indicate frequencies of itemsets.

Condensed Representations: Closed Sets and Nonderivable Sets Closed sets and nonderivable sets are a powerful concept for working with frequent itemsets, especially if the data is relatively dense or there are strong dependencies. Unlike the aforementioned simple model for borders, here also the known frequencies of sets are used to make inferences about frequencies of other sets.

As a motivation for closed sets (Pasquier, Bastide, Taouil, & Lakhal, 1999), consider a situation where the frequency of itemset $\{i,j\}$ equals the frequency of item j. This implies that whenever j occurs, so does i. Thus, any set $A \cup \{j\}$ that contains item j also contains item i, and the frequencies of sets $A \cup \{j\}$ and $A \cup \{i,j\}$ must be equal. As a result, it sufficies to evaluate sets $A \cup \{j\}$ to obtain the frequencies of sets $A \cup \{i,j\}$, too.

More formally, the *closure* of set A is its largest superset with identical frequency. A is *closed* iff it is its own closure, i.e., if every proper superset of A has a smaller

frequency than A. The utility of closed sets comes from the fact that frequent closed sets and their frequencies are a sufficient representation of all frequent sets. Namely, if B is a frequent set then its closure is a frequent closed set in $\mathcal{CL}$, where $\mathcal{CL}$ denotes the collection of all frequent closed itemsets. B's frequency is obtained as $\mathrm{fr}(B) = \max\{\mathrm{fr}(A) \mid A \in \mathcal{CL} \text{ and } B \subseteq A\}$. If B is not a frequent set, then it has no superset in $\mathcal{CL}$. ▶Formal concept analysis studies and uses closed sets and other related concepts.

Generators are a complementary concept, and also constitute a sufficient representation of frequent itemsets. (To be more exact, in addition to frequent generators, generators in the border are also needed). Set A is a *generator* (also known as a key pattern or a free set) if all its proper subsets have a larger frequency than A has. Thus, in an equivalence class of itemsets, defined by the set of examples in which they occur, the maximal element is unique and is the closed set, and the minimal elements are generators. The property of being a generator is monotone in the same way that being frequent is, and generators can be found with simple modifications to the Apriori algorithm.

Example 4 *Figure 3 illustrates the equivalence classes of itemsets by circles. For instance, the closure of itemset $\{A, B\}$ is $\{A, B, C\}$, i.e., whenever $\{A, B\}$ occurs in the data, C also occurs, but no other items. Given just the frequent closed sets and their frequencies, the frequency of, say, $\{B\}$ is obtained by finding its smallest frequent closed superset. It is $\{B, C\}$, with frequency 4, which is also B's frequency. Alternatively, using generators as the condensed representation, the frequency of itemset $\{B, C\}$ can be obtained by finding its maximal generator subset, i.e., $\{B\}$, with which it shares the same frequency.*

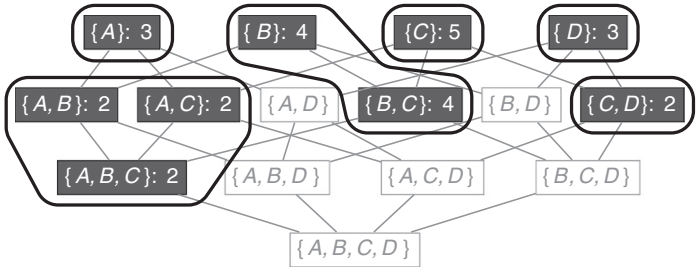
Nonderivability of an itemset (Calders & Goethals, 2002) is a more complex but often also a more powerful concept than closed sets. Given the frequencies of (some) subsets of itemset A, the frequency of A may actually be uniquely determined, i.e., there is only one possible consistent value. A practical method of trying to determine the frequency is based on deriving upper and lower bounds with inclusion–exclusion formula from the known frequencies of some subsets, and checking if these coincide. An itemset is derivable if this is indeed the case, otherwise it is *nonderivable*. Obviously, the collection of nonderivable frequent sets is a sufficient representation for all frequent sets.

Bounds for the absolute frequency of set I are obtained from its subsets as follows, for any $X \subseteq I$:

$$\mathrm{fr}(I) \leq \sum_{J: X \subseteq J \subset I} (-1)^{|I \setminus J|+1} \mathrm{fr}(J) \text{ if } |I \setminus X| \text{ is odd}, \quad (1)$$

$$\mathrm{fr}(I) \geq \sum_{J: X \subseteq J \subset I} (-1)^{|I \setminus J|+1} \mathrm{fr}(J) \text{ if } |I \setminus X| \text{ is even}. \quad (2)$$

Using all subsets X of I, one can obtain a number of upper and lower bounds. If the least upper bound equals the greatest lower bound, then set I is derivable. The conceptual elegance of this solution lies in the fact that derivable sets follow logically from the nonderivable ones – the aforementioned formula is one way of finding (some) such situations – whereas with closed sets the user must know the closure properties.



Frequent closed set $\mathcal{CL}$ = {{A},{C},{D},{B, C},{C, D},{A, B, C}}.
Frequent generators: {{A},{B},{C},{D},{A,B},{A,C},{C,D}}.

Frequent Pattern. Figure 3. Frequencies and equivalence classes of frequent itemsets in data $\mathcal{D}$ of the running example, and the corresponding closed sets and generators

Generalizations of Frequent Patterns The concept of frequent patterns has been extended in two largely orthogonal directions. One is to more complex patterns and data, such as frequent sequences, trees (see ►tree mining), graphs (see ►graph mining), and first-order logic (Dehaspe & Toivonen, 1999). The other direction to generalize the concept is to ►constraint-based mining, where other and more complex conditions are considered beyond frequency. We encourage the interested reader to continue at the entry for ►constraint-based mining, which also gives further insight into many of the more theoretical aspects of frequent pattern mining.

Programs and Data

Frequent itemset mining implementations repository: http://fimi.cs.helsinki.fi/
Weka: http://www.cs.waikato.ac.nz/ml/weka/
Christian Borgelt's implementations: http://www.borgelt.net/software.html
Data mining template library: http://dmtl.sourceforge.net/

Applications

Frequent patterns are a general purpose tool for data exploration, with applications virtually everywhere. Market ►basket analysis was the first application, telecom alarm correlation and gene mapping are examples of quite different application fields.

Future Directions

Work on frequent pattern mining is being expanded in several directions. New types of pattern languages are being developed, either to meet some specific needs or to increase the expressive power. Many of these developments are motivated by different types of data and applications. Within machine learning, frequent patterns are increasingly being used as a tool for feature construction in complex domains. For an end-user application, methods for choosing and ranking the most interesting patterns among thousands or millions of them is a crucial problem, for which there are no perfect solutions (cf. Geng & Hamilton, 2006). At the same time, theoretical understanding of the problem and solutions of frequent pattern discovery still has room for improvement.

Cross References

►Apriori Algorithm
►Association Rule
►Basket Analysis
►Constraint-Based Mining
►Data Mining
►Frequent Itemset
►Graph Mining
►Knowledge Discovery in Databases
►Tree Mining

Recommended Reading

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC* (pp. 207–216). New York: ACM.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307–328). Menlo Park, CA: AVAAI Press.

Bayardo, R. J. Jr. (1998). Efficiently mining long patterns from databases. In *Proceedings of the 1998 ACM SIGMOD international conference on management of data, Seatle, Washington, DC* (pp. 85–93). New York: ACM.

Calders, T., & Goethals, B. (2002). Mining all non-derivable frequent itemsets. In *Proceedings of the 6th European Conference on principles of data mining and knowledge discovery, Helsinki, Finland*. Lecture Notes in Computer Science (vol. 2431, pp. 74–85). London: Springer.

Ceglar, A., & Roddick, J. F. (2006). Association mining. *ACM Computing Surveys 38*(2): Article No. 5.

Dehaspe, L., & Toivonen, H. (1999). Discovery of frequent datalog patterns. *Data mining and knowledge discovery 3*(1): 7–36.

Geng, L., & Hamilton, H. J. (2006). Interestingness measures for data mining: A survey. *ACM Computing Surveys 38*(3): Article No. 9.

Gunopulos, D., Khardon, R., Mannila, H., Saluja, S., Toivonen, H., & Sharma, R. S. (2003). Discovering all most specific sentences. *ACM transactions on database systems 28*(2): 140–174.

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery 8*(1): 53–87.

Mannila, H., & Toivonen, H. (1997). Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery 1*(3): 241–258.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *Proceedings of 7th international conference on database theory, Jerusalem, Israel*. Lecture Notes in Computer Science (vol. 1540, pp. 398–416). London: Springer.

Zaki, M. J. (2000). Scalable algorithms for association mining. In *IEEE transactions on knowledge and data engineering* 12(3): 372–390.

Frequent Set

►Frequent Itemset

Functional Trees

►Model Trees

Fuzzy Sets

Fuzzy sets were introduced by Lofti Zadeh as a generalization of the concept of a regular set. A fuzzy set is characterized by a membership function that assigns a degree (or grade) of membership to all the elements in the universe of discourse. The membership value is a real number in the range [0, 1], where 0 denotes no definite membership, 1 denotes definite membership, and intermediate values denote partial membership to the set. In this way, the transition from nonmembership to membership in a fuzzy set is gradual and not abrupt like in a regular set, allowing the representation of imprecise concepts like "small," "cold," "large," or "very" for example.

A variable with its values defined by fuzzy sets is called a linguistic variable. For example, a linguistic variable used to represent a temperature can be defined as taking the values "cold," "comfortable," and "warm," each one of them defined as a fuzzy set. These linguistic labels, which are imprecise by their own nature, are, however, defined very precisely by using fuzzy set concepts.

Based on the concepts of fuzzy sets and linguistic variables, it is possible to define a complete fuzzy logic, which is an extension of the classical logic but appropriate to deal with approximate knowledge, uncertainty, and imprecision.

Recommended Reading

Zadeh, L. A. (1965). Fuzzy sets. *Information and control*. 8(3): 338–353.

Fuzzy Systems

A fuzzy system is a computing framework based on the concepts of the theory of ►fuzzy sets, fuzzy rules, and fuzzy inference. It is structured in four main components: a knowledge base, a fuzzification interface, an inference engine, and a defuzzification interface. The knowledge base consists of a rule base defined in terms of fuzzy rules, and a database that contains the definitions of the linguistic terms for each input and output linguistic variable. The fuzzification interface transforms the (crisp) input values into fuzzy values, by computing their membership to all linguistic terms defined in the corresponding input domain. The inference engine performs the fuzzy inference process, by computing the activation degree and the output of each rule. The defuzzification interface computes the (crisp) output values by combining the output of the rules and performing a specific transformation.

Fuzzy systems can be classified in different categories. The most widely used are the Mamdani and the Takagi-Sugeno models. In a Mamdani fuzzy system the output variables are defined as linguistic variables while in a Takagi-Sugeno fuzzy system they are defined as a linear combination of the input variables.

Fuzzy systems can model nonlinear functions of arbitrary complexity, however, their main strength comes from their ability to represent imprecise concepts and to establish relations between them.

Recommended Reading

Mamdani, E. H., & Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of man-machine studies*. 7(1): 1–13.

Sugeno, M. (1985) *Industrial applications of fuzzy control*. Elsevier Science Publishers, New York.

G

Gaussian Distribution

Xinhua Zhang
Australian National University, Canberra, Australia
NICTA London Circuit, Canberra, Australia

Synonyms

Normal distribution

Definition

The simplest form of Gaussian distribution is the one-dimensional standard Gaussian distribution, which can be described by the probability density function (pdf):

$$p(x) = \phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2},$$

where $\frac{1}{\sqrt{2\pi}}$ ensures the normalization, i.e., $\int_{\mathbb{R}} p(x)\mathrm{d}x = 1$. This distribution centers around $x = 0$ and the rate of decay or "width" of the curve is 1.

More generally, we can apply translation and scaling to obtain a Gaussian distribution that centers on arbitrary $\mu \in \mathbb{R}$ and with arbitrary width $\sigma > 0$. The pdf is:

$$p(x) = \frac{1}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

Technically, μ is called the mean and σ^2 is called the variance. Obviously, μ is the peak/mode of the density, and is also the mean and median of the distribution due to the symmetry of the density around μ. If a random variable X has this density, then we write

$$X \sim \mathcal{N}(\mu, \sigma^2).$$

Example density functions are plotted in Fig. 1a.

As an extension to multivariate random variables, the multivariate Gaussian distribution is a distribution on d-dimensional column vector $\boldsymbol{x}$ with mean column vector $\boldsymbol{\mu}$ and positive definite variance matrix Σ. This gives

$$p(\boldsymbol{x}|\mathbf{b}\boldsymbol{\mu}, \mathbf{b}\Sigma) = \frac{1}{(2\pi)^{d/2} \det^{1/2} \Sigma} \times \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right),$$

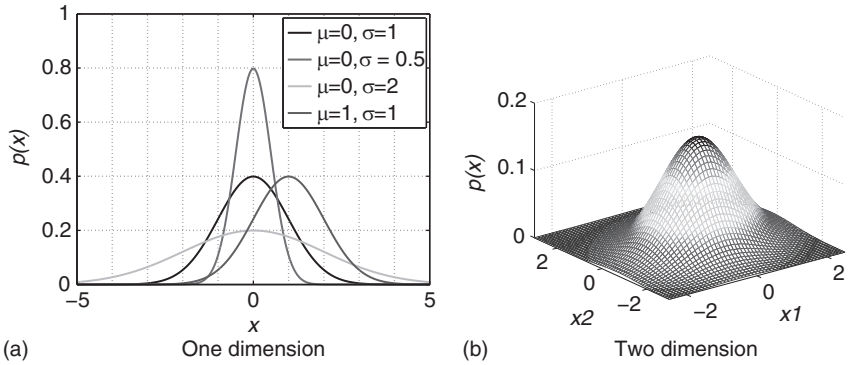
and is denoted by $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. An example *pdf* for the two dimensional case is plotted in Fig. 1b.

Motivation and Background

Gaussian distributions are one of the most important distributions in statistics. It is a continuous probability distribution that approximately describes some mass of objects that concentrate about their mean. The probability density function is bell-shaped, peaking at the mean. Its popularity also arises partly from the central limit theorem, which says the average of a large number of independent and identically-distributed random variables are approximately Gaussian distributed. Moreover, under some reasonable conditions, posterior distributions become approximately Gaussian in the large data limit. Therefore, the Gaussian distribution has been used as a simple model for many theoretical and practical problems in statistics, natural science, and social science.

In history, Abraham de Moivre first introduced this distribution in 1733 under the name "normal distribution" (of course, he did not call it Gaussian distribution since Gauss had not yet been born). Then Laplace used it to analyze experiment errors, based on which Legendre invented the least squares in 1805. Carl Friedrich Gauss rigorously justified it in 1809, and determined the formula of its probability density function. Finally this distribution is named the Gaussian distribution after Gauss. The name "normal distribution" is also widely

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,



Gaussian Distribution. Figure 1. Gaussian probability density functions

used, meaning it is a typical, common, or usual distribution. It was coined by Peirce, Galton, and Lexis around 1875, and made popular by Karl Pearson near the inception of the twentieth century.

Theory/Solution

Canonical Form

The standard definition allows one to easily read off the moments from the *pdf*. Another useful parameterization is called canonical parameterization:

$$p(\boldsymbol{x}|\boldsymbol{\eta},\Lambda) = \exp\left(\eta^\top x - \frac{1}{2}x^\top \Lambda x - \frac{1}{2}\left(d\log(2\pi) - \log\det\Lambda + \boldsymbol{\eta}^\top \Lambda \eta\right)\right),$$

where $\eta = \Sigma^{-1}\boldsymbol{\mu}$ and $\Lambda = \Sigma^{-1}$. Λ is often called precision. This parameterization is useful when posing the distribution as a member of the exponential family.

Cumulative Distribution Function

For a one-dimensional Gaussian distribution, the cumulative distribution function (cdf) is defined by

$$\Phi(x) = \int_{-\infty}^{x} \phi(t)\mathrm{d}t.$$

Formally, it can be conveniently represented by the error function and its complement:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\mathrm{d}t,$$

$$\mathrm{erfc}(x) = 1 - \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}\mathrm{d}t.$$

So

$$\Phi(x) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = \frac{1}{2}\mathrm{erfc}\left(-\frac{x}{\sqrt{2}}\right).$$

The inverse of the *cdf*, called quantile function, can be written as

$$\Phi^{-1}(s) = \sqrt{2}\,\mathrm{erf}^{-1}(2s-1), \quad \text{for } s \in (0,1).$$

The error function erf() and its inverse $\mathrm{erf}^{-1}()$ do not usually have a closed form, and can be computed numerically by functions like *ERF* in Fortran, and *double erf(double x)* in C/C++. For the multi-variate case, the corresponding *cdf* is highly challenging to compute numerically.

Moments

The first order moment is $\mathbb{E}[X] = \boldsymbol{\mu}$, the variance is $\mathrm{Var}[X] = \Sigma$, and all higher order cumulants are 0. Any central moments with odd terms are 0, i.e., $\mathbb{E}[\Pi_{i=1}^{d}(x_i - \mu_i)^{p_i}] = 0$ when $\sum_i p_i$ is odd.

Entropy and Kullback–Leibler Divergence

The differential entropy of a multi-variate Gaussian is

$$h(p) = -\int_{\mathbb{R}^d} p(x)\ln p(x)\mathrm{d}x = \frac{1}{2}\ln\left((2\pi e)^d \det\Sigma\right).$$

The Kullback–Leibler divergence from $\mathcal{N}(\boldsymbol{\mu}_1,\Sigma_1)$ to $\mathcal{N}(\boldsymbol{\mu}_2,\Sigma_2)$ is

$$\mathrm{KL}(\mathcal{N}(\boldsymbol{\mu}_1,\Sigma_1)||\mathcal{N}(\boldsymbol{\mu}_2,\Sigma_2)) = \frac{1}{2}\left(\ln\frac{\det\Sigma_2}{\det\Sigma_1} + \mathrm{tr}\,\Sigma_2^{-1}\Sigma_1 + (\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)^\top\Sigma_2^{-1}(\boldsymbol{\mu}_2-\boldsymbol{\mu}_1) - d\right).$$

Properties Under Affine Transform

Let $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. Suppose A is a linear transform from $\mathbb{R}^d$ to $\mathbb{R}^s$ and $c \in \mathbb{R}^s$, then

$$\begin{aligned} Ax + c &\sim \mathcal{N}(A\boldsymbol{\mu} + c, A\Sigma A^\top) \\ \mathbb{E}[(x-\boldsymbol{\mu})^\top A(x-\boldsymbol{\mu})] &= \text{tr } A\Sigma \\ \text{Var}[(x-\boldsymbol{\mu})^\top A(x-\boldsymbol{\mu})] &= 2 \text{ tr } A\Sigma A\Sigma. \end{aligned}$$

where the last two relations require $s = d$.

Conjugate Priors

Conjugate priors where discussed in ▶prior probabilities. With known variance, the conjugate prior for the mean is again a multi-variate Gaussian. With known mean, the conjugate prior for the variance matrix is the Wishart distribution, while the conjugate prior for the precision matrix is the Gamma distribution.

Parameter Estimation

Given n *iid* observations $X_1, \ldots, X_n$, the maximum likelihood estimator of the mean is simply the sample mean

$$\boldsymbol{\mu} = \bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

The maximum likelihood estimator of the covariance matrix is:

$$\tilde{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})^\top.$$

This estimator is biased, and its expectation is $\mathbb{E}[\tilde{\Sigma}] = \frac{n-1}{n}\Sigma$. An unbiased estimator is

$$\tilde{\Sigma} = S = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})^\top.$$

Distributions Induced by the Gaussian

If $X \sim \mathcal{N}(0, \Sigma)$, then $X^\top \Sigma^{-1} X$ has a Gamma distribution $\text{Gamma}(d/2, 2)$.

Let $X_1, X_2 \sim \mathcal{N}(0,1)$ and they are independent. Their ratio is the standard Cauchy distribution, $X_1/X_2 \sim \text{Cauchy}(0,1)$.

Given n independent univariate random variables $X_i \sim \mathcal{N}(0,1)$, the random variable $Z := \sqrt{\sum_i X_i^2}$ has a χ distribution with degree of freedom n. And Z^2 has a χ^2 distribution with degree of freedom n.

Using Basu's theorem or Cochran's theorem, one can show that the sample mean of $X_1, \ldots, X_n$ and the sample standard deviation are independent. Their ratio

$$t := \frac{\bar{X}}{S} = \frac{\frac{1}{n}(X_1 + \cdots + X_n)}{\sqrt{\frac{1}{n-1}\left[(X_1 - \bar{X})^2 + \cdots + (X_n - \bar{X})^2\right]}}$$

has the student's t-distribution with degree of freedom $n-1$.

Applications

This section discusses some applications and properties of the Gaussian.

Central Limit Theorem

Given n independent and identically distributed observations drawn from a distribution whose variance is finite, the average of the observations is asymptotically Gaussian distributed when n tends to infinity. Under certain conditions, the requirement for identical distribution can be relaxed and asymptotic normality still holds.

Approximate Gaussian Posterior

Consider n independent and identically distributed observations drawn from a distribution $p(X_i|\theta)$, so the data set is $\mathbf{X} = (X_1, \ldots, X_n)^\top$. Under certain conditions, saying roughly that the posterior on θ converges in probability to a single interior point in its domain as $n \to \infty$, the posterior for $\vec{\theta}$ is approximately Gaussian for large n, $\theta|\mathbf{X} \approx \mathcal{N}\left(\widehat{\theta}, I\left(\widehat{\theta}\right)\right)$, where $\widehat{\theta}$ is the maximum likelihood or aposterior value for θ and $I(\theta)$ is the *observed (Fisher) information*, the negative of the second derivative (Hessian) of the likelihood w.r.t. the parameters θ.

The Gaussian approximation to the posterior, while a poor approximation in many cases, serves as a useful insight into the nature of asymptotic reasoning. It is justified based on the multi-dimensional Taylor expansion of the log likelihood at the maximum likelihood or aposterior value, together with its asymptotic convergence property.

3-σ Rule

For standard Gaussian distribution, 99.7% of the probability mass lie within the three standard deviations $[-3\sigma, 3\sigma]$, i.e., $\int_{-3\sigma}^{3\sigma} \phi(x)\mathrm{d}x > 0.997$. About 95% mass

lies within two standard deviations, and about 68% within one standard deviation. This empirical rule is called 3-σ rule, and can be easily extended to general one dimensional Gaussian distributions.

Combination of Random Variables

Let d-dimensional random variables $X_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$. If they are independent, then for any set of linear transforms A_i from $\mathbb{R}^d$ to $\mathbb{R}^s$, we have $\sum_i A_i X_i \sim \mathcal{N}(\sum_i A_i \boldsymbol{\mu}_i, \sum_i A_i \Sigma_i A_i^\top)$. The converse is also true by the Cramer's theorem: if X_i are independent and their sum $\sum_i X_i$ is Gaussian distributed, then all X_i must be Gaussian.

Correlations and Independence

In general, independent random variables must be uncorrelated but not vice versa. However, if a multivariate random variable is jointly Gaussian, then any uncorrelated subset of the random variables *must be* independent. Notice the precondition of joint Gaussian. It is possible for two Gaussian random variables to be uncorrelated but not independent, for the reason that they are not jointly Gaussian. For example, let $X \sim \mathcal{N}(0,1)$ and $Y = -X$ if $|X| < c$, and $Y = X$ if $|X| > c$. By properly setting c, Y and X can be made uncorrelated but obviously not independent.

Marginalization, Conditioning, and Agglomeration

Suppose the vector x can be written as $(x_1^\top, x_2^\top)^\top$ and correspondingly the mean and covariance matrix can be written as

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

Then the marginal distribution of x_1 is Gaussian $\mathcal{N}(\mu_1, \Sigma_{11})$, and the conditional distribution of x_1 conditioned on x_2 is $\mathcal{N}(\boldsymbol{\mu}_{1|2}, \Sigma_{1|2})$, where

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \boldsymbol{\mu}_2), \quad \Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}.$$

Suppose the multi-variate Gaussian vector $x_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$, and a vector x_2 is a linear function of x_1 with Gaussian noise, i.e., $x_2|x_1 \sim \mathcal{N}(Ax_1 + \boldsymbol{\mu}_{12}, \Sigma_{12})$. Then the joint distribution of $(x_1^\top, x_2^\top)^\top$ is also Gaussian:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_1 \\ A\boldsymbol{\mu}_1 + \boldsymbol{\mu}_{12} \end{pmatrix}, \begin{pmatrix} \Sigma_{11} + A^\top \Sigma_{12} A & -A^\top \Sigma_{12} \\ -\Sigma_{12} A & \Sigma_{12} \end{pmatrix} \right).$$

For a complete treatment of Gaussian distributions from a statistical perspective, see Casella and Berger (2002), and Mardia, Kent, and Bibby (1979) provides details for the multi-variate case. Bernardo and Smith (2000) shows how Gaussian distributions can be used in the Bayesian theory. Bishop (2006) introduces Gaussian distributions in Chap. 2, and shows how it is extensively used in machine learning. Finally, some historical notes on Gaussian distributions can be found at http://jeff560.tripod.com/mathword.html, especially under the entries "NORMAL" and "GAUSS."

Cross References

►Gaussian Processes

Recommended Reading

Bernardo, J. M., & Smith, A. F. M. (2000). *Bayesian theory*. Chichester: Wiley.

Bishop, C. (2006). *Pattern recognition and machine learning*. New York: Springer.

Casella, G., & Berger, R. (2002). *Statistical inference* (2nd ed.). Pacific Grove, CA: Duxbury.

Jolliffe, I. T. (2002). *Principal component analysis* (2nd ed.). *Springer series in statistics*. New York: Springer.

Mardia, K. V., Kent, J. T., & Bibby, J. M. (1979). *Multivariate analysis*. London: Academic Press.

Miller, J., Aldrich, J., Cabillón, J. G., de Araújo, C. C., Landau, J. A. *Earliest known uses of some of the words of mathematics*. http://jeff560.tripod.com/mathword.html

Gaussian Process

Novi Quadrianto[1], Kristian Kersting[2], Zhao Xu[2]
[1]Department of Engineering and Computer Science, RSISE, ANU and SML, NICTA, Canberra, Australia
[2]Fraunhofer IAIS, Sankt Augustin, Germany

Synonyms

Expectation propagation; Kernels; Laplace estimate; Nonparametric Bayesian

Definition

Gaussian processes generalize multivariate Gaussian distributions over finite dimensional vectors to infinite dimensionality. Specifically, a Gaussian process is a

stochastic process that has Gaussian distributed finite dimensional marginal distributions, hence the name. In doing so, it defines a distribution over functions, i.e., each draw from a Gaussian process is a function. Gaussian processes provide a principled, practical, and probabilistic approach to inference and learning in kernel machines.

Motivation and Background

Bayesian probabilistic approaches have many virtues, including their ability to incorporate prior knowledge and their ability to link related sources of information. Typically, we are given a set of data points sampled from an underlying but unknown distribution, each of which includes input x and output y, such as the ones shown in Fig. 1a. The task is to learn a functional relationship between x and y. Traditionally, in a parametric approach, an assumption on the mathematical form of the relationship such as linear, polynomial, exponential, or combination of them needs to be chosen a priori. Subsequently, weights (or parameters) are placed on each of the chosen forms, and a prior distribution is then defined over parameters. Thus, the learning task is now reduced to the Bayesian estimation over the parameters, cf. Fig. 1a–c. This approach, however, may not always be practical, as illustrated in Fig. 1d. To discover the latent input–output relationship in Fig. 1d, we might need infinitely many functional forms, and this translates to infinite number of parameters. Instead of working over a parameter space, Gaussian processes place a prior directly on the space of functions without parameterizing the function, hence nonparametric. As will be shown, the computational complexity of inference now scales as the number of data points instead of the number of parameters.

Several nonparametric Bayesian models have been developed for different tasks such as density estimation, regression, classification, survival time analysis, topic modeling, etc. Among the most popular ones are ▶*Dirichlet processes* and *Gaussian processes*. Just as the Gaussian process, a Dirichlet process has Dirichlet distributed finite dimensional marginal distributions, hence the name.

Gaussian processes were first formalized for machine learning tasks by Williams and Rasmussen (1996) and Neal (1996).

Theory

Formally, a Gaussian process is a stochastic process (i.e., a collection of random variables) in which all the finite-dimensional distributions are multivariate Gaussian distributions for any finite choice of variables. In general, Gaussian processes are used to define a probability distribution over functions $f : \mathcal{X} \to \mathbb{R}$ such that the set of values of f evaluated at an arbitrary set of points $\{x_i\}_{i=1}^N \in \mathcal{X}$ will have an N-variate Gaussian distribution. Note that, for $x_i \in \mathbb{R}^2$, this may also be known as a Gaussian random field.

Gaussian Process

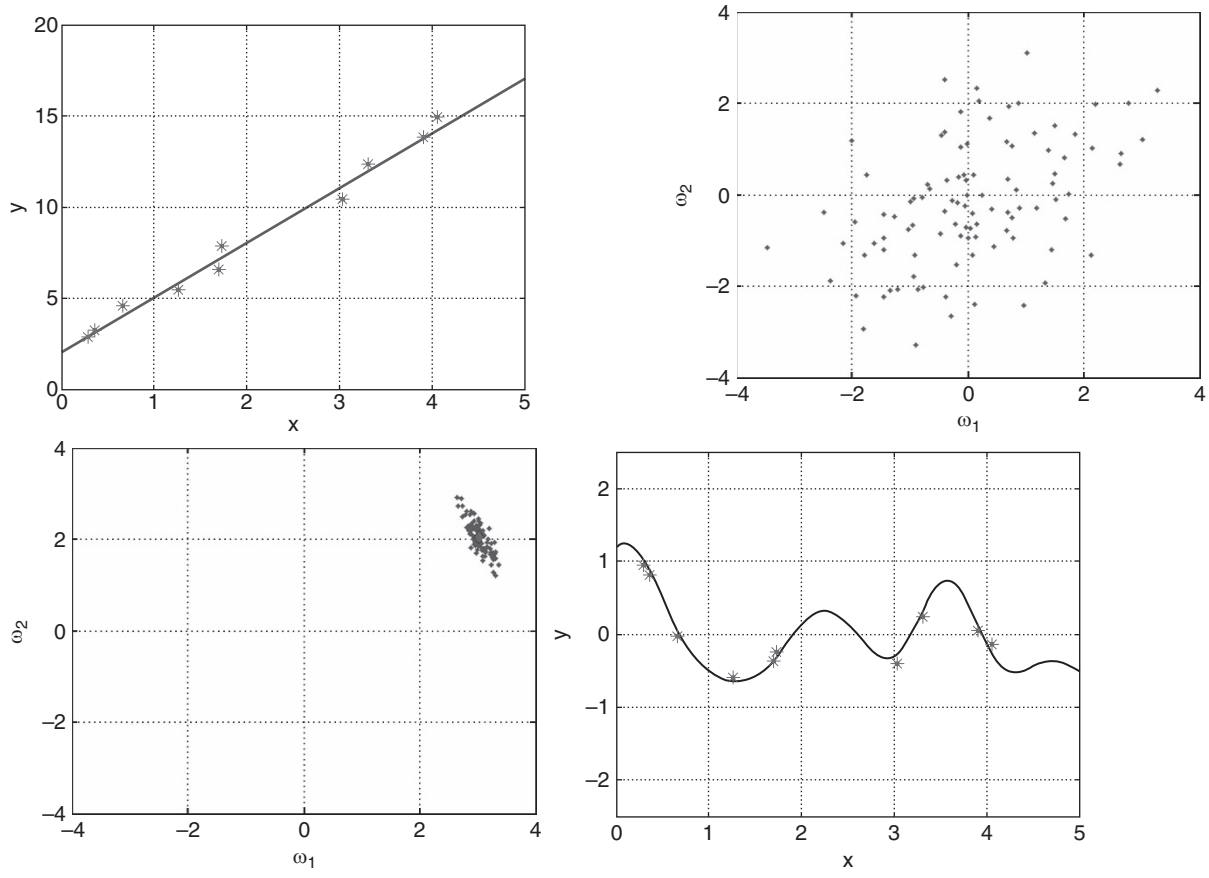
A Gaussian distribution is completely specified by its mean and covariance matrix. Similarly, a Gaussian process is characterized by its mean function $m(x) := \mathbf{E}[f(x)]$ and covariance function

$$C(x, x') := \mathbf{E}[(f(x) - m(x))(f(x') - m(x'))] .$$

We say a real process $f(x)$ is Gaussian process distributed with a mean function $m(x)$ and a covariance function $C(x, x')$, written as $f \sim \mathcal{GP}(m(x), C(x, x'))$.

The mean function can be arbitrarily chosen (for convenience, it is often taken to be a zero function since we can always center our observed outputs to have a zero mean), but the covariance function must be a positive definite function to ensure the existence of all finite-dimensional distributions. That is, the positive definiteness of $C(.,.)$ ensures the positive (semi-) definiteness of all covariance matrices, Σ, appearing in the exponent of the finite-dimensional multivariate Gaussian distribution.

The attractiveness of Gaussian processes is that they admit the marginalization property (▶Gaussian Distribution), i.e., if the Gaussian process specifies $(f(x_1), f(x_2)) \sim \mathcal{N}(\mu, \Sigma)$, then it must also specify $f(x_1) \sim \mathcal{N}(\mu_1, \Sigma_{11})$, where Σ_{11} is the relevant submatrix of Σ. This means, addition of novel points will not influence the distribution of existing points. The marginalization property allows us to concentrate on distribution of only observed data points with the rest of unobserved points considered to be marginalized out; thus a finite amount of computation for inference can be achieved.



Gaussian Process. Figure 1. (a) Ten observations (one-dimensional input *x* and output *y* variables) generated from a ▶linear regression model $y = 3x + 2 + \epsilon$ with Gaussian noise ϵ. The task is to learn the functional relationship between *x* and *y*. Assuming the parametric model $y = \omega_1 x + \omega_2 + \epsilon$, i.e., $\omega = (\omega_1, \omega_2)$ is the vector of parameters, and the prior distribution over ω be a 2-dimensional Gaussian as shown in (b), the posterior distribution over ω can be estimated as shown in (c). Its mean $(3.0287, 2.0364)$ is close to the true parameters $(3, 2)$. The inference, however, was performed in an ideal situation where in the relationship between *x* and *y* was indeed linear. If the true relationship is not known in advances and/or cannot easily be described using a finite set of parameters, this approach may fail. For example, in (d), infinite number of parameters might be required to recover the functional relationship

Covariance Functions

A covariance function bears an essential role in a Gaussian process model as its continuity properties determine the continuity properties of samples (functions) from the Gaussian process. In the literature, covariance functions are also known as positive (semi-)definite kernels or Mercel kernels.

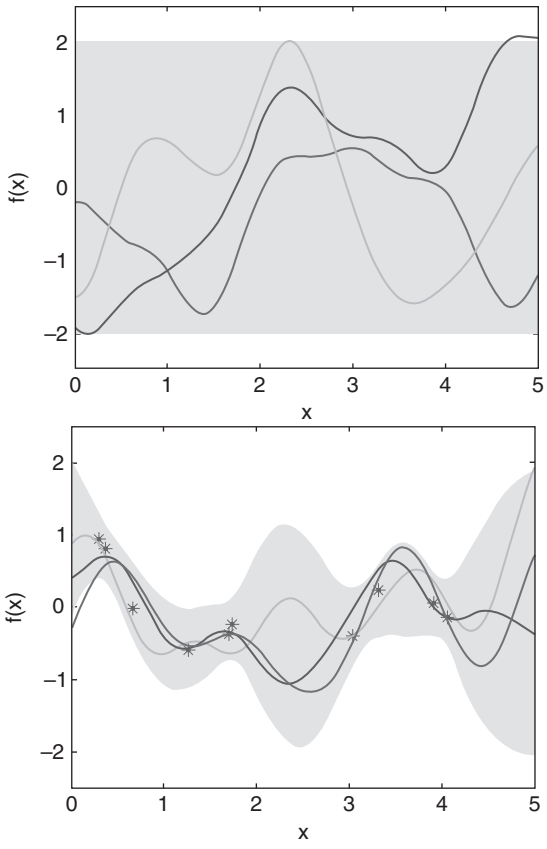
There are generally two types of covariance functions: *stationary* and *non-stationary*. A stationary covariance function is a function that is translation invariant, i.e., $C(x, x') = D(x - x')$ for some function D. The typical examples include squared exponential, Matérn class, γ-exponential, exponential, rational quadratic, while examples of non-stationary covariance functions are dot product and polynomial.

Squared exponential (SE) is a popular form of stationary covariance function, and it corresponds to the class of sums of infinitely many Gaussian shaped basis functions placed everywhere, $f(x) := \lim_{n\to\infty} \frac{s}{n} \sum_i \gamma_i \exp\left(-((x - x_i)/2\ell)^2\right)$ with $\gamma_i \sim \mathcal{N}(0,1)\ \forall i$. This covariance function is in the form of

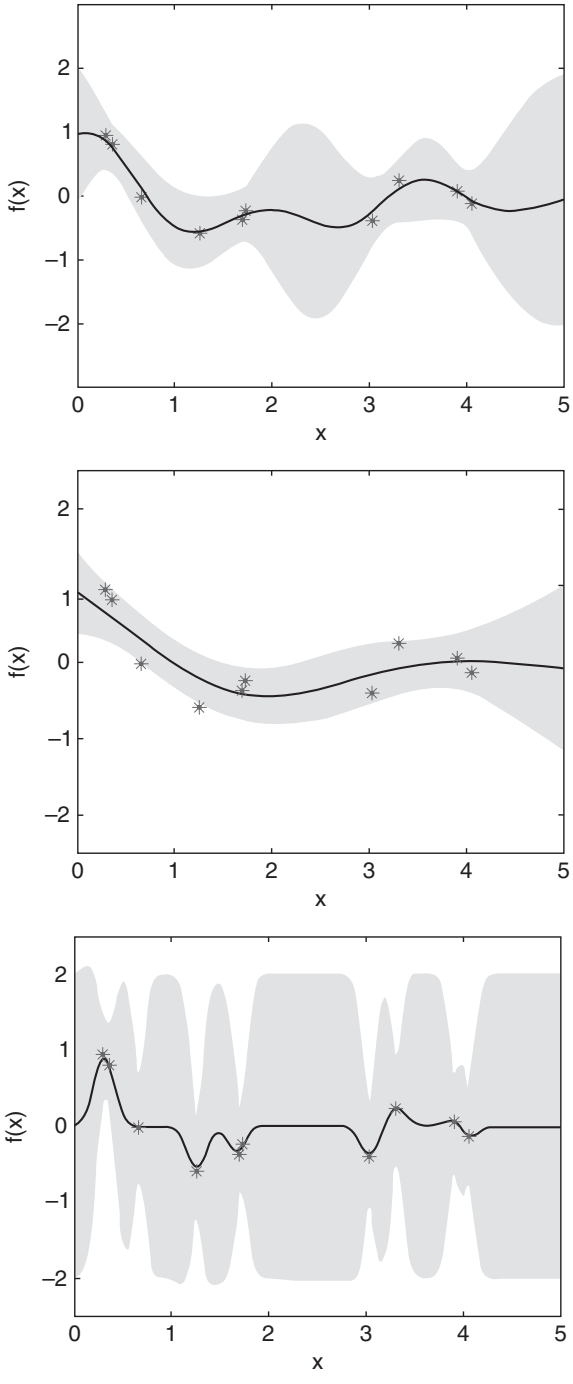
$$C(x, x') = \mathbf{E}[f(x)f(x')] = s^2 \exp\left(-\frac{1}{2\ell^2} \|x - x'\|_2^2\right).$$

Typical functions sampled from this covariance function can be seen in Fig. 2a. This covariance function has the characteristic length scale ℓ and the signal variance s^2 as free parameters (hyperparameters). The longer the characteristic length scale, the more slowly varying the typical sample function is. The signal variance defines the vertical scale of variations of a sample function. Figure 3 illustrates prediction with SE covariance function with varying characteristic length scale. Several other covariance functions are listed in Table 1.

For a comprehensive review on the field of covariance functions, we refer interested readers to (Abrahamsen, 1992).



Gaussian Process. Figure 2. (a) Three functions drawn at random from a Gaussian process prior. (b) Three random functions drawn from the posterior, i.e., the distribution learned with the prior from Fig. 2a and the ten observations from Fig. 1d. In both plots the shaded area shows the pointwise mean plus and minus two times the standard deviation for each input value, i.e., the 95% confidence region



Gaussian Process. Figure 3. Gaussian process prediction with the SE kernel. (a) mean of the prediction distribution with length-scale 1.0 and signal variance 1.0 (the hyperparameters of the original process used to generate the data in Fig. 1). The other two plots show the prediction setting the length-scale (b) longer (1.5) and (c) shorter (0.1). In all plots, the 95% confidence region is shown

Gaussian Process. Table 1 Examples of covariance functions. θ_{cov} denotes the set of hyperparameters

Name	$C(x,x')$	θ_{cov}	Remark
Squared exp. (SE)	$s^2 \exp\left(-\frac{1}{2\ell^2}\|x-x'\|_2^2\right)$	$\{s,\ell\}$	Strong smoothness assumption
Matérn class	$\frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}\|x-x'\|}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}r}{\ell}\right)$	$\{\nu,\ell\}$	Less smooth than SE
γ-exponential	$\exp(-(\|x-x'\|/\ell)^\gamma)$, with $0 < \gamma <= 2$	$\{\ell\}$	Includes both Exp. and SE
Exponential	$\exp\left(\frac{-\|x-x'\|}{\ell}\right)$	$\{\ell\}$	$\nu = 1/2$ in the Matérn class
Rational quadratic	$\left(1+\frac{\|x-x'\|_2^2}{2\alpha\ell^2}\right)^{-\alpha}$	$\{\alpha,\ell\}$	An infinite sum of SE
Dot product	$\sigma_w^2\langle x,x'\rangle + \sigma_c^2$	$\{\sigma_w,\sigma_c\}$	
Polynomial	$(\langle x,x'\rangle + \sigma_c^2)^p$	$\{\sigma_c\}$	Effective for high-dimensional classification with binary or grayscale input

Applications

For Gaussian processes, there are two main classes of applications: regression and classification. We will discuss each of them in turn.

Regression

In a ▶regression problem, we are interested to recover a functional dependency $y_i = f(x_i) + \epsilon_i$ from N observed training data points $\{(x_i,y_i)\}_{i=1}^N$, where $y_i \in \mathbb{R}$ is the noisy observed output at input location $x_i \in \mathbb{R}^d$. Traditionally, in the Bayesian ▶linear regression model, this regression problem is tackled by requiring us to parameterize the latent function f by a parameter $w \in \mathbb{R}^H$, $f(x) := \langle \phi(x), w\rangle$ for H fixed basis functions $\{\phi_h(x)\}_{h=1}^H$. A prior distribution is then defined over parameter w. The idea of Gaussian process regression (in the geostatistical literature, this is also called *kriging*, see e.g., (Krige, 1951; Matheron, 1963)) is to place a prior directly on the space of functions without parameterizing the function (vide Motivation and Background).

Likelihood Function and Posterior Distribution: Assuming independent and normally distributed noise terms, $\epsilon_i \sim \mathcal{N}(0,\sigma_{\text{noise}}^2)$, the likelihood model on an output vector $Y \in \mathbb{R}^N$ and an input matrix $X \in \mathbb{R}^{N\times d}$ will be

$$Y|f,X \sim \mathcal{N}(f_X, \sigma_{\text{noise}}^2 I).$$

That is, the data likelihood is distributed according to a Gaussian distribution with the function values evaluated at training input locations as its mean and the variance of the noise terms as its variance.

Placing a (zero mean) Gaussian process prior over functions

$$f \sim \mathcal{GP}(m(x) \equiv 0, k(x,x')), \tag{1}$$

will lead us to a Gaussian process posterior (this form of posterior process is described in the next section),

$$f|X,Y \sim \mathcal{GP}(m_{\text{post}}(x) = k(x,X)[K+\sigma_{\text{noise}}^2 I]^{-1}Y,$$
$$k_{\text{post}}(x,x') = k(x,x') - k(x,X)[K+\sigma_{\text{noise}}^2 I]^{-1}k(x',X)). \tag{2}$$

In the above equations, $K \in \mathbb{R}^{N\times N}$ denotes the Gram matrix with elements $K_{ij} = k(x_i,x_j)$ and $k(x,x')$ is the kernel function. The term $k(x,X)$ denotes a kernel function with one of the inputs fixed at training points.

Predictive Distribution: The final goal in regression is to make an output prediction for a novel input x_*, given a set of input-output training points. By the marginalization property, instead of working with a prior over infinite dimensional function spaces as in (1), we can concentrate on the marginal distribution over the training inputs,

$$f_X \sim \mathcal{N}(0,K). \tag{3}$$

Subsequently, the marginal distribution over training outputs (conditioned on inputs) can be computed via

$$p(Y|X) = \int p(Y|f_X)p(f_X)df_X = \mathcal{N}(0, K + \sigma^2_{\text{noise}}I). \tag{4}$$

The above integration is computed by using the standard result for the convolution of two Gaussian distributions (►Gaussian Distribution). The joint distribution over sets of training points Y and the quantity we wish to predict y_* is given by

$$p(Y, y_*|X, x_*) = \mathcal{N}(0, C), \tag{5}$$

where $C \in \mathbb{R}^{(N+1)\times(N+1)}$ is the joint covariance matrix. We can partition this joint covariance matrix as follows:

$$C = \begin{bmatrix} K + \sigma^2_{noise}I & k_{X,x_*} \\ k^\top_{X,x_*} & k(x_*, x_*) + \sigma^2_{noise} \end{bmatrix},$$

where the vector $k_{X,x_*} \in \mathbb{R}^N$ has elements $k(x_i, x_*)$ for $i = 1, \ldots, N$ and $\top$ denotes a transpose operation. The noise variance appears only at the diagonal elements of the covariance matrix C, this is due to the independence assumption about the noise. Using a standard Gaussian property on computing conditional distribution from a joint Gaussian distribution (►Gaussian Distribution), the predictive distribution is given by

$$p(y_*|x_*, X, Y) = \mathcal{N}(\mu_*, \sigma^2_*), \tag{6}$$

with

$$\mu_* = k^\top_{X,x_*}(K + \sigma^2_{\text{noise}}I)^{-1}Y, \tag{7}$$

$$\sigma^2_* = k(x_*, x_*) - k^\top_{X,x_*}(K + \sigma^2_{\text{noise}}I)^{-1}k_{X,x_*} + \sigma^2_{\text{noise}}. \tag{8}$$

Note that, (7) and (8) are the mean function and the covariance function of the posterior process in (2) for any novel inputs. The only difference is the additional term σ^2_{noise}, since there exists observation noise ϵ_* such that $y_* = f_* + \epsilon_*$.

Point Prediction: The previous section has shown how to compute a predictive distribution for outputs y_* associated with the novel test inputs x_*. To convert this predictive distribution into a point prediction, we need the notion of a loss function, $\mathcal{L}(y_{\text{true}}, y_{\text{prediction}})$. This function specifies the loss incurred for predicting the value $y_{\text{prediction}}$ while the true value is y_{true}. Thus, the optimal point prediction can be computed by minimizing the expected loss as follows

$$y_{\text{optimal}}|x_* = \text{argmin}_{y_{\text{prediction}}} \int \mathcal{L}(y_*, y_{\text{prediction}}) \times p(y_*|x_*, X, Y)dy_*. \tag{9}$$

For a squared loss function (or any other symmetric loss functions) and predictive distribution (6), the solution to the above equation is the mean of the predictive distribution, i.e.,

$$y_{\text{optimal}}|x_* = \mathbf{E}_{y_*\sim p(y_*|x_*,X,Y)}[y_*] = \mu_*.$$

The above Gaussian process regression description is known as a function space view in the literature (Rasmussen & Williams, 2006). Equivalently, a Gaussian process regression can also be viewed from the traditional Bayesian linear regression with a possibly infinite number of basis functions $\phi(x)$ and with a zero mean and arbitrary positive definite covariance matrix Gaussian prior on the parameter w, see e.g., Rasmussen & Williams (2006).

Classification

Gaussian process models can also be applied to classification problems. In a probabilistic approach to classification, the goal is to model posterior probabilities of an input point x_i belonging to one of Ω classes, $y_i \in \{1, \ldots, \Omega\}$. These probabilities must lie in the interval $[0, 1]$, however, a Gaussian process model draws functions that lie on $(-\infty, \infty)$. For the binary classification ($\Omega = 2$), we can turn the output of a Gaussian process into a class probability using an appropriate nonlinear activation function. In the following, we will show this for the case of binary classification. For the more general cases, see e.g., Rasmussen & Williams (2006).

Likelihood Function and Posterior Distribution: In a regression problem, a Gaussian likelihood is chosen and combined with a Gaussian process prior, which leads to a Gaussian posterior process over functions where in all required integrations remain analytically tractable. For classification, however, Gaussian likelihood is not

the most suitable owing to the discreteness nature of the class labels. The most commonly used likelihood functions are

$$p(y_i|f,x_i) = \frac{1}{1+\exp(-y_i f_{x_i})} \quad \text{or}$$

$$p(y_i|f,x_i) = \int_{-\infty}^{y_i f_{x_i}} \mathcal{N}(0,1)dt = \Phi_{0,1}(y_i f_{x_i}), \quad (10)$$

known as logistic and cumulative Gaussian likelihood functions, respectively. Assuming that the class labels (conditioned on f) are generated independent and identically distributed, the joint likelihood over N data points can be expressed as $p(Y|f,X) = \prod_{i=1}^{N} p(y_i|f,x_i)$. By Bayes' rule, the posterior distribution over latent functions is given by $p(f_X|X,Y) = \frac{p(Y|f)p(f_X)}{\int p(Y|f)p(f_X)df_X}$. This posterior is no longer analytically tractable (due to intractable integration in the denominator) and an approximation is needed.

There are several approximation methods to handle intractability of the inference stage in Gaussian process classification such as Laplace approximation, expectation propagation, variational bounding, and MCMC, among others (see (Nickisch & Rasmussen, 2008) for a comprehensive overview of approximate inference in binary Gaussian process classification). Most of the methods (if not all) approximate the non-Gaussian posterior with a tractable Gaussian distribution. We describe in detail the straightforward Laplace approximation method, but note that the more complicated expectation propagation (▶Expectation Propagation) is almost always the method of choice unless the computational budget is very tight (Nickisch & Rasmussen, 2008).

Laplace's method approximates the non-Gaussian posterior with a Gaussian one by performing a second order Taylor expansion of the log posterior, $\log p(f_X|X,Y)$ at the maximum point of the posterior

$$p(f_X|X,Y) \approx \hat{p}(f_X|X,Y) = \mathcal{N}(\hat{f}_X, H^{-1}), \quad (11)$$

where $\hat{f}_X = \text{argmax}_{f_X} \log p(f_X|X,Y)$ and $H = -\nabla\nabla \log p(f_X|X,Y)|_{f_X=\hat{f}_X}$ is the Hessian of the negative log posterior at the maxima. Since the denominator of the Bayes' theorem is independent of the latent function, the mode of the posterior can be computed instead from the log un-normalized posterior

$$\Psi(f_X) := \log p(Y|f) + \log p(f_X), \quad (12)$$

with the expression for $p(f_X)$ given in (3). Computation of the mode requires us to evaluate the gradient of $\Psi(f_X)$ which is given as

$$\nabla\Psi(f_X) = \nabla \log p(Y|f) + K^{-1}f_X. \quad (13)$$

To find the stationary point, however, we cannot simply set this gradient to zero as $\nabla \log p(Y|f)$ depends non-linearly on f_X. We need to resort to an iterative scheme based on the Newton–Raphson's method with the update equation given by

$$f_X^{\text{new}} \leftarrow f_X^{\text{old}} - (\nabla\nabla\Psi(f_X))^{-1}\nabla\Psi(f_X), \quad (14)$$

and the Hessian given by

$$\nabla\nabla\Psi(f_X) = -W - K^{-1}, \quad (15)$$

and $W := -\nabla\nabla \log p(Y|f)$ is a diagonal matrix. It is important to note that if the likelihood function $p(Y|f,X)$ is log-concave, the diagonal elements of W are non-negative and the Hessian in (15) is negative definite (since $-K$ and its inverse is negative definite by construction and the sum of two negative definite matrices is also negative definite). Thus, $\Psi(f_X)$ is concave and has a unique maxima point.

Predictive Distribution: The latent function f_X plays the role of a nuisance function, i.e., we do not observe values of f_X itself, and more importantly, we are not particularly interested in the values of f_X. What we are interested in is a class conditional posterior probability, $p(y_* = +1|x_*,X,Y)$ (as the probability of the two classes must sum to 1, $p(y_* = -1|x_*,X,Y) = 1 - p(y_* = +1|x_*,X,Y)$ is a class conditional probability of a class label of not 1) for a novel input x_*.

The inference strategy involves marginalizing out the nuisance function and is divided into two steps. First, we need to compute the distribution of the latent function at the novel input x_*,

$$p(f_*|X,Y,x_*) = \int p(f_*|x_*,X,f_X)p(f_X|X,Y)df_X. \quad (16)$$

The conditional distribution $p(f_*|x_*, X, f_X)$ is computed by invoking the Gaussian process regression model in (6) to arrive at

$$p(f_*|x_*, X, f_X) = \mathcal{N}(k_{X,x_*}^\top K^{-1} f_X, k(x_*, x_*) - k_{X,x_*}^\top K^{-1} k_{X,x_*}). \tag{17}$$

Note that, the underlying Gaussian process regression model is assumed to be a noise-free process. Another approach would be assuming an independent Gaussian noise in combination with a step function likelihood function. However, this is equivalent to the noise-free latent process with a cumulative Gaussian likelihood function (Rasmussen & Williams, 2006). With Laplace approximation of posterior distribution $p(f_X|X, Y) \approx \mathcal{N}(\hat{f}_X, (K^{-1} + W)^{-1})$, we can now compute the integration in (16) by using the standard result for the convolution of two Gaussian distributions. Thus, the conditional distribution is given by

$$p(f_*|x_*, X, Y) = \mathcal{N}(\mathbf{E}[f_*|x_*, X, Y], \mathbf{Var}[f_*|x_*, X, Y]), \tag{18}$$

with

$$\mathbf{E}[f_*|x_*, X, Y] = k_{X,x_*}^\top K^{-1} \hat{f}_X,$$
$$\mathbf{Var}[f_*|x_*, X, Y] = k(x_*, x_*) - k_{X,x_*}^\top (K + W^{-1})^{-1} k_{X,x_*}.$$

The predictive distribution can now be computed as follows

$$\begin{aligned} \pi_* &:= p(y_* = +1|x_*, X, Y) \\ &= \int p(y_* = +1|f_*) p(f_*|x_*, X, Y) df_*. \end{aligned}$$

The above integral can be solved analytically for a cumulative Gaussian likelihood function,

$$\begin{aligned} \pi_* &= \int_{-\infty}^{\frac{\mathbf{E}[f_*|x_*,X,Y]}{(y_*^{-2}+\mathbf{Var}[f_*|x_*,X,Y])^{1/2}}} \mathcal{N}(t|0,1) dt \\ &= \Phi_{0,1}\left(\frac{\mathbf{E}[f_*|x_*, X, Y]}{(y_*^{-2} + \mathbf{Var}[f_*|x_*, X, Y])^{1/2}}\right), \end{aligned}$$

and can be approximated for a logistic likelihood function (MacKay, 1992),

$$\pi_* = \frac{1}{1 + \exp(-\mathbf{E}[f_*|x_*, X, Y]\kappa(\mathbf{Var}[f_*|x_*, X, Y]))},$$

with $\kappa(c) = (1 + c\pi/8)^{-1/2}$.

Point Prediction: Similar to the regression case, we might need to make a point prediction from the predictive distribution described in the section above. For a zero-one loss function, i.e., a loss of one unit is suffered for a wrong classification and 0 for not making a classification mistake, the optimal point prediction (in the sense of expected loss) is

$$y_{\text{optimal}}|x^* = \underset{y_* \in \{1,\ldots,\Omega\}}{\text{argmax}} \; p(y_*|x_*, X, Y). \tag{19}$$

It is worth noting that the probabilistic approach to classification allows the same inference stage to be reused with different loss functions. In some situations, a cost sensitive loss function, i.e., different classification mistakes incur different losses, is more desirable. The optimal point prediction is now taken by minimizing expected cost sensitive loss with respect to the same $p(y_*|x_*, X, Y)$.

Extension of the Laplace approximation to multiclass Gaussian process classification ($\Omega > 2$) (Williams & Barber, 1998) can be achieved via the softmax activation function, i.e., a generalization of logistic activation function.

Practical Issues

We have seen how to do regression and classification using Gaussian processes. Like other kernel based methods such as support vector machines, they are very flexible in that all operations are kernelized, i.e., the operations are performed in the (possibly infinite dimensional) feature space. However, this feature space is only defined implicitly via positive definite kernels (covariance functions), which only requires computation in the (lower dimensional) input space. Compared to other non-Bayesian kernel approaches, Gaussian processes provide an explicit probabilistic formulation of the model. This directly provides us with confidence intervals (for regression) or posterior class probabilities (for classification).

So far, however, we have assumed a covariance function with the known functional form and hyperparameters. In many practical applications, it may not be easy to specify all aspects of the covariance function by hand. Furthermore, inverting the corresponding $N \times N$ Gram matrix is the main computational cost and it may be

prohibitive as it scales as $\mathcal{O}(N^3)$. We will now discuss approaches to overcome both limitations in turn.

Model Selection

In many practical applications, the functional form of the covariance function needs to be chosen and any values of hyperparameters associated with the chosen covariance function and possible free parameters of the likelihood function needs to be optimally determined. This is called model selection.

Ideally, we would like to define a prior distribution over the hyperparameters θ, and predictions are made by integrating over different possible choice of hyperparameters. More formally,

$$p(y_*|x_*, X, Y) = \int p(y_*|x_*, X, Y, \theta)p(\theta|X, Y)d\theta. \tag{20}$$

The evaluation of the above integral, however, may be difficult, and an approximation is needed either by using the most likely value of hyperparameters, $p(y_*|x_*, X, Y) \approx p(y_*|x_*, X, Y, \theta_{\mathrm{ML}})$, or by performing the integration numerically via Monte Carlo methods. We will focus here on the approximation approach and show how to use it for regression and classification problems respectively.

Marginal Likelihood for Regression: The posterior probability of the hyperparameters θ in (20) is

$$p(\theta|X, Y) \propto p(Y|X, \theta)p(\theta), \tag{21}$$

where the first term is known as marginal likelihood or evidence for the hyperparameters and its logarithm is in the form of (from (4))

$$\log p(Y|X, \theta) = -\frac{1}{2}Y^\top \bar{K}^{-1}Y - \frac{1}{2}\log|\bar{K}| - \frac{N}{2}\log(2\pi),$$

with $\bar{K} := K + \sigma^2_{\mathrm{noise}}I$. We can then set the hyperparameters by maximizing this marginal likelihood (We can also maximize the un-normalized posterior instead, assuming finding the derivatives of the priors is straightforward.) (also known as type II maximum likelihood approximation, ML-II) and its partial derivatives with respect to hyperparameters is

$$\frac{\partial}{\partial_{\theta_j}}\log p(Y|X, \theta) = \frac{1}{2}Y^\top \bar{K}^{-1}\frac{\partial \bar{K}}{\partial_{\theta_j}}\bar{K}^{-1}Y - \frac{1}{2}\mathrm{tr}\left(\bar{K}^{-1}\frac{\partial \bar{K}}{\partial_{\theta_j}}\right).$$

Marginal Likelihood for Classification: The Laplace approximation of the marginal likelihood, $p(Y|X, \theta) \approx \hat{p}(Y|X, \theta)$

$$= \int \exp(\Psi(f_X))df_X$$
$$= \exp(\Psi(\hat{f}_X)) \int \exp(-\frac{1}{2}(f_X - \hat{f}_X)^\top H(f_X - \hat{f}_X))df_X,$$

which is achieved via a Taylor expansion of (12) locally around $\hat{f}_X$ to obtain $\Psi(f_X) \approx \Psi(\hat{f}_X) - \frac{1}{2}(f_X - \hat{f}_X)^\top H(f_X - \hat{f}_X)$. Computing the integral analytically gives us the approximate marginal likelihood

$$\log \hat{p}(Y|X, \theta) = -\frac{1}{2}\hat{f}_X K^{-1}\hat{f}_X + \log p(Y|\hat{f}, X) - \frac{1}{2}\log|I + W^{\frac{1}{2}}KW^{\frac{1}{2}}|.$$

Subsequently, the partial derivatives with respect to hyperparameters is given by

$$\frac{\partial}{\partial_{\theta_j}}\log \hat{p}(Y|X, \theta) = \frac{1}{2}\hat{f}_X^\top K^{-1}\frac{\partial K}{\partial_{\theta_j}}K^{-1}\hat{f}_X - \frac{1}{2}\mathrm{tr}\left((K + W^{-1})^{-1}\frac{\partial K}{\partial_{\theta_j}}\right) + \sum_{i=1}^{N}\frac{\partial \log \hat{p}(Y|X, \theta)}{\partial \hat{f}_{x_i}}\frac{\partial \hat{f}_{x_i}}{\partial \theta_j}.$$

The familiar multiple local optima problem is also present in the marginal likelihood maximization. However, practical experiences suggest that local optima are not a devastating problem especially with simple functional forms of covariance function.

Sparse Approximation

A significant problem with Gaussian process model is associated with the computation cost of inverting the $N \times N$ Gram matrix. A number of sparse approximation methods have been proposed to overcome this high computational demand. Common to all these methods is that only a subset of the latent function values of size $M < N$ are treated exactly and the remaining latent values are approximated with cheaper computational demand. Quiñonero-Candela and Rasmussen (2005) describe a unifying view of sparse

approximation. All existing sparse methods are shown to be an instance of it. The framework is described for regression problems, however, it should also be applicable for classification learning settings, albeit with complicacy associated with the non-Gaussian likelihood.

In this unifying treatment, an additional set of M latent variables $f_U \in \mathbb{R}^M$, called inducing variables, are introduced. These latent variables are latent function values corresponding to a set of input locations $X_U \in \mathbb{R}^{M\times d}$, called inducing inputs. The choice of inducing inputs are not restricted to only from the training or test inputs. Due to the marginalization property, introducing more latent variables will not change the distribution of the original variables. Consider (5) with the covariance matrix contains no noise components, that is the distribution now defines joint distribution over latent training and test function values, $p(f_X, f_*|X, x_*)$

$$= \int p(f_X, f_*, f_U|X, x_*)df_U$$

$$= \int p(f_X, f_*|X, x_*, f_U)p(f_U)df_U, \tag{22}$$

with $p(f_U) = \mathcal{N}(0, K_{u,u})$. So far, no approximations have been introduced. Introducing the key assumption which is f_X is conditionally independent of f_* given f_U, $f_* \perp\!\!\!\perp f_X \,|f_U$, allow us to approximate (22) as

$$p(f_X, f_*|X, x_*) \approx \int p(f_*|x_*, f_U)p(f_X|X, f_U)p(f_U)df_U, \tag{23}$$

where $p(f_*|x_*, f_U)$ and $p(f_X|X, f_U)$ admit the same form as (6) without noise components. Different computationally efficient algorithms in the literature correspond to different assumptions made on those two conditional distributions. Table 2 shows various sparse approximation methods with their corresponding approximated conditional distributions. For all sparse approximation methods, the computational complexity is reduced from $\mathcal{O}(N^3)$ to $\mathcal{O}(NM^2)$.

Current and Future Directions

Gaussian processes are an active area of research both within the machine learning and the Bayesian statistics community. First, there is the issue of efficient inference and learning as already discussed in the text above. Second, there is interest in adapting Gaussian processes to other learning settings. They have been used for ordinal regression (Chu & Ghahramani, 2005a; Yu, Yu, Tresp & Kriegel, 2006), preference learning (Chu & Ghahramani, 2005b), ranking (Guiver & Snelson, 2008), mixtures of experts (Tresp, 2000b), transductive learning (Schwaighofer & Tresp, 2003), multi-task learning (Yu, Tresp, & Schwaighofer, 2005), dimensionality reduction (Lawrence, 2005), matrix factorization (Lawrence & Urtasun, 2009), reinforcement learning (Deisenroth & Rasmussen, 2009; Engel, Mannor, & Meir, 2005), among other settings. They have also been extended to handle relational data (Chu, Sindhwani, Ghahramani, & Keerthi, 2006; Kersting & Xu, 2009; Silva, Chu, & Ghahramani, 2007; Xu, Kersting, & Tresp, 2009; Yu, Chu, Yu, Tresp, & Xu, 2006). Standard Gaussian processes only exploit the available information about attributes of instances and typically

Gaussian Process. Table 2 Sparse approximation methods

Method	$\hat{p}(f_X\|X, f_U)$	$\hat{p}(f_*\|x_*, f_U)$	Ref.
SR	$\mathcal{N}(K_{X,X_U}K_{X_U,X_U}^{-1}f_U, 0)$	$\mathcal{N}(K_{x_*,X_U}K_{X_U,X_U}^{-1}f_U, 0)$	Silverman (1985)
PP	$\mathcal{N}(K_{X,X_U}K_{X_U,X_U}^{-1}f_U, 0)$	$p(f_*\|x_*, f_U)$	Seeger, Williams, and Lawrence (2003)
SPGPs	$\mathcal{N}(K_{X,X_U}K_{X_U,X_U}^{-1}f_U, \Delta_1)$ $\Delta_1 = \text{diag}[K_{X,X} - K_{X,X_U}K_{X_U,X_U}^{-1}K_{X_U,X}]$	$p(f_*\|x_*, f_U)$	Snelson and Ghahramani (2006)
BCM	$\mathcal{N}(K_{X,X_U}K_{X_U,X_U}^{-1}f_U, \Delta_2)$ $\Delta_2 = \text{blockdiag}[K_{X,X} - K_{X,X_U}K_{X_U,X_U}^{-1}K_{X_U,X}]$	$p(f_*\|x_*, f_U)$	Tresp (2000a)

SR subset of regressors; *PP* projected process; *SPGPs* sparse pseudo-input gaussian processes; *BCM*: bayesian committe machine

ignore any relations among the instances. Intuitively, however, we would like to use our information about one instance to help us reach conclusions about other, related instances.

Gaussian processes are also of great interest for practical applications because they naturally deal with noisy measurements, unevenly distributed observations, and fill small gaps in the data with high confidence while assigning higher predictive uncertainty in sparsely sampled areas. For instance, Platt (2002) generated music playlists using Gaussian processes. Schwaighofer, Grigoras, Tresp, and Hoffmann (2004) used them for realizing positioning systems using cellular networks. Chu, Ghahramani, and Falciani (2005) proposed a gene selection algorithm based on Gaussian processes to discover consistent gene expression patterns associated with ordinal clinical phenotypes. Brooks, Makarenko, and Upcroft (2006) proposed a Gaussian process model in the context of appearance-based localization with an omni-directional camera. Ferris, Haehnel, and Fox (2006) applied Gaussian processes to locate a mobile robot from wireless signal strength. Plagemann, Fox, and Burgard (2007) used them to detect failures on a mobile robot. Gao, Honkela, Rattray, and Lawrence (2008) inferred latent chemical species in biochemical interaction networks using Gaussian processes. Krause, Singh, and Guestrin (2008) modeled precipitation data using Gaussian processes.

Finally, there is the issue of relaxing the assumption of the standard Gaussian process model that the noise on the output is uniform throughout the domain. If we assume that the noise is a smooth function of the inputs, the noise variance can be modeled using a second Gaussian process, in addition to the process governing the noise-free output values. The posterior distribution of the noise levels can then be sampled using MCMC or approximated using maximum-aposteriori inference. The resulting heteroscedastic, i.e., input-dependent noise regression model has been shown to outperform state-of-the-art methods for mobile robot localization (Plagemann, Kersting, Pfaff, & Burgard, 2007).

In addition to the references embedded in the text above, we also recommend http://www.gaussian-process.org/. A highly recommended textbook is Rasmussen & Williams (2006).

Cross References

►Dirichlet Process

Recommended Reading

Abrahamsen, P. (1992). *A review of Gaussian random fields and correlation functions*. Rapport 917, Norwegian Computing Center, Oslo. www.nr.no/publications/917_Rapport.ps.

Brooks, A., Makarenko, A., & Upcroft, B. (2006). Gaussian process models for sensor-centric robot localisation. In *Proceedings of ICRA*. IEEE.

Chu, W., & Ghahramani, Z. (2005a). Gaussian processes for ordinal regression. *Journal of Machine Learning Research, 6*, 1019–1041.

Chu, W., & Ghahramani, Z. (2005b). Npreference learning with gaussian processes. In: *Proceedings of the international conference on machine learning* (pp. 137–144). New York: ACM.

Chu, W., Ghahramani, Z., Falciani, F., & Wild, D. (2005). Biomarker discovery in microarray gene expression data with Gaussian processes. *Bioinformatics, 21*(16), 3385–3393.

Chu, W., Sindhwani, V., Ghahramani, Z., & Keerthi, S. (2006). Relational learning with gaussian processes. In *Proceedings of neural information processing systems*. Canada: Vancouver.

Deisenroth, M. P., Rasmussen, C. E., & Peters, J. (2009). Gaussian process dynamic programming. *Neurocomputing, 72*(7–9), 1508–1524.

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. In *Proceedings of the international conference on machine learning*, Bonn, Germany (pp. 201–208). New York: ACM.

Ferris, B., Haehnel, D., & Fox, D. (2006). Gaussian processes for signal strength-based location estimation. In *Proceedings of robotics: Science and systems*, Philadelphia, USA. Cambridge, MA: The MIT Press.

Gao, P., Honkela, A., Rattray, M., & Lawrence, N. (2008). Gaussian process modelling of latent chemical species: applications to inferring transcription factor activities. *Bioinformatics 24*(16), i70–i75.

Guiver, J., & Snelson, E. (2008). Learning to rank with softrank and gaussian processes. In *Proceedings of SIGIR*. (pp. 259–266). New York: ACM.

Kersting, K., & Xu, Z. (2009). Learning preferences with hidden common cause relations. In *Proceedings of ECML PKDD*. Berlin: Springer.

Krause, A., Singh, A., & Guestrin, C. (2008). Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research, 9*, 235–284.

Krige, D. G. (1951). A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa, 52*(6), 119–139.

Lawrence, N. (2005). Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of Machine Learning Research, 6*, 1783–1816.

Lawrence, N., & Urtasun, R. (2009). Non-linear matrix factorization with Gaussian processes. In *Proceedings of the international conference on machine learning* (pp. 601–608). New York: ACM.

MacKay, D. J. C. (1992). The evidence framework applied to classification networks. *Neural Computation, 4*(5), 720–736.

Matheron, G. (1963). Principles of geostatistics. *Economic Geology (58)*, 1246–1266.
Neal, R. (1996). Bayesian learning in neural networks. New York: Springer.
Nickisch, H., & Rasmussen, C. E. (2008). Approximations for binary gaussian process classification. *Journal of Machine Learning Research, 9*, 2035–2078.
Plagemann, C., Fox, D., & Burgard, W. (2007). Efficient failure detection on mobile robots using particle filters with gaussian process proposals. In Proceedings of the international joint conference on artificial intelligence (IJCAI), Hyderabad, India. Morgan Kaufmann.
Plagemann, C., Kersting, K., Pfaff, P., & Burgard, W. (2007). Gaussian beam processes: A nonparametric bayesian measurement model for range finders. In *Proceedings of the robotics: Science and systems conference (RSS-07)*, Atlanta, GA, USA. The MIT Press.
John C. Platt., Christopher J. C. Burges., Steven Swenson., Christopher Weare., & Alice Zheng. (2002). Learning a gaussian process prior for automatically generating music playlists. *In Advances in Neural Information Processing Systems*, 1425–1432, MIT Press.
Quiñonero-Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research, 6*, 1939–1959.
Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. Cambridge, MA: MIT Press.
Schwaighofer, A., Grigoras, M., Tresp, V., & Hoffmann, C. (2004). A Gaussian process positioning system for cellular networks. In *Advances in neural information processing systems 16*. Cambridge, MA: MIT Press.
Schwaighofer, A., & Tresp, V. (2003). Transductive and inductive methods for approximate guassian process regression. In *Neural information processing systems*. Cambridge, MA: MIT Press.
Seeger, M., Williams, C. K. I., & Lawrence, N. (2003). Fast forward selection to speed up sparse gaussian process regression. In *Ninth international workshop on artificial intelligence and statistics*. Society for Artificial Intelligence and Statistics.
Silva, R., Chu, W., & Ghahramani, Z. (2007). Hidden common cause relations in relational learning. In *Proceedings of neural information processing systems*. Canada: Vancouver.
Silverman, B. W. (1985). Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal of Royal Statistical Society B, 47*(1), 1–52.
Snelson, E., & Ghahramani, Z. (2006). Sparse gaussian processes using pseudo-inputs. In *Advanes in neural information processing systems* (pp. 1257–1264). The MIT Press.
Tresp, V. (2000a). A Bayesian committee machine. *Neural Computation, 12*(11), 2719–2741.
Tresp, V. (2000b). Mixtures of gaussian processes. In T. K. Leen, T. G. Dietterich, V. Tresp (Eds.), *Advances in neural information processing systems 13* (pp. 654–660). The MIT Press.
Williams, C., & Barber, D. (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI, 20*(12), 1342–1351.
Williams, C., & Rasmussen, C. (1996). Gaussian processes for regression. In D. S. Touretzky, M. C. Mozer, M. E. Hasselmo (Eds.), *Advances in neural information processing systems 8* (Vol. 8, pp. 514–520). Cambridge, MA: MIT Press.
Xu, Z., Kersting, K., & Tresp, V. (2009). Multi-relational learning with gaussian processes. In *Proceedings of the international joint conference on artificial intelligence (IJCAI)*. Morgan Kaufmann.
Yu, K., Chu, W., Yu, S., Tresp, V., & Xu, Z. (2006). Stochastic relational models for discriminative link prediction. In *Proceedings of neural information processing systems*. Canada: Vancouver.
Yu, K., Tresp, V., & Schwaighofer, A. (2005). Learning gaussian processes from multiple tasks. In *Proceedings of the international conference on machine learning* (pp. 1012–1019). New York: ACM.
Yu, S., Yu, K., Tresp, V., & Kriegel, H. P. (2006). Collaborative ordinal regression. In W. Cohen, A. Moore (Eds.), *Proceedings of the 23rd international conference on machine learning* (pp. 1089–1096). New York: ACM.

Gaussian Process Reinforcement Learning

Yaakov Engel
University of Alberta, Edmonton, Alberta, Canada

Definition

Gaussian process reinforcement learning generically refers to a class of ►reinforcement learning (RL) algorithms that use Gaussian processes (GPs) to model and learn some aspect of the problem.

Such methods may be divided roughly into two groups:

1. *Model-based methods*: Here, GPs are used to learn the transition and reward model of the ►Markov decision process (MDP) underlying the RL problem. The estimated MDP model is then used to compute an approximate solution to the true MDP.
2. *Model-free methods*: Here no explicit representation of the MDP is maintained. Rather, GPs are used to learn either the MDP's value function, state-action value function, or some other quantity that may be used to solve the MDP.

This entry is concerned with the latter class of methods, as these constitute the majority of published research in this area.

Motivation and Background

►Reinforcement learning is a class of learning problems concerned with achieving long-term goals in unfamiliar, uncertain, and dynamic environments. Such tasks are conventionally formulated by modeling the environment as a ►MDPs (Or more generally as partially observable MDPs (►POMDPs).), and modeling the agent as an adaptive controller implementing an action-selection policy.

Markov Decision Processes

Let us denote by $\mathcal{P}(\mathcal{S})$ the set of probability distributions over (Borel) subsets of a set $\mathcal{S}$. A discrete time MDP is a tuple $(\mathcal{X}, \mathcal{U}, p_0, p, q, \gamma)$, where $\mathcal{X}$ and $\mathcal{U}$ are the state and action spaces, respectively; $p_0(\cdot) \in \mathcal{P}(\mathcal{X})$ is a probability density over initial states; $p(\cdot|\mathbf{x}, \mathbf{u}) \in \mathcal{P}(\mathcal{X})$ is a probability density over successor states, conditioned on the current state $\mathbf{x}$ and action $\mathbf{u}$; $q(\cdot|\mathbf{x}, \mathbf{u}) \in \mathcal{P}(\mathbb{R})$ is a probability distribution over immediate single-step rewards, conditioned on the current state and action. We denote by $R(\mathbf{x}, \mathbf{u})$ the random variable distributed according to $q(\cdot|\mathbf{x}, \mathbf{u})$. Finally, $\gamma \in [0,1]$ is a discount factor. We assume that both p and q are stationary, that is, they do not depend explicitly on time. To maintain generality, we use $\mathbf{z}$ to denote either a state $\mathbf{x}$ or a state-action pair $(\mathbf{x}, \mathbf{u})$. This overloaded notation will allow us to present models and algorithms in a concise and unified form.

In the context of control it is useful to make several additional definitions. A *stationary policy* $\mu(\cdot|\mathbf{x}) \in \mathcal{P}(\mathcal{U})$ is a time-independent mapping from states to action selection probabilities. A stationary policy μ induces a Markov reward process (MRP) (Puterman, 1994) via *policy-dependent* state-transition probability density, defined as (Here and in the sequel, whenever integration is performed over a finite or discrete space, the integral should be understood as a summation.)

$$p^{\mu}_{\mathbf{x}}(\mathbf{x}'|\mathbf{x}) = \int_{\mathcal{U}} \mathrm{d}\mathbf{u}\, \mu(\mathbf{u}|\mathbf{x}) p(\mathbf{x}'|\mathbf{u}, \mathbf{x}).$$

Similarly, the policy μ may also be used to define a state-action transition probability density, defined as

$$p^{\mu}_{\mathbf{x},\mathbf{u}}(\mathbf{x}', \mathbf{u}'|\mathbf{x}, \mathbf{u}) = p(\mathbf{x}'|\mathbf{u}, \mathbf{x}) \mu(\mathbf{u}'|\mathbf{x}').$$

Using our overloaded notational convention, we refer to either of these as $p^{\mu}_{\mathbf{z}}$. Let us denote by $\boldsymbol{\xi}(\mathbf{z})$ a path that starts at $\mathbf{z}$. Hence, for a fixed policy μ and a fixed initial state or state-action pair $\mathbf{z}_0$, the probability (density) of observing the path $\boldsymbol{\xi}(\mathbf{z}_0) = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_t)$ of length t is (we take $\mathbf{z}_0$ as given) $\mathbb{P}(\boldsymbol{\xi}(\mathbf{z}_0)) = \prod_{i=1}^{t} p^{\mu}_{\mathbf{z}}(\mathbf{z}_i|\mathbf{z}_{i-1})$. The *discounted return* $D^{\mu}(\boldsymbol{\xi}(\mathbf{z}))$ of a path $\boldsymbol{\xi}(\mathbf{z})$ is a random process, defined (with some abuse of notation) as

$$D^{\mu}(\mathbf{z}) = D^{\mu}(\boldsymbol{\xi}(\mathbf{z})) = \sum_{i=0}^{\infty} \gamma^i R(\mathbf{z}_i) | (\mathbf{z}_0 = \mathbf{z}), \qquad (1)$$

where $\gamma \in [0,1]$ is the discount factor (When $\gamma = 1$ the policy must be proper, see Bertsekas and Tsitsiklis (1996).) The randomness in $D^{\mu}(\mathbf{z})$, for any given $\mathbf{z}$, is due both to $\boldsymbol{\xi}(\mathbf{z})$ being a random process and to the randomness, or noise, in the rewards $R(\mathbf{z}_0), R(\mathbf{z}_1), \ldots$, etc., both of which jointly constitute the *intrinsic* randomness of the MDP. Equation (1) together with the stationarity of the MDP yield the recursive formula

$$D^{\mu}(\mathbf{z}) = R(\mathbf{z}) + \gamma D^{\mu}(\mathbf{z}') \quad \text{where } \mathbf{z}' \sim p^{\mu}_{\mathbf{z}}(\cdot|\mathbf{z}). \qquad (2)$$

Let us define the expectation operator $\mathbf{E}_{\xi}$ as the expectation over all possible trajectories and all possible rewards collected in them. This allows us to define the *value function* $V^{\mu}(\mathbf{z})$ as the result of applying this expectation operator to the discounted return $D^{\mu}(\mathbf{z})$, i.e.,

$$V^{\mu}(\mathbf{z}) = \mathbf{E}_{\xi} D^{\mu}(\mathbf{z}). \qquad (3)$$

Applying the law of total expectation to this equation results in the MRP (fixed policy) version of the Bellman equation:

$$V^{\mu}(\mathbf{z}) = R(\mathbf{z}) + \gamma \mathbf{E}_{\mathbf{z}'|\mathbf{z}}[V^{\mu}(\mathbf{z}')]. \qquad (4)$$

A policy that maximizes the expected discounted return from each state is called an optimal policy, and is denoted by μ^*. In the case of stationary MDPs, there exists a *deterministic* optimal policy (This is no longer the case for POMDPs and Markov games, see Kaelbling, Littman, and Cassandra (1998) and Littman (1994)). The value function corresponding to an optimal policy is called the optimal value, and is denoted by $V^* = V^{\mu^*}$. While there may exist more than one optimal policy, the optimal value function is unique (Bertsekas, 1995).

Reinforcement Learning

Many of the algorithms developed for solving RL problems may be traced back to the ►dynamic programming *Value Iteration* and *Policy Iteration* algorithms (Bellman, 1957; Bertsekas, 1995; Bertsekas & Tsitsiklis, 1996; Howard, 1960). However, there are two major features distinguishing RL from the traditional planning framework. First, while in planning it is assumed that the environment is fully known, in RL no such assumption is made. Second, the learning process in RL is usually assumed to take place *online*, namely, concurrently with the acquirement of data by the learning agent as it interacts with its environment. These two features make solving RL problems a significantly more challenging undertaking.

An important algorithmic component of policy-iteration-based RL algorithms is the estimation of either state or state-action values of a fixed policy controlling a MDP, a task known as *policy evaluation*. Sutton's TD(λ) algorithm (Sutton, 1984) is an early RL algorithm that performs policy evaluation based on observed sample trajectories from the MDP, while it is being controlled by the policy being evaluated (see ►Temporal Difference Learning). In its original formulation, TD(λ) as well as many other algorithms (e.g., Watkins' ►Q-learning (1989)), employs a lookup table to store values corresponding to the MDP's states or state-action pairs. This approach clearly becomes infeasible when the size of the MDPs joint state-action space exceeds the memory capacity of modern workstations. One solution to this problem is to represent the value function using a parametric function approximation architecture, and allow these algorithms to estimate the parameters of approximate value functions. Unfortunately, with few exceptions, this seemingly benign modification turns out to have ruinous consequences to the convergence properties of these algorithms. One notable exception is TD(λ), when it is used in conjunction with a function approximator $\hat{V}(\mathbf{z}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{z})$, which is linear in its tunable parameters $\mathbf{w} = (w_1, \ldots, w_N)^\top$. Under certain technical conditions, it has been shown that in this case, TD(λ) converges almost surely, and the limit of convergence is "close" (in a well defined manner) to a projection ΠV^μ of the true value function V^μ onto the finite-dimensional space $\mathcal{H}_\phi$ of functions spanned by $\{\phi_i | i = 1, \ldots, N\}$ (Tsitsiklis & Van Roy, 1996). Note that this projection is the best one may hope for, as long as one is restricted to a fixed function approximation architecture. In fact, when $\lambda = 1$, the bound of Tsitsiklis and Van Roy (1996) implies that TD(1) converges to ΠV^μ (assuming it is unique). However, as λ is reduced toward 0, the quality of TD(λ)'s solution may deteriorate significantly. If V^μ happens to belong to $\mathcal{H}_\phi$, then $V^\mu = \Pi V^\mu$ and TD(λ) converges almost surely to V^μ, for any $\lambda \in [0,1]$.

As noted in Bertsekas and Tsitsiklis (1996), TD(λ) is a stochastic approximation algorithm (Kushner & Yin, 1997). As such, to ensure convergence to a meaningful result, it relies on making small and diminishing updates to its value-function estimates. Moreover, in the typical on-line mode of operation of TD(λ), a sample is observed, acted upon (by updating the parameters of $\hat{V}$) and is then discarded, never to be seen again. A negative consequence of these two properties is that on-line TD(λ) is inherently wasteful in its use of the observed data. The least-squares TD(λ), or LSTD(λ) algorithm (Boyan, 1999; Bradtke & Barto, 1996), was put forward as an alternative to TD(λ) that makes better use of data, by directly solving a set of equations characterizing the fixed point of the TD(λ) updates. LSTD(λ) is amenable to a recursive implementation, at a time and memory cost of $O(N^2)$ per sample. A more fundamental shortcoming, shared by both TD(λ) and LSTD(λ) is that they do not supply the user with a measure of the accuracy of their value predictions.

The discussion above motivates the search for:

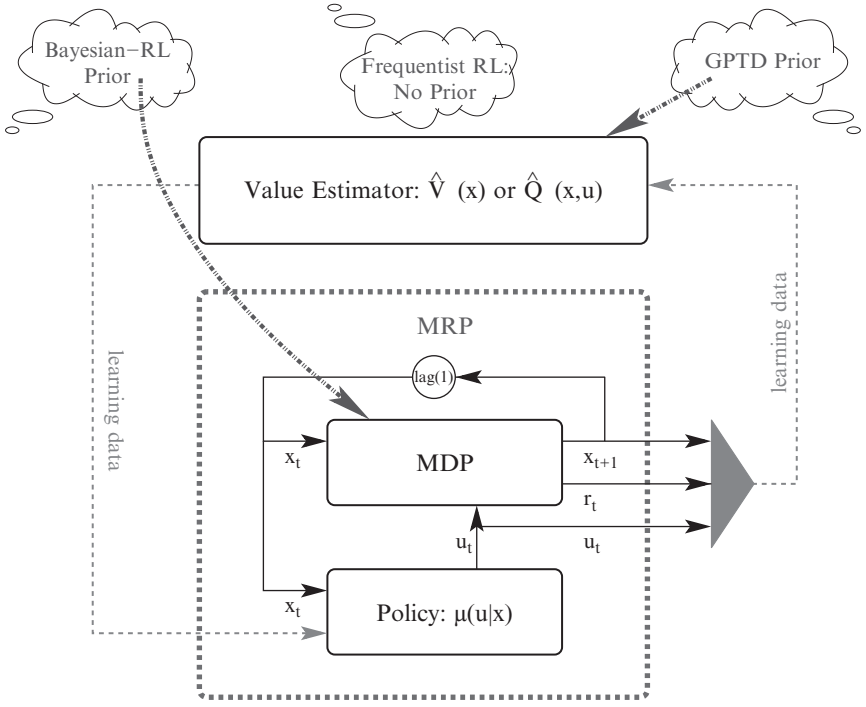
1. Nonparametric estimators for V^μ, since these are not generally restricted to searching in any finite dimensional hypothesis space, such as $\mathcal{H}_\phi$.
2. Estimators that make efficient use of the data.
3. Estimators that, in addition to value predictions, deliver a measure of the uncertainty in their predictions.

Structure of Learning System

We first describe the structure and operation of the basic GP temporal differences (GPTD) algorithm for policy evaluation. We then build on this algorithm to describe policy improving algorithms, in the spirit of Howard's Policy Iteration (Howard, 1960).

In the preceding section we showed that the value V is the result of taking the expectation of the discounted return D with respect to the randomness in

the trajectories and in the rewards collected therein. In the classic, or frequentist approach V is no longer random, since it is the true, albeit unknown value function induced by the policy μ. Adopting the Bayesian approach, we may still view the value V as a random entity by assigning it additional randomness, that is due to our *subjective uncertainty* regarding the MDP's transition model (p, q). We do not know what the true distributions p and q are, which means that we are also uncertain about the true value function. Previous attempts to apply Bayesian reasoning to RL modeled this uncertainty by placing priors over the MDP's transition and reward model (p, q) and applying Bayes' rule to update a posterior based on observed transitions. This line of work may be traced back to the pioneering works of Bellman and Howard (Bellman, 1956; Howard, 1960) followed by more recent contributions in the machine learning literature (Dearden, Friedman, & Andre, 1999; Dearden, Friedman, & Russell, 1998; Duff, 2002; Mannor, Simester, Sun, & Tsitsiklis, 2004; Poupart, Vlassis, Hoey, & Regan, 2006; Strens, 2000; Wang, Lizotte, Bowling, & Schuurmans, 2005). A fundamental shortcoming of this approach is that the resulting algorithms are limited to solving MDPs with finite (and typically rather small) state and action spaces, due to the need to maintain a probability distribution over the MDP's transition model. In this work, we pursue a different path – we choose to model our uncertainty about the MDP by placing a prior (and updating a posterior) directly on V. We achieve this by modeling V as a random process, or more specifically, as a Gaussian Process. This mirrors the traditional classification of classical RL algorithms to either model-based or model-free (direct) methods, see Chapter 9 in Sutton and Barto (1998). Figure 1 illustrates these different approaches.



Gaussian Process Reinforcement Learning. Figure 1. An illustration of the frequentist as well as the two different Bayesian approaches to value-function based reinforcement learning. In the traditional Bayesian RL approach a prior is placed on the MDP's model, whereas in our GPTD approach the prior is placed directly on the value function. x, u, and *r* denote state, action, and reward, respectively. The data required to learn value estimators typically consists of a temporal stream of state-action-reward triplets. Another stream of data is used to update the policy based on the current estimate of the value function. A MDP and a stationary policy controlling it, jointly constitute a MRP. lag(1) denotes the 1-step time-lag operator

Gaussian Process Temporal Difference Learning

GPTD should be viewed as a family of *statistical generative models* (see ▶Generative Learning) for value functions, rather than as a family of algorithms. As such, GPTD models specify the statistical relation between the unobserved value function and the observable quantities, namely the observed trajectories and the rewards collected in them. The set of equations prescribing the GPTD model for a path $\boldsymbol{\xi} = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_t)$ is (Here and in the sequel, to simplify notation, we omit the superscript μ, with the understanding that quantities such as D, V, or $\boldsymbol{\xi}$ generally depend on the policy μ being evaluated.)

$$R(\mathbf{z}_i) = V(\mathbf{z}_i) - \gamma V(\mathbf{z}_{i+1}) + N(\mathbf{z}_i, \mathbf{z}_{i+1}) \quad \text{for } i = 0, 1, \ldots, t-1.$$

$N(\mathbf{z}_i, \mathbf{z}_{i+1})$ is a zero-mean noise term that must account for the statistics of $R(\mathbf{z}_i) + \gamma V(\mathbf{z}_{i+1}) - V(\mathbf{z}_i)$. If V is a priori distributed according to a GP prior, and the noise term $N(\mathbf{z}_i, \mathbf{z}_{i+1})$ is also normally distributed then $R(\mathbf{z}_i)$ is also normally distributed, and so is the posterior distribution of V conditioned on the observed rewards. To fully specify the GPTD model, we need to specify the GP prior over V in terms of prior mean and covariance as well as the covariance of the noise process N. In Engel, Mannor, and Meir (2003) it was shown that modeling N as a white noise process is a suitable choice for MRPs with deterministic transition dynamics. In Engel, Mannor, and Meir (2005) a different, correlated noise model was shown to be useful for general MRPs. Let us define $R_t = (R(\mathbf{z}_0), \ldots, R(\mathbf{z}_t))$, $V_t = (V(\mathbf{z}_0), \ldots, V(\mathbf{z}_t))$, and $N_t = (N(\mathbf{z}_0, \mathbf{z}_1), \ldots, N(\mathbf{z}_{t-1}, \mathbf{z}_t))$, also define the $t \times (t+1)$ matrix

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & 0 & \ldots & 0 \\ 0 & 1 & -\gamma & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 & -\gamma \end{bmatrix}.$$

In the white-noise and correlated-noise GPTD models the noise covariance matrices $\boldsymbol{\Sigma}_t = \mathrm{Cov}[N_t]$ are given, respectively, by

$$\begin{bmatrix} \sigma_R^2(\mathbf{z}_0) & 0 & \ldots & 0 \\ 0 & \sigma_R^2(\mathbf{z}_1) & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \sigma_R^2(\mathbf{z}_{t-1}) \end{bmatrix}$$

and

$$\mathbf{H}_t \begin{bmatrix} \sigma_0^2 & 0 & \ldots & 0 \\ 0 & \sigma_1^2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \sigma_t^2 \end{bmatrix} \mathbf{H}_t^\top.$$

The final component of the GPTD model remaining to be specified is the prior distribution of the GP V. This distribution is specified by prior mean and covariance functions $v_0(\mathbf{z})$ and $k(\mathbf{z}, \mathbf{z}')$, respectively.

Let us define $\boldsymbol{v}_t = (v_0(\mathbf{z}_0), \ldots, v_0(\mathbf{z}_t))^\top$. Employing ▶Bayes' rule, the posterior distribution of $V(\mathbf{z})$ – the value function at some arbitrary query point $\mathbf{z}$ – is now given by

$$(V(\mathbf{z}) | R_{t-1} = \mathbf{r}_{t-1}) \sim \mathcal{N}\{\hat{V}_t(\mathbf{z}), P_t(\mathbf{z}, \mathbf{z}),$$

where

$$\begin{aligned} \hat{V}_t(\mathbf{z}) &= v_0(\mathbf{z}) + \mathbf{k}_t(\mathbf{z})^\top \boldsymbol{\alpha}_t, \quad P_t(\mathbf{z}, \mathbf{z}') \\ &= k(\mathbf{z}, \mathbf{z}') - \mathbf{k}_t(\mathbf{z})^\top \mathbf{C}_t \mathbf{k}_t(\mathbf{z}'), \\ \boldsymbol{\alpha}_t &= \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1} (\mathbf{r}_{t-1} - \mathbf{H}_t \boldsymbol{v}_t), \\ \mathbf{C}_t &= \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1} \mathbf{H}_t. \end{aligned}$$

It is seen here that in order to compute the posterior distribution of V for arbitrary sets of query points, one only needs the vector $\boldsymbol{\alpha}_t$ and the matrix $\mathbf{C}_t$. Consequently, $\boldsymbol{\alpha}_t$ and $\mathbf{C}_t$ are sufficient statistics for the posterior of V.

Algorithms 1 and 2 provide pseudocode for recursive computations of these sufficient statistics, in the deterministic-transitions and general MDP models, respectively.

It can be seen that after observing t sample transitions, both the algorithms require storage quadratic in t (due to the matrix $\mathbf{C}_t$). The updates also require time quadratic in t due to matrix-vector products involving

$\mathbf{C}_t$. These properties are unsatisfying from a practical point of view, since realistic RL problems typically require large amounts of data to learn. There are two general approaches for reducing the memory and time footprints of GPTD. One approach is to define parametric counterparts of the two GPTD models described earlier, and derive the corresponding recursive algorithms. If the number of independent parameters (i.e., the dimensionality of the hypothesis space $\mathcal{H}_\phi$) used to represent the value function is m, the memory and time costs of the algorithms become quadratic in m, rather than t. Another approach, which is based on an efficient sequential kernel sparsification method, allows us to selectively exclude terms from $\mathcal{D}_t$, while controlling the error incurred as a result. Here again (Bounds on m in this case may be derived using arguments based on the finiteness of packing numbers of the hypothesis space, see Engel (2005) for details.), if the size of $\mathcal{D}_t$ saturates at m, the memory and time costs of the resulting algorithms are quadratic in m. For the complete derivations, as well as detailed pseudocode of the corresponding algorithms we refer the reader to Engel (2005).

Algorithm 1 Recursive nonparametric GPTD for deterministic MDPs

Initialize $\boldsymbol{\alpha}_0 = 0$, $\mathbf{C}_0 = 0$, $\mathcal{D}_0 = \{\mathbf{z}_0\}$

for $t = 1, 2, \ldots$

observe $\mathbf{z}_{t-1}, r_{t-1}, \mathbf{z}_t$

$\mathbf{h}_t = (0, \ldots, 1, -\gamma)^\top$

$\Delta\mathbf{k}_t = \mathbf{k}_{t-1}(\mathbf{z}_{t-1}) - \gamma\mathbf{k}_{t-1}(\mathbf{z}_t)$

$\Delta k_{tt} = k(\mathbf{z}_{t-1}, \mathbf{z}_{t-1}) - 2\gamma k(\mathbf{z}_{t-1}, \mathbf{z}_t) + \gamma^2 k(\mathbf{z}_t, \mathbf{z}_t)$

$$\mathbf{c}_t = \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\Delta\mathbf{k}_t \\ 0 \end{pmatrix}$$

$d_t = r_{t-1} - \Delta\mathbf{k}_t{}^\top \boldsymbol{\alpha}_{t-1}$

$s_t = \sigma_{t-1}^2 + \Delta k_{tt} - \Delta\mathbf{k}_t{}^\top \mathbf{C}_{t-1}\Delta\mathbf{k}_t$

$$\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t$$

$$\mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\mathbf{c}_t\mathbf{c}_t^\top$$

$\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{z}_t\}$

end for

return $\boldsymbol{\alpha}_t$, $\mathbf{C}_t$, $\mathcal{D}_t$

Theory

In this section we derive the two GPTD models mentioned above, explicitly stating the assumptions underlying each model.

MRPs with Deterministic Transitions

In the deterministic case, the Bellman equation (4) degenerates into

$$\bar{R}(\mathbf{z}) = V(\mathbf{z}) - \gamma V(\mathbf{z}'), \tag{5}$$

where $\mathbf{z}'$ is the state or state-action pair succeeding $\mathbf{z}$, under the deterministic policy μ. We also assume that the noise in the rewards is independent and Gaussian, but not necessarily identically distributed. We denote the reward variance by $\sigma_R^2(\mathbf{z}) = \mathrm{Var}[R(\mathbf{z})]$. Formally, this means that the reward $R(\mathbf{z})$, at some $\mathbf{z}$, satisfies $R(\mathbf{z}) = \bar{R}(\mathbf{z}) + N(\mathbf{z})$ where $\bar{R}(\mathbf{z})$ is the mean reward for that state. Assume we have a sequence of rewards sampled along a sampled path $\boldsymbol{\xi}$. Then, at the ith time step we have $R(\mathbf{z}_i) = \bar{R}(\mathbf{z}_i) + N(\mathbf{z}_i)$. Using the random vectors R_t, V_t, and N_t defined earlier, we have $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_t)$, where

$$\boldsymbol{\Sigma}_t = \mathrm{diag}(\sigma_R^2(\mathbf{z}_0), \ldots, \sigma_R^2(\mathbf{z}_{t-1})), \tag{6}$$

and $\mathrm{diag}(\cdot)$ denotes a diagonal matrix whose diagonal elements are the components of the argument vector. Writing the Bellman equations (5) for the points belonging to the sample path, and substituting $R(\mathbf{z}_i) = \bar{R}(\mathbf{z}_i) + N(\mathbf{z}_i)$, we obtain the following set of t equations

$$R(\mathbf{z}_i) = V(\mathbf{z}_i) - \gamma V(\mathbf{z}_{i+1}) + N(\mathbf{z}_i), \quad i = 0, 1, \ldots, t-1.$$

This set of linear equations may be concisely written as

$$R_{t-1} = \mathbf{H}_t V_t + N_t. \tag{7}$$

General MRPs

Let us consider a decomposition of the discounted return D into its mean V and a zero-mean residual ΔV:

$$D(\mathbf{z}) = \mathbf{E}_\xi D(\mathbf{z}) + \big(D(\mathbf{z}) - \mathbf{E}_\xi D(\mathbf{z})\big) \stackrel{\text{def}}{=} V(\mathbf{z}) + \Delta V(\mathbf{z}). \tag{8}$$

This decomposition is useful, since it separates the two sources of uncertainty inherent in the discounted return

Algorithm 2 Recursive nonparametric GPTD for general MDPs

Initialize $\boldsymbol{\alpha}_0 = \mathbf{0}$, $\mathbf{C}_0 = 0$, $\mathcal{D}_0 = \{\mathbf{z}_0\}$, $\mathbf{c}_0 = \mathbf{0}$, $d_0 = 0, 1/s_0 = 0$

for $t = 1, 2, \ldots$

observe $\mathbf{z}_{t-1}$, r_{t-1}, $\mathbf{z}_t$

$$\mathbf{h}_t = (0, \ldots, 1, -\gamma)^\top$$

$$\Delta\mathbf{k}_t = \mathbf{k}_{t-1}(\mathbf{z}_{t-1}) - \gamma\mathbf{k}_{t-1}(\mathbf{z}_t)$$

$$\Delta k_{tt} = k(\mathbf{z}_{t-1}, \mathbf{z}_{t-1}) - 2\gamma k(\mathbf{z}_{t-1}, \mathbf{z}_t) + \gamma^2 k(\mathbf{z}_t, \mathbf{z}_t)$$

$$\mathbf{c}_t = \frac{\gamma\sigma_{t-1}^2}{s_{t-1}} \begin{pmatrix} \mathbf{c}_{t-1} \\ 0 \end{pmatrix} + \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\Delta\mathbf{k}_t \\ 0 \end{pmatrix}$$

$$d_t = \frac{\gamma\sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \Delta\mathbf{k}_t{}^\top \boldsymbol{\alpha}_{t-1}$$

$$s_t = \sigma_{t-1}^2 + \gamma^2\sigma_t^2 - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}} + \Delta k_{tt} - \Delta\mathbf{k}_t{}^\top \mathbf{C}_{t-1}\Delta\mathbf{k}_t + \frac{2\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{c}_{t-1}^\top\Delta\mathbf{k}_t$$

$$\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t$$

$$\mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\mathbf{c}_t\mathbf{c}_t^\top$$

$$\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{z}_t\}$$

end for

return $\boldsymbol{\alpha}_t$, $\mathbf{C}_t$, $\mathcal{D}_t$

process D: For a known MDP model, V is a (deterministic) function and the randomness in D is fully attributed to the intrinsic randomness in the trajectories generated by the MDP and policy pair, modeled by ΔV. On the other hand, in a MDP in which both transitions and rewards are deterministic but otherwise unknown, ΔV is deterministic (identically zero), and the randomness in D is due solely to the extrinsic Bayesian uncertainty, modeled by the random process V.

Substituting (8) into (2) and rearranging we get

$$R(\mathbf{z}) = V(\mathbf{z}) - \gamma V(\mathbf{z}') + N(\mathbf{z}, \mathbf{z}'),$$

where $\mathbf{z}' \sim p^\mu(\cdot|\mathbf{z})$, and

$$N(\mathbf{z}, \mathbf{z}') \stackrel{\text{def}}{=} \Delta V(\mathbf{z}) - \gamma\Delta V(\mathbf{z}'). \tag{9}$$

As before, we are provided with a sample path $\boldsymbol{\xi}$, and we may write the model equations (9) for these samples, resulting in the following set of t equations

$$R(\mathbf{z}_i) = V(\mathbf{z}_i) - \gamma V(\mathbf{z}_{i+1}) + N(\mathbf{z}_i, \mathbf{z}_{i+1}) \text{ for } i = 0, \ldots, t-1.$$

Using our standard definitions for R_t, V_t, $\mathbf{H}_t$ and with $N_t = (N(\mathbf{z}_0, \mathbf{z}_1), \ldots, N(\mathbf{z}_{t-1}, \mathbf{z}_t))^\top$, we again have

$$R_{t-1} = \mathbf{H}_t V_t + N_t. \tag{10}$$

In order to fully define a complete probabilistic generative model, we also need to specify the distribution of the noise process N_t. We model the residuals $\Delta V_t = (\Delta V(\mathbf{z}_0), \ldots, \Delta V(\mathbf{z}_t))^\top$ as random Gaussian noise (This may not be a correct assumption in general; however, in the absence of any prior information concerning the distribution of the residuals, it is the *simplest* assumption we can make, since the Gaussian distribution possesses the highest entropy among all distributions with the same covariance. It is also possible to relax the Gaussianity requirement on both the prior and the noise. The resulting estimator may then be shown to be the *linear minimum mean-squared error* estimator for the value.). In particular, this means that the distribution of the vector ΔV_t is completely specified by its mean and covariance. Another assumption we make is that each of the residuals $\Delta V(\mathbf{z}_i)$ is independently distributed. Denoting $\sigma_i^2 = Var[D(\mathbf{z}_i)]$, the distribution of ΔV_t is given by:

$$\Delta V_t \sim \mathcal{N}(\mathbf{0}, \text{diag}(\boldsymbol{\sigma}_t)),$$

where $\boldsymbol{\sigma}_t = \left(\sigma_0^2, \sigma_1^2, \ldots, \sigma_t^2\right)^\top$. Since $N_t = \mathbf{H}_t\Delta V_t$, we have $N_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_t)$ with,

$$\boldsymbol{\Sigma}_t = \mathbf{H}_t \text{diag}(\boldsymbol{\sigma}_t)\mathbf{H}_t^\top$$

$$= \begin{bmatrix} \sigma_0^2 + \gamma^2\sigma_1^2 & -\gamma\sigma_1^2 & 0 & \ldots & 0 & 0 \\ -\gamma\sigma_1^2 & \sigma_1^2 + \gamma^2\sigma_2^2 & -\gamma\sigma_2^2 & 0 & \ldots & 0 \\ 0 & -\gamma\sigma_2^2 & \sigma_2^2 + \gamma^2\sigma_3^2 & \ddots & & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \vdots & & \ddots & \ddots & -\gamma\sigma_{t-1}^2 \\ 0 & 0 & \ldots & 0 & -\gamma\sigma_{t-1}^2 & \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix}. \tag{11}$$

Applications

Any RL algorithm that requires policy evaluation as an algorithmic component can potentially use a GPTD algorithm for this task. In particular, this is true of algorithms based on Howard's Policy Iteration. In Engel

et al. (2005) and Engel (2005) is shown how GPTD may be used to construct a SARSA-type algorithm (Rummery & Niranjan, 1994; Sutton & Barto, 1998), called GPSARSA. In Engel, Szabo, and Volkinshtein (2005), GPSARSA was used to learn control policies for a simulated Octopus arm. In Ghavamzadeh and Engel (2007) GPTD was used within a Bayesian actor–critic learning algorithm.

Future Directions

By virtue of the posterior covariance, GPTD algorithms compute a confidence measure (or more precisely, Bayesian *credible intervals*) for their value estimates. So far, little use has been made of this additional information. Several potential uses of the posterior covariance may be envisaged:

1. It may be used to construct stopping rules for value estimation.
2. It may be used to guide exploration.
3. In the context of Bayesian actor–critic algorithms (Ghavamzadeh & Engel, 2007), it may used to control the size and direction of policy updates.

Further Reading

Yaakov Engel's doctoral thesis (Engel, 2005) is currently the most complete reference to GPTD methods. Two conference papers (Engel et al., 2003, 2005) provide a more concise view. The first of these introduces the GPTD model for deterministic MRPs, while the second introduces the general MDP model, as well as the GPSARSA algorithm. A forthcoming journal article will subsume these two papers, and include some additional results, concerning the connection between GPTD and the popular TD(λ) and LSTD(λ) algorithms.

Recommended Reading

Bellman, R. E. (1956). A problem in the sequential design of experiments. *Sankhya, 16*, 221–229.

Bellman, R. E. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control*. Belmont, MA: Athena Scientific.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Boyan, J. A. (1999). Least-squares temporal difference learning. In *Proceedings of the 16th international conference on machine learning* (pp. 49–56). San Francisco: Morgan Kaufmann.

Bradtke, S. J., & Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning, 22*, 33–57.

Dearden, R., Friedman, N., & Andre, D. (1999). Model based Bayesian exploration. In *Proceedings of the fifteenth conference on uncertainty in artificial intelligence* (pp. 150–159). San Francisco: Morgan Kaufmann.

Dearden, R., Friedman, N., & Russell, S. (1998). Bayesian Q-learning. In *Proceedings of the fifteenth national conference on artificial intelligence* (pp. 761–768). Menlo Park, CA: AAAI Press.

Duff, M. (2002). *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts, Amherst.

Engel, Y. (2005). *Algorithms and representations for reinforcement learning*. PhD thesis, The Hebrew University of Jerusalem.

Engel, Y., Mannor, S., & Meir, R. (2003). Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proceedings of the 20th international conference on machine learning*. San Francisco: Morgan Kaufmann.

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd international conference on machine learning*.

Engel, Y., Szabo, P., & Volkinshtein, D. (2005). *Learning to control an Octopus arm with Gaussian process temporal difference methods*. Technical report, Technion Institute of Technology. www.cs.ualberta.ca/~yaki/reports/octopus.pdf.

Ghavamzadeh, M., & Engel, Y. (2007). Bayesian actor-critic algorithms. In Z. Ghahramani (Ed.), *24th international conference on machine learning*. Corvallis, OR: Omnipress.

Howard, R. (1960). *Dynamic programming and Markov processes*. Cambridge, MA: MIT Press.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*, 99–134.

Kushner, H. J., & Yin, C. J. (1997). *Stochastic approximation algorithms and applications*. Berlin: Springer.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th international conference on machine learning (ICML-94)* (pp. 157–163). New Brunswick, NJ: Morgan Kaufmann.

Mannor, S., Simester, D., Sun, P., & Tsitsiklis, J. N. (2004). Bias and variance in value function estimation. In *Proceedings of the 21st international conference on machine learning*.

Poupart, P., Vlassis, N. A., Hoey, J., & Regan, K. (2006). An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the twenty-third international conference on machine learning* (pp. 697–704). Pittsburgh, PA.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: Wiley.

Rummery, G., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems*. Technical report CUED/F-INFENG/TR 166, Cambridge University Engineering Department.

Strens, M. (2000). A Bayesian framework for reinforcement learning. In *Proceedings of the 17th international conference on machine learning* (pp. 943–950). San Francisco: Morgan Kaufmann.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts, Amherst.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Tsitsiklis, J. N., & Van Roy, B. (1996). *An analysis of temporal-difference learning with function approximation*. Technical report LIDS-P-2322, Cambridge, MA: MIT Press.
Wang, T., Lizotte, D., Bowling, M., & Schuurmans, D. (2005). Bayesian sparse sampling for on-line reward optimization. In *Proceedings of the 22nd international conference on machine learning* (pp. 956–963). New York: ACM Press.
Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, UK.

Generality And Logic

►Logic of Generality

Generalization

Claude Sammut
University of New South Wales, Sydney, Australia

A hypothesis, h, is a predicate that maps an instance to *true* or *false*. That is, if $h(x)$ is true, then x is hypothesized to belong to the concept being learned, the *target*. Hypothesis, h_1, is more general than or equal to h_2, if h_1 covers at least as many examples as h_2 (Mitchell, 1997). That is, $h_1 \geq h_2$ if and only if

$$(\forall x)[h_1(x) \rightarrow h_2(x)]$$

A hypothesis, h_1, is strictly more general than h_2, if $h_1 \geq h_2$ and $h_2 \not\leq h_1$.

Note that the *more general than* ordering is strongly related to *subsumption*.

Cross References

►Classification
►Specialization
►Subsumption
►Logic of Generality

Recommended Reading

Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.

Generalization Bounds

Mark Reid
The Australian National University, Canberra, Australia

Synonyms

Inequalities; Sample complexity

Definition

In the theory of statistical machine learning, a generalization bound – or, more precisely, a generalization error bound – is a statement about the predictive performance of a learning algorithm or class of algorithms. Here, a learning algorithm is viewed as a procedure that takes some finite training sample of labeled instances as input and returns a hypothesis regarding the labels of all instances, including those which may not have appeared in the training sample. Assuming labeled instances are drawn from some fixed distribution, the quality of a hypothesis can be measured in terms of its risk – its incompatibility with the distribution. The performance of a learning algorithm can then be expressed in terms of the expected risk of its hypotheses given randomly generated training samples.

Under these assumptions, a generalization bound is a theorem, which holds for any distribution and states that, with high probability, applying the learning algorithm to a randomly drawn sample will result in a hypothesis with risk no greater than some value. This bounding value typically depends on the size of the training sample, an empirical assessment of the risk of the hypothesis on the training sample as well as the "richness" or "capacity" of the class of predictors that can be output by the learning algorithm.

Motivation and Background

Suppose we have built an e-mail classifier and then collected a random sample of e-mail labeled as "spam" or "not spam" to test it on. We notice that the classifier incorrectly labels 5% of the sample. What can be said about the accuracy of this classifier when it is applied to new, previously unseen e-mail? If we make the reasonable assumption that the mistakes made on future

e-mails are independent of mistakes made on the sample, basic results from statistics tell us that the classifier's true error rate will also be around 5%.

Now suppose that instead of building a classifier by hand we use a learning algorithm to *infer* one from the sample. What can be said about the future error rate of the inferred classifier if it also misclassifies 5% of the training sample? In general, the answer is "nothing" since we can no longer assume future mistakes are independent of those made on the training sample. As an extreme case, consider a learning algorithm that outputs a classifier that just "memorizes" the training sample – predicts labels for e-mail in the sample according to what appears in the sample – and predicts randomly otherwise. Such a classifier will have a 0% error rate on the sample, however, if most future e-mail does not appear in the training sample the classifier will have a true error rate around 50%.

To avoid the problem of memorizing or over-fitting the training data it is necessary to restrict the "flexibility" of the hypotheses a learning algorithm can output. Doing so forces predictions made off the training set to be related to those made on the training set so that some form of generalization takes place. However, doing this can limit the ability of the learning algorithm to output a hypothesis with small risk. Thus, there is a classic bias and variance trade-off: the bias being the limits placed on how flexible the hypotheses can be versus the variance between the training and the true error rates (see ►bias variance decomposition).

By quantifying the notion of hypothesis flexibility in various ways, generalization bounds provide inequalities that show how the flexibility and empirical error rate can be traded off to control the true error rate. Importantly, these statements are typically probabilistic but distribution-independent—they hold for nearly all sets of training data drawn from a fixed but unknown distribution. When such a bound holds for a learning algorithm it means that, unless the choice of training sample was very unlucky, we can be confident that some form of generalization will take place. The first results of this kind were established by Vapnik and Chervonenkis (1971) about 40 years ago and the measure of hypothesis flexibility they introduced – the ►VC dimension (see below) – now bears their initials. A similar style of results were obtained independently by Valiant in 1984 in the Probably Approximately Correct, or ►PAC learning framework (Valiant, 1984). These two lines of work were drawn together by Blumer et al. (1989) and now form the basis of what is known today as statistical learning theory.

Details

For simplicity, we restrict our attention to generalization bounds for binary ►classification problems such as the spam classification example above. In this setting *instances* (e.g., e-mail) from a set $\mathcal{X}$ are associated with *labels* from a set $\mathcal{Y} = \{-1, 1\}$ (e.g., indicating not spam/spam) and an *example* $z = (x, y)$ is a labeled instance from $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$. The association of instances to labels is assumed to be governed by some unknown distribution P over $\mathcal{Z}$.

A *hypothesis* h is a function that assigns labels $h(x) \in \mathcal{Y}$ to instances. The quality of a hypothesis is assessed via a *loss* function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$, which assigns penalty $\ell(y, y')$ when h predicts the label $y' = h(x)$ for the example (x, y). For convenience, we will often combine the loss and hypothesis evaluation on an example $z = (x, y)$ by defining $\ell_h(z) = \ell(y, h(x))$. When examples are sampled from P the expected penalty, or *risk*

$$L_P(h) := \mathbb{E}_P[\ell_h(z)]$$

can be interpreted as a measure of how well h models the distribution P. A loss that is prevalent in classification is the *0–1 loss* $\ell^{0-1}(y, y') = [\![y \neq y']\!]$ where $[\![p]\!]$ is the indicator function for the predicate p. This loss simply assigns a penalty of 1 for an incorrect prediction and 0 otherwise. The associated 0–1 risk for h is the probability the prediction $h(x)$ disagrees with a randomly drawn sample (x, y) from P. Unless stated otherwise, the bounds discussed below are for the 0–1 loss only but, with care, can usually be made to hold with more general losses also.

Once a loss is specified, the goal of a learning algorithm is to produce a low-risk hypothesis based on a finite number of examples. Formally, a *learning algorithm* $\mathcal{A}$ is a procedure that takes a *training sample* $\mathbf{z} = (z_1, \ldots, z_n) \in \mathcal{Z}^n$ as input and returns a hypothesis $h = \mathcal{A}(\mathbf{z})$ with an associated *empirical risk*

$$\hat{L}_{\mathbf{z}}(h) := \frac{1}{n} \sum_{i=1}^{n} \ell_h(z_i).$$

In order to relate the empirical and true risks, a common assumption made in statistical learning theory is that the examples are drawn independently from P. In this case, a sample $\mathbf{z} = (z_1, \ldots, z_n)$ is a random variable from the product distribution P^n over $\mathcal{Z}^n$. Since the sample can be of arbitrary but finite size a learning algorithm can be viewed as a function $\mathcal{A} : \bigcup_{n=1}^{\infty} \mathcal{Z}^n \to \mathcal{H}$ where $\mathcal{H}$ is the algorithm's ▶hypothesis space.

A generalization bound typically comprises several quantities: an empirical estimate of a hypothesis's performance $\hat{L}_{\mathbf{z}}(h)$; the actual (and unknown) risk of the hypothesis $L_P(h)$; a confidence term $\delta \in [0,1]$; and some measure of the flexibility or *complexity* C of the hypotheses that can be output by learning algorithm. The majority of the bounds found in the literature fit the following template.

- *A generic generalization bound*: Let $\mathcal{A}$ be a learning algorithm, P some unknown distribution over $\mathcal{X} \times \mathcal{Y}$, and $\delta > 0$. Then, with probability at least $1-\delta$ over randomly drawn samples $\mathbf{z}$ from P^n, the hypothesis $h = \mathcal{A}(\mathbf{z})$ has risk $L_P(h)$ no greater than $\hat{L}_{\mathbf{z}}(h) + \epsilon(\delta, C)$.

Of course, there are many variations, refinements, and improvements of the bounds presented below and not all fit this template. The bounds discussed below are only intended to provide a survey of some of the key ideas and main results.

Basic bounds: The penalties $\ell_h(z_i) := \ell(y_i, h(x_i))$ made by a fixed hypothesis h on a sample $\mathbf{z} = (z_1, \ldots, z_n)$ drawn from P^n are independent random variables. The law of large numbers guarantees (under some mild conditions) that their mean $\hat{L}_{\mathbf{z}}(h) = \frac{1}{n}\sum_{i=1}^{n} \ell_h(z_i)$ converges to the true risk $L_P(h) = \mathbb{E}_P[\ell_h(z)]$ for h as the sample size increases and several inequalities from probability theory can be used to quantify this convergence. A key result is ▶McDiarmid's inequality, which can be used to bound the deviation of a function of independent random variables from its mean. Since the 0–1 loss takes values in $[0,1]$, applying this result to the random variables $\ell_h(Z_i)$ gives

$$P^n\left(L_P(h) > \hat{L}_{\mathbf{z}}(h) + \epsilon\right) \leq \exp\left(-2n\epsilon^2\right). \tag{1}$$

We can invert this and obtain an upper bound for the true risk that will hold on a given proportion of samples. That is, if we want $L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + \epsilon$ to hold on at least $1-\delta$ of the time on randomly drawn samples we can solve $\delta = \exp(-2n\epsilon^2)$ for ϵ and obtain $\epsilon = \sqrt{\frac{\ln \frac{1}{\delta}}{2n}}$ so that

$$P^n\left(L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln \frac{1}{\delta}}{2n}}\right) \geq 1 - \delta. \tag{2}$$

This simple bound lays the foundation for many of the subsequent bounds discussed below and is the reason for the ubiquity of the $\sqrt{\frac{\ln \frac{1}{\delta}}{n}}$-like terms.

A crucial observation to make about the above bound is that while it holds for any hypothesis h it does *not* hold for all $h \in \mathcal{H}$ *simultaneously*. That is, the samples for which the bounds hold for h_1 may be completely different to those which make the bound hold for h_2. Since a generalization bound must hold for all possible hypotheses output by a learning algorithm we need to extend the above analysis by exploiting additional properties of the hypothesis space $\mathcal{H}$.

In the simple case when there are only finitely many hypothesis, we use the *union bound*. This states that for any distribution P and any finite or countably infinite sequence of events $A_1, A_2 \ldots$ we have $P\left(\bigcup_i A_i\right) \leq \sum_i P(A_i)$. For $\mathcal{H} = \{h_1, \ldots, h_m\}$ we consider the events $Z_h = \{\mathbf{z} \in \mathcal{Z}^n : L_P(h) > \hat{L}_{\mathbf{z}}(h) + \epsilon\}$ when samples of size n give empirical risks for h that are least ϵ smaller than its true risk. Using the union bound and (1) on these events gives

$$P^n\left(\bigcup_{h \in \mathcal{H}} Z_h(n,\epsilon)\right) \leq \sum_{i=1}^{m} P^n\left(Z_h(n,\epsilon)\right) = m \cdot \exp(-2n\epsilon^2).$$

This is a bound on the probability of drawing a training sample from P^n such that *every* hypothesis has a true risk that is ϵ larger than its empirical risk. Inverting this inequality by setting $\delta = m\exp(-2n\epsilon^2)$ yields the following bound.

- *Finite class bound*: Suppose $\mathcal{A}$ has finite hypothesis class $\mathcal{H} = \{h_1, \ldots, h_m\}$. Then with probability at least $1-\delta$ over draws of $\mathbf{z}$ from P^n the hypothesis $h = \mathcal{A}(\mathbf{z})$ satisfies

$$L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln|\mathcal{H}| + \ln \frac{1}{\delta}}{2n}}. \tag{3}$$

It is instructive to compare this to the single hypothesis bound in (2) and note the bound is weakened by the additional term $\ln|\mathcal{H}|$.

Since the union bound also holds for countable sets of events this style of bound can be extended from finite hypothesis classes to countable ones. To do this requires a slight modification of the above argument and the introduction of a distribution π over a countable hypothesis space $\mathcal{H} = \{h_1, h_2, \ldots\}$, which is chosen before any samples are seen. This distribution can be interpreted as a prior belief or preference over the hypotheses in $\mathcal{H}$. Letting $\delta(h) = \delta \cdot \pi(h)$ in the bound (2) implies that for each $h \in \mathcal{H}$ we have

$$P^n\left(L_P(h) > \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln \frac{1}{\delta.\pi(h)}}{2n}}\right) < \delta \cdot \pi(h).$$

Thus, applying the countable union bound to the union of these events over all of $\mathcal{H}$, and noting that $\sum_{h\in\mathcal{H}} \delta \cdot \pi(h) = \delta$ since π is a distribution over $\mathcal{H}$, gives use the following bound:

- *Countable class bound*: Suppose μ is a probability distribution over a finite or countably infinite hypothesis space $\mathcal{H}$. Then with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n the following bound holds for all $h \in \mathcal{H}$

$$L_P(h) \le \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln \frac{1}{\pi(h)} + \ln \frac{1}{\delta}}{2n}}. \tag{4}$$

Although the finite and countable class bounds are proved using very similar techniques (indeed, the former can be derived from the latter by choosing $\pi(h) = \frac{1}{|\mathcal{H}|}$), they differ in the type of penalty they introduce for simultaneously bounding all the hypotheses in $\mathcal{H}$. In (3), the penalty $\ln |\mathcal{H}|$ is purely a function of the size of the class whereas in (4) the penalty $\ln \frac{1}{\pi(h)}$ varies with h. These two different styles of bound can be seen as templates for the two main classes of bounds discussed below: the hypothesis-independent bounds of the next section and the hypothesis-dependent bounds in the section on PAC-Bayesian bounds. The main conceptual leap from here is the extension of the arguments above to non-countable hypothesis classes.

Class complexity bounds: A key result in extending the notion of size or complexity in the above bounds to more general classes of hypotheses is the *symmetrization lemma*. Intuitively, it is based on the observation that if the empirical risks for different samples are frequently near the true risk then they will also be near each other. Formally, it states that for any $\epsilon > 0$ such that $n\epsilon^2 \ge 2$ we have

$$P^n\left(\sup_{h\in\mathcal{H}} \left|L_P(h) - \hat{L}_{\mathbf{z}}(h)\right| > \epsilon\right)$$

$$\le 2P^{2n}\left(\sup_{h\in\mathcal{H}} \left|\hat{L}_{\mathbf{z}'}(h) - \hat{L}_{\mathbf{z}}(h)\right| > \frac{\epsilon}{2}\right). \tag{5}$$

Thus, to obtain a bound on the difference between empirical and true risk it suffices to bound the difference in empirical risks on two independent samples $\mathbf{z}$ and $\mathbf{z}'$, both drawn from P^n. This is useful since the maximum difference $\sup_{h\in\mathcal{H}} \left|\hat{L}_{\mathbf{z}'}(h) - \hat{L}_{\mathbf{z}}(h)\right|$ is much easier to handle than the difference involving $L_P(h)$ as the former term only evaluates losses on the points in $\mathbf{z}$ and $\mathbf{z}'$ while the latter takes into account the entire space $\mathcal{Z}$.

To study these restricted evaluations, we define the restriction of a function class $\mathcal{F}$ to the sample $\mathbf{z}$ by $\mathcal{F}_{\mathbf{z}} = \{(f(z_1), \ldots, f(z_n)) : f \in \mathcal{F}\}$. Since the empirical risk $\hat{L}_{\mathbf{z}}(h) = \frac{1}{n}\sum_{i=1}^n \ell_h(z_i)$ only depends on the values of the loss functions ℓ_h on samples from $\mathbf{z}$ we define the *loss class* $\mathbb{L} = \ell_{\mathcal{H}} = \{\ell_h : h \in \mathcal{H}\}$ and consider its restriction $\mathbb{L}_{\mathbf{z}}$ as well as the restriction $\mathcal{H}_{\mathbf{z}}$ of the hypothesis class it is built upon. As we will see, the measures of complexity of these two classes are closely related.

One such complexity measure is arrived at by examining the size of a restricted function class $\mathcal{F}_{\mathbf{z}}$ as the size of the sample $\mathbf{z}$ increases. The *growth function* or ▶shattering coefficient for the function class $\mathcal{F}$ is defined as the maximum number of distinct values the vectors in $\mathcal{F}_{\mathbf{z}}$ can take given a sample of size n: $S_n(\mathcal{F}) = \sup_{\mathbf{z}\in\mathcal{Z}^n} |\mathcal{F}_{\mathbf{z}}|$. In the case of binary classification with a 0–1 loss, it is not hard to see that the growth functions for both $\mathbb{L}$ and $\mathcal{H}$ are equal, that is, $S_n(\mathbb{L}) = S_n(\mathcal{H})$, and so they can be used interchangeably. Applying a union bound argument to (1) as in the previous bounds guarantees that $P^n\left(\sup_{h\in\mathcal{H}} \left|L_P(h) - \hat{L}_{\mathbf{z}}(h)\right| > \epsilon\right) \le 2S_n(\mathcal{H}) \exp(-n\epsilon^2/8)$ and by inversion we obtain the following generalization bound for arbitrary hypothesis classes $\mathcal{H}$:

- *Growth function bound*: For all $\delta > 0$, a draw of $\mathbf{z}$ from P^n will, with probability at least $1 - \delta$, satisfy for all $h \in \mathcal{H}$

$$L_P(h) \le \hat{L}_{\mathbf{z}}(h) + 2\sqrt{\frac{2\ln S_n(\mathcal{H}) + 2\ln \frac{2}{\delta}}{n}}. \tag{6}$$

One conclusion that can be immediately drawn from this bound is that the shattering coefficient must grow

sub-exponentially for the bound to provide any meaningful guarantee. If the class $\mathcal{H}$ is so rich that hypotheses from it can fit all 2^n possible label combinations – if $S_n(\mathcal{H}) = 2^n$ for all n – then the term $\sqrt{2 \ln S_n(\mathcal{H})/n} > 1$ and so (6) just states $L_P(h) \leq 1$. Therefore, to get nontrivial bounds from (6) there needs to exist some value d for which $S_n(\mathcal{H}) < 2^n$ whenever $n > d$.

VC dimension: This desired property of the growth function is exactly what is captured by the ▶VC dimension $VC(\mathcal{H})$ of a hypothesis class $\mathcal{H}$. Formally, it is defined as $VC(\mathcal{H}) = \max\{n \in \mathbb{N} : S_n(\mathcal{H}) = 2^n\}$ and is infinite if no finite maximum exists. Whether or not the VC dimension is finite plays a central role in the consistency of empirical risk minimization techniques. Indeed, it is possible to show that using ERM on a hypothesis class $\mathcal{H}$ is consistent if and only if $VC(\mathcal{H}) < \infty$. This is partly due to *Sauer's lemma*, which shows that when a hypothesis class $\mathcal{H}$ has finite VC dimension $VC(\mathcal{H}) = d_{\mathcal{H}} < \infty$ its growth function is eventually polynomial in the sample size. Specifically, for all $n \geq d_{\mathcal{H}}$ the growth function satisfies $S_n(\mathcal{H}) \leq \left(\frac{en}{d_{\mathcal{H}}}\right)^{d_{\mathcal{H}}}$. By substituting this result into the Growth Function Bound (6) we obtain the following bound, which shows how the VC dimension plays a role that is analogous to the size a hypothesis class in the finite case.

- *VC dimension bound*: Suppose $\mathcal{A}$ has hypothesis class $\mathcal{H}$ with finite VC dimension $d_{\mathcal{H}}$. Then with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n the hypothesis $h = \mathcal{A}(\mathbf{z})$ satisfies

$$L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + 2\sqrt{\frac{2d_{\mathcal{H}} \ln\left(\frac{2en}{d_{\mathcal{H}}}\right) + 2\ln\frac{2}{\delta}}{n}}. \quad (7)$$

There are many other bounds in the literature that are based on the VC dimension. See the Recommended Reading for pointers to these.

Rademacher averages: ▶Rademacher averages are a second kind of measure of complexity for uncountable function classes and can be used to derive more refined bounds than those above. These averages arise naturally by treating as a random variable the sample-dependent quantity $M_{\mathcal{F}}(\mathbf{z}) = \sup_{f \in \mathcal{F}} [\mathbb{E}_P[f] - \mathbb{E}_{\mathbf{z}}[f]]$. This is just the largest difference taken over all $f \in \mathcal{F}$ between its true mean $\mathbb{E}_P[f]$ and its empirical mean $\mathbb{E}_{\mathbf{z}}[f] := \frac{1}{|\mathbf{z}|}\sum_{i=1}^{|\mathbf{z}|} f(z_i)$. For a loss class $\mathbb{L} = \ell_{\mathcal{H}}$ a bound on this maximum difference – $M_{\mathbb{L}}(\mathbf{z}) \leq B$ – immediately gives a generalization bound of the form $L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + B$. Since $M_{\mathcal{F}}(\mathbf{z})$ is a random variable, McDiarmid's inequality can be used to bound its value in terms of its expected value plus the usual $\sqrt{\frac{\ln\frac{1}{\delta}}{2n}}$ term. Applying symmetrization it can then be shown that this expected value satisfies

$$\mathbb{E}_{P^n}[M_{\mathcal{F}}(\mathbf{z})] \leq \mathbb{E}\left[\sup_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \rho_i \left(f(z_i') - f(z_i)\right)\right] \leq 2R_n(\mathcal{F})$$

where the right-hand expectation is taken over two independent samples $\mathbf{z}, \mathbf{z}' \sim P^n$ and the *Rademacher variables* $\rho_1, \ldots, \rho_n$. These are independent random variables, each with equal probability of taking the values -1 or 1, that give their name to the *Rademacher average*

$$R_n(\mathcal{F}) = \mathbb{E}\left[\sup_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \rho_i f(z_i)\right].$$

Intuitively, this quantity measures how well the functions in $\mathcal{F}$ can be chosen to align with randomly chosen labels ρ_i. The Rademacher averages for the loss class $\mathbb{L}$ and the hypothesis class $\mathcal{H}$ are closely related. For 0–1 loss, it can be shown they satisfy $R_n(\mathbb{L}) = \frac{1}{2}R_n(\mathcal{H})$.

Putting all the above steps together gives the following bounds.

- *Rademacher bound*: Suppose $\mathcal{A}$ has hypothesis class $\mathcal{H}$. Then with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n the hypothesis $h = \mathcal{A}(\mathbf{Z})$ satisfies

$$L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + R_n(\mathcal{H}) + \sqrt{\frac{\ln\frac{1}{\delta}}{2n}}. \quad (8)$$

This bound is qualitatively different to the Growth Function and VC bounds above as the Rademacher average term is *distribution-dependent* whereas the other complexity terms are purely a function of the hypothesis space. Indeed, it is possible to bound the Rademacher average in terms of the VC dimension and obtain the VC bound (7) from (8). Furthermore, the Rademacher average is closely related to the minimum empirical risk via $R_n(\mathcal{H}) = 1 - 2\mathbb{E}[\inf_{h \in \mathcal{H}} \hat{L}_{\mathbf{x},\boldsymbol{\rho}}(h)]$ where $\hat{L}_{\mathbf{x},\boldsymbol{\rho}}(h)$ is the empirical risk of h for the randomly labeled sample $\mathbf{z} = ((x_1, \rho_1), \ldots, (x_n, \rho_n))$. Thus, in

principle, $R_n(\mathcal{H})$ could be estimated for a given learning problem using standard ERM methods.

The Rademacher bound can be further refined so that the complexity term is *data-dependent* rather than distribution-dependent. This is done by noting that the Rademacher average $R_n(\mathcal{F}) = \mathbb{E}\left[\hat{R}_{\mathbf{z}}(\mathcal{F})\right]$ where $\hat{R}_{\mathbf{z}}(\mathcal{F})$ is the *empirical Rademacher average* for $\mathcal{F}$ conditioned on the sample $\mathbf{z}$. Applying McDiarmid's inequality to the difference between $\hat{R}_{\mathbf{z}}(\mathcal{F})$ and its mean gives a sample-dependent bound:

- *Empirical Rademacher bound*: Under the same conditions as the Rademacher bound, the following holds with probability $1-\delta$:

$$L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + \hat{R}_{\mathbf{z}}(\mathcal{H}) + 3\sqrt{\frac{\ln\frac{2}{\delta}}{2n}}. \tag{9}$$

PAC-Bayesian bounds: All the bounds in the previous section provide bounds on deterministic hypotheses, which include complexity terms that are functions of the entire hypothesis space. PAC-Bayesian bounds differ from these in two ways: they provide bounds on nondeterministic hypotheses – labels may be predicted for instances stochastically; and their complexity terms are *hypothesis-dependent*. The term "Bayesian" given to these bounds refers to the use of a distribution over hypotheses that is used to define the complexity term. This distribution can be interpreted as a prior belief over the efficacy of each hypothesis before any observations are made.

Nondeterministic hypotheses are modeled by assuming that a distribution μ over $\mathcal{H}$ is used to randomly draw a deterministic hypothesis $h \in \mathcal{H}$ to predict $h(x)$ each time a new instance x is seen. Such a strategy is called a *Gibbs hypothesis* for μ. Since its behavior is defined by the distribution μ, we will abuse our notation slightly and define its loss on the example z to be $\ell_\mu(z) := \mathbb{E}_{h\sim\mu}[\ell_h(z)]$. Similarly, the true risk and empirical risk for a Gibbs hypothesis are, respectively, defined to be $L_P(\mu) := \mathbb{E}_{h\sim\mu}[L_P(h)]$ and $\hat{L}_{\mathbf{z}}(\mu) := \mathbb{E}_{h\sim\mu}\left[\hat{L}_{\mathbf{z}}(h)\right]$. As with the earlier generalization bounds, the aim is to provide guarantees about the difference between $L_P(\mu)$ and $\hat{L}_{\mathbf{z}}(\mu)$. In the case of 0–1 loss, $p := L_P(\mu) \in [0,1]$ is just the probability of the Gibbs hypothesis for μ misclassifying an example and $q := \hat{L}_{\mathbf{z}}(\mu) \in [0,1]$ can be thought of as an estimate of p. However, unlike the earlier bounds on the difference between the true and estimated risk, PAC-Bayesian bounds are expressed in terms the *Kullback–Leibler (KL) divergence*. For the values $p, q \in [0,1]$ this is defined as $kl(q\|p) := q\ln\frac{q}{p} + (1-q)\ln\frac{1-q}{1-p}$ and for distributions μ and π over the hypothesis space $\mathcal{H}$ we write $KL(\mu\|\pi) := \int_{\mathcal{H}} \ln\frac{d\mu}{d\pi}\, d\mu$. Using these definitions, the most common PAC-Bayesian bound states the following.

- *Theorem (PAC-Bayesian bound)*: For all choices of the distribution π over $\mathcal{H}$ made prior to seeing any examples, the Gibbs hypothesis defined by μ satisfies

$$kl(L_P(\mu), \hat{L}_{\mathbf{z}}(\mu)) \leq \frac{KL(\mu\|\pi) + \ln\frac{n+1}{\delta}}{n} \tag{10}$$

- with probability at least $1-\delta$ over draws of $\mathbf{z}$ from P^n.

This says that the difference (as measured by kl) between the true and empirical risk for the Gibbs hypothesis based on μ is controlled by two terms: a *complexity* term $\frac{KL(\mu\|\pi)}{n}$ and a *sampling* term $\frac{\ln\frac{n+1}{\delta}}{n}$, both of which converge to zero as n increases. To make connections with the previous bounds more apparent, we can weaken (10) using the inequality $kl(q\|p) \geq 2(p-q)^2$ to get the following bound that holds under the same assumptions:

$$L_P(\mu) \leq \hat{L}_{\mathbf{z}}(\mu) + \sqrt{\frac{KL(\mu\|\pi) + \ln\frac{n+1}{\delta}}{2n}}.$$

The sampling term is similar to the ubiquitous estimation penalty in the earlier bounds but with an additional $\ln(n+1)/n$. The complexity term is a measure of the complexity of the Gibbs hypothesis for μ *relative* to the distribution π. Intuitively, $KL(\cdot\|\pi)$ can be thought of as a parametrized family of complexity measures where hypotheses from a region where π is large are "cheap" and those where π is small are "expensive". Information theoretically, it is the expected number of extra bits required to code hypotheses drawn from μ using a code based on π instead of a code based on μ. It is for these reasons the PAC-Bayes bound is said to demonstrate the importance of choosing a good prior. If the Gibbs hypothesis μ, which minimizes $\hat{L}_{\mathbf{z}}(\mu)$ is also "close" to π then the bound will be tight.

Unlike the other bounds discussed above, PAC-Bayesian bounds are in terms of the complexity of single meta-classifiers rather than the complexity of classes. Furthermore, for specific base hypothesis classes such as margin classifiers used by SVMs it is possible to get hypothesis-specific bounds via the PAC-Bayesian

bounds. These are typically much tighter than the VC or Rademacher bounds.

Other bounds: While the above bounds are landmarks in statistical learning theory there is obviously much more territory that has not been covered here. For starters, the VC bounds for classification can be refined by using more sophisticated results from empirical process theory such as the Bernstein and Variance-based bounds. These are discussed in Sect. 5 of (Boucheron et al., 2005). There are also other distribution- and sample-dependent complexity measures that are motivated differently to Rademacher averages. For example, the *VC entropy* (see Sect. 4.5 of (Bousquet et al., 2004)) is a distribution-dependent measure obtained by averaging $|\mathcal{F}_{\mathbf{z}}|$ with respect to the sample distribution rather than taking supremum in the definition of the shattering coefficient.

Moving beyond classification, bounds for regression problems have been studied in depth and have similar properties to those for classification. These bounds are obtained by essentially discretizing the function spaces. The growth function is replaced by what is known as a *covering number* but the essence of the bounds remain the same. The reader is referred to (Herbrich and Williamson, 2002) for a brief discussion and (Anthony and Bartlett, 1999) for more detail.

There are a variety of bounds that, unlike those above, are algorithm-specific. For example, the regularized empirical risk minimization performed by SVMs has been analyzed within an *algorithmic stability* framework. As discussed in Boucheron et al. (2005) and Herbrich and Williamson (2002), hypotheses are considered stable if their predictions are not varied too much when a single training example is perturbed. Two other algorithm-dependent frameworks include the *luckiness* and *compression* frameworks, both summarized in Herbrich and Williamson (2002). The former gives bounds in terms of an *a priori* measure of luckiness – how well a training sample aligns with biases encoded in an algorithm – while the latter considers algorithms, like SVMs, which base hypotheses on key examples within a training sample.

Recently, there has been work on a type of algorithm-dependent, relative bound called *reductions* (see Beygelzimer et al., 2008 for an overview). By transforming inputs and outputs for one type of problem (e.g., probability estimation) into a different type of problem (e.g., classification), bounds for the former can be given in terms of bounds for the latter while making very few assumptions. This opens up a variety of avenues for applying existing results to new learning tasks.

Cross References

- ▶Classification
- ▶Empirical Risk Minimization
- ▶Hypothesis Space
- ▶Loss
- ▶PAC Learning
- ▶Regression
- ▶Regularization
- ▶Structural Risk Minimization
- ▶VC Dimension

Recommended Readings

As mentioned above, the uniform convergence bounds by Vapnik and Chervonenkis (1971) and the PAC framework of Valiant (1984) were the first generalization bounds for statistical learning. Ideas from both were synthesized and extended by Blumer et al. (1989). The book by Kearns and Vazirani (1994) provides a good overview of the early PAC-style bounds while Vapnik's comprehensive book (Vapnik, 1998), and Antony and Bartlett's book (1999) cover classification and regression bounds involving the VC dimension. Rademacher averages were first considered as an alternative to VC dimension in the context of learning theory by Koltchinskii and Panchenko (2001) and were refined and extended by Bartlett and Mendelson (2003) who provide a readable overview. Early PAC-Bayesian bounds were established by McAllester (1999) based on an earlier PAC analysis of Bayesian estimators by Shawe-Taylor and Williamson (1997). Applications of the PAC-Bayesian bound to SVMs are discussed in Langford's tutorial on prediction theory (Langford, 2005) and recent paper by Banerjee (2006) provides an information theoretic motivation, a simple proof of the bound in (10), as well as connections with similar bounds in online learning.

There are several well-written surveys of generalization bounds and learning theory in general. Herbrich and Williamson (2002) present a unified view of VC, compression, luckiness, PAC-Bayesian, and stability bounds. In a very readable introduction to statistical learning theory, Bousquet et al. (2004) provide good intuition and concise proofs for all but the PAC-Bayesian bounds presented above. That introduction is a good companion for the excellent but more technical survey by Boucheron et al. (2005) based on tools from the theory of empirical processes. The latter paper also provides a wealth of further references and a concise history of the development of main techniques in statistical learning theory.

Anthony, M., & Bartlett, P. L. (1999). *Neural network learning: Theoretical foundations*. Cambridge: Cambridge University Press.

Banerjee, A. (2006). On Bayesian bounds. *ICML '06: Proceedings of the 23rd International Conference on Machine learning*, Pittsburgh, pp. 81–88.

Bartlett, P. L., & Mendelson, S. (2003). Rademacher and Gaussian complexities: risk bounds and structural results. *Journal of Machine Learning Research, 3*, 463–482.
Beygelzimer, A., Langford, J., & Zadrozny, B. (2008). Machine learning techniques – reductions between prediction quality metrics. In Liu, Zhen; Xia, Cathy H. (Eds.) *Performance modeling and engineering* (pp. 3–28). Springer.
Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM), 36*(4), 929–965.
Boucheron, S., Bousquet, O., & Lugosi, G. (2005). Theory of classification: A survey of some recent advances. *ESAIM Probability and statistics, 9*, 323–375.
Bousquet, O., Boucheron, S., & Lugosi, G. (2004). *Introduction to statistical learning theory, volume 3176 of lecture notes in artificial intelligence* (pp. 169–207). Berlin: Springer.
Herbrich, R., & Williamson, R. C. (2002). Learning and generalization: Theory and bounds. In M. Arbib (Ed.), *Handbook of brain theory and neural networks* (2nd ed.). Cambridge: MIT Press.
Kearns, M. J., & Vazirani, U. V. (1994). *An introduction to computational learning theory*. Cambridge: MIT Press.
Koltchinskii, V. (2001). Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory, 47*(5), 1902–1914.
Langford, J. (2005). Tutorial on practical prediction theory for classification. *Journal of Machine Learning Research, 6*(1), 273–306.
McAllester, D. A. (1999). Some PAC-Bayesian theorems. *Machine Learning, 37*(3), 355–363.
Shawe-Taylor, J., & Williamson, R. C. (1997). A PAC analysis of a Bayesian estimator. *Proceedings of the Tenth Annual Conference on Computational Learning Theory*, ACM, p. 7.
Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM, 27*(11), 1142.
Vapnik, V. N. (1998). *Statistical learning theory*. New York: Wiley.
Vapnik, V. N., & Chervonenkis, A. Y., (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications, 16*(2), 264–280.

Generalization Performance

The *generalization performance* of a learning algorithm refers to the performance on ►out-of-sample data of the ►models learned by the algorithm.

Cross References

►Algorithm Evaluation

Generalized Delta Rule

►Backpropagation

General-to-Specific Search

When searching a hypothesis space, a general-to-specific search starts from the most general hypothesis and expands the search by specialization. See ►Learning as Search.

Generative and Discriminative Learning

Bin Liu, Geoffrey I. Webb
Monash University

Definition

Generative learning refers alternatively to any classification learning process that classifies by using an estimate of the joint probability P(y, **x**) or to any classification learning process that classifies by using estimates of the prior probability P(y) and the conditional probability P(**x** | y) (Bishop, 2007; Jaakkola & Haussler, 1999; Jaakkola, Meila & Jebara, 1999; Lasserre, Bishop & Minka, 2006; Ng & Jordan, 2002), where y is a class and **x** is a description of an object to be classified. Generative learning contrasts with *discriminative learning* in which a model or estimate of P(y | **x**) is formed without reference to an explicit estimate of any of P(y, **x**), P(**x**) or P(**x** | y).

It is also common to categorize as discriminative approaches based on a decision function that directly maps from input **x** onto the output y (such as support vector machines, neural networks, and decision trees), where the decision risk is minimized without estimation of P(y, **x**), P(**x** | y) or P(y | **x**) (Jaakkola & Haussler, 1999).

The standard exemplar of generative learning is naïve Bayes and of discriminative learning, ►logistic regression. Another important contrasting pair is the generative hidden Markov model and discriminative conditional random field.

It is widely accepted that generative learning works well when samples are rare while discriminative learning has better asymptotic error performance (Ng & Jordan, 2002).

Motivation and Background

Efron (1975) provides an early examination of the generative/discriminative distinction. Efron performs an empirical comparison of the efficiency of the generative linear discriminant analysis (LDA) and discriminative logistic regression. His results show that logistic regression has 30% less efficiency than LDA, which means the discriminative approach is 30% slower to reach the asymptotic error than the generative approach.

Ng et al. (2002) give a theoretical discussion of the efficiency of generative naïve Bayes and discriminative logistic regression. Their result shows that logistic regression converges toward its asymptotic error in order n samples while naïve Bayes converges in order $\log n$ samples. While logistic regression converges much slower than naïve Bayes, it has lower asymptotic error than naïve Bayes. These results suggest that it is desirable to use a generative approach when training data is scarce and to use a discriminative approach when there is enough training data.

Recent research into the generative/discriminative learning distinction has concentrated on the area of hybrids of generative and discriminative learning, as well as generative learning and discriminative learning in structured data learning or semi-supervised learning context.

In hybrid approaches, researchers seek to obtain the merits of both generative learning and discriminative learning. Some examples include the Fisher kernel for discriminative learning (Jaakkola & Haussler, 1999), max-ent discriminative learning (Jaakkola, Meila & Jebara, 1999), and principled hybrids of generative and discriminative models (Lasserre, Bishop & Minka, 2006).

In structured data learning, the output data have dependent relationships. As an example of generative learning, the hidden Markov models are used in structured data problems which need sequential decisions. The discriminative analog is the conditional random field models. Another example of discriminatively structured learning is Max-margin Markov networks (Taskar, Guestrin & Koller, 2004).

In semi-supervised learning, co-training and multiview learning are usually applied to generative learning (Blum & Mitchell, 1998). It is less straightforward to apply semi-supervised learning in traditional discriminative learning, since $P(y|\ \mathbf{x})$ is estimated by ignoring $P(\mathbf{x})$. Examples of semi-supervised learning methods in discriminative learning include transductive SVM, Gaussian processes, information regularization, and graph-based methods (Chapelle, Schölkopf & Zien, 2006).

Cross References

►Evolutionary Feature Selection and Construction

Recommended Reading

Bishop, C. M. (2007). *Pattern recognition and machine learning*. Springer.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. *Proceedings of the eleventh annual conference on Computational learning theory*, Madison, Wisconsin, USA. New York: ACM.

Chapelle, O., Schölkopf, B., & Zien, A. (2006). *Semi-supervised learning*. Cambridge: The MIT Press.

Efron, B. (1975). The efficiency of logistic regression compared to normal discriminant analysis. *Journal of the American Statistical Association, 70*(352), 892–898.

Jaakkola, T. S., & Haussler, D. (1999). Exploiting generative models in discriminative classifiers. *Advances in neural information processing systems, 11*.

Jaakkola, T., Meila, M., & Jebara, T. (1999). Maximum entropy discrimination. *Advances in neural information processing systems, 12*.

Lasserre, J. A., Bishop, C. M., & Minka, T. P. (2006). Principled hybrids of generative and discriminative models. *IEEE Conference on Computer Vision and Pattern Recognition*, New York.

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes. *Advances in neural information processing systems, 14*.

Taskar, B., Guestrin, C., & Koller, D. (2004). Max-margin Markov networks. *Advances in neural information processing systems, 16*.

Generative Learning

Definition

Generative learning refers alternatively to any classification learning process that classifies by using an estimate of the joint probability $P(y, \mathbf{x})$ or to any classification learning process that classifies by using estimates of the prior probability P(y) and the conditional probability

$P(\mathbf{x} \mid y)$, where y is a class and $\mathbf{x}$ is a description of an object to be classified. Given such models or estimates it is possible to generate synthetic objects from the joint distribution. Generative learning contrasts to discriminative learning in which a model or estimate of $P(y \mid \mathbf{x})$ is formed without reference to an explicit estimate of any of $P(\mathbf{x})$, $P(y, \mathbf{x})$, or $P(\mathbf{x} \mid y)$.

Cross References

►Generative and Discriminative Learning

Genetic and Evolutionary Algorithms

Claude Sammut
The University of New South Wales
Sydney, Australia

Definitions

There are many variations of genetic algorithms (GA). Here, we describe a simple scheme to introduce some of the key terms in genetic and evolutionary algorithms. See the main entry on ►Evolutionary Algorithms for references to specific methods.

In genetic learning, we assume that there is a population of individuals, each of which represents a candidate problem solver for a given task. GAs can be thought of as a family of general purpose search methods that are capable of solving a broad range of problems from optimization and scheduling to robot control. Like evolution, genetic algorithms test each individual from the population and only the fittest survive to reproduce for the next generation. The algorithm creates new generations until at least one individual is found that can solve the problem adequately.

Each problem solver is a *chromosome*. A position, or set of positions in a chromosome is called a *gene*. The possible values (from a fixed set of symbols) of a gene are known as *alleles*. For example, a simple genetic algorithm may define the set of symbols to be $\{0, 1\}$, and chromosome lengths are fixed. The most critical problem in applying a genetic algorithm is in finding a suitable encoding of the examples in the problem domain to a chromosome. A good choice of representation will make the search easier by limiting the size of the search space. A poor choice will result in a large search space. Choosing the size of the population can be problematic since a small population size provides an insufficient sample over the space of solutions for a problem and large population requires extensive evaluation and will be slow.

Each iteration in a genetic algorithm is called a *generation*. Each chromosome in a population is used to solve a problem. Its performance is evaluated and the chromosome is given a rating of fitness. The population is also given an overall fitness rating based on the performance of its members. The fitness value indicates how close a chromosome or population is to the required solution.

New sets of chromosomes are produced from one generation to the next. Reproduction takes place when selected chromosomes from one generation are recombined with others to form chromosomes for the next generation. The new ones are called *offspring*. Selection of chromosomes for reproduction is based on their fitness values. The average fitness of the population may also be calculated at the end of each generation. The strategy must be modified if too few or too many chromosomes survive. For example, at least 10% and at most 60% must survive.

Genetic Operators

Operators that recombine the selected chromosomes are called *genetic operators*. Two common operators are *crossover* and *mutation*. Crossover exchanges portions of a pair of chromosomes at a randomly chosen point called the crossover point. Some Implementations have more than one crossover point. For example, if there are two chromosomes, X and Y:

$$X = 1001\ 01011,\ \ Y = 1110\ 10010$$

and the crossover point is after position 4, the resulting offspring are:

$$O1 = 100110010,\ \ O2 = 1110\ 01011$$

Offspring produced by crossover cannot contain information that is not already in the population, so an additional operator, *mutation*, is required. Mutation generates an offspring by randomly changing the values of

genes at one or more gene positions of a selected chromosome. For example, if the following chromosome,

$$Z = 100101011$$

is mutated at positions 2, 4, and 9, then the resulting offspring is:

$$O = 110001010$$

The number of offspring produced for each new generation depends on how members are introduced so as to maintain a fixed population size. In a *pure* replacement strategy, the whole population is replaced by a new one. In an *elitist* strategy, a proportion of the population survives to the next generation.

Cross References

►Evolutionary Algorithms

Genetic Attribute Construction

►Evolutionary Feature Selection and Construction

Genetic Clustering

►Evolutionary Clustering

Genetic Feature Selection

►Evolutionary Feature Selection and Construction

Genetic Grouping

►Evolutionary Clustering

Genetic Neural Networks

►Neuroevolution

Genetic Programming

Moshe Sipper
Ben-Gurion University, Beer-Sheva, Israel

Genetic Programming is a subclass of ►evolutionary algorithms, wherein a population of individual programs is evolved. The main mechanism behind genetic programming is that of a ►generic algorithm, namely, the repeated cycling through four operations applied to the entire population: evaluate–select–crossover–mutate. Starting with an initial population of randomly generated programs, each individual is evaluated in the domain environment and assigned a fitness value representing how well the individual solves the problem at hand. Being randomly generated, the first-generation individuals usually exhibit poor performance. However, some individuals are better than others, that is, as in nature, variability exists, and through the mechanism of selection, these have a higher probability of being selected to parent the next generation. The size of the population is finite and usually constant.

See ►Evolutionary Games for a more detailed explanation of genetic programming.

Genetics-Based Machine Learning

►Classifier Systems

Gibbs Sampling

Gibbs Sampling is a heuristic inference algorithm for ►Bayesian networks. See ►Graphical Models for details.

Gini Coefficient

The Gini coefficient is an empirical measure of classification performance based on the area under an ROC curve (AUC). Attributed to the Italian statistician Corrado Gini (1884-1965), it can be calculated as $2 \cdot \text{AUC} - 1$

and thus takes values in the interval $[-1, 1]$, where 1 indicates perfect ranking performance and -1 indicates that all negatives are ranked before all positives. See ▶ROC Analysis.

Gram Matrix

▶Kernel Matrix

Grammar Learning

▶Grammatical Inference

Grammatical Inference

Lorenza Saitta[1], Michele Sebag[2]
[1]Università del Piemonte Orientale, Alessandria, Italy
[2]CNRS – INRIA – Université Paris-Sud, Orsay, France

Synonyms

Grammatical inference, Grammar learning

Definition

Grammatical inference is concerned with inferring grammars from positive (and possibly negative) examples (Angluin, 1978; Korfiatis & Paliouras, 2008; Sakakibara, 2005). A context-free grammar (CFG) $\mathcal{G}$ (equivalent to a push-down finite-state automaton), is described by a four-tuple $(\mathcal{Q}, \mathcal{E}, \delta, \Sigma)$:

- Σ is the alphabet of *terminal* symbols, upon which the grammar is defined.
- The pair $(\mathcal{Q}, \mathcal{E})$ defines a graph, where $\mathcal{Q}$ is the set of nodes (states), and $\mathcal{E}$ is the set of edges (production rules). $\mathcal{Q}$ includes one *starting* node q_0 and a set $\mathcal{Q}_f$ ($\mathcal{Q}_f \subset \mathcal{Q}$) of final or *accepting* nodes.
- Every edge in $\mathcal{E}$ is labelled by one or several letters in Σ, expressed through mapping $\delta : \mathcal{E} \mapsto 2^{\Sigma}$.
- Let $\mathcal{L}(\mathcal{G})$ denote the language associated to the grammar. Each string s in $\mathcal{L}(\mathcal{G})$ is generated along a random walk in the graph, starting in q_0 with an initially empty s. Upon traversing edge e, one symbol from $\delta(e)$ is concatenated to s. The walk ends upon reaching a final node ($e \in \mathcal{Q}_f$).

A CFG is determinist iff all pairs of edges (q, q') and (q, q'') $(q' \neq q'')$ bear different labels $(\delta(q, q') \cap \delta(q, q'') = \varnothing)$.

One generalizes a given CFG by applying one or several operators, among the following: (1) introducing additional nodes and edges; (2) turning a node into an accepting one; (3) *merging* two nodes q and q'. In the latter case, some non-determinism can be introduced (if some edges (q, r) and (q', r') have label(s) in common); enforcing a deterministic generalization is done using the *recursive determinisation operator* (e.g., merging nodes r and r').

In general, grammatical inference proceeds as follows (Lang, Pearlmutter, & Price, 1998; Oncina & Garcia, 1992). Let S be the set of positive examples, strings on alphabet Σ. The *prefix tree acceptor* (PTA), a most specific generalization of S, is constructed by associating to each character of every string a distinct node, and applying the determinisation operator. This PTA is thereafter iteratively generalized by merging a pair of nodes. Well known grammar learners are Rpni (Oncina & Garcia, 1992) and Blue-Fringe (Lang et al., 1998). Rpni uses a depth first search strategy, and merges the pair of nodes which are closest to the start node, such that their deterministic generalization does not cover any negative example. Blue-Fringe uses a beam search from a candidate list, selecting the pair of nodes to be merged after the *evidence-driven* state merging (EDSM) criterion, i.e., such that their generalization involves a minimal number of final states.

Recommended Reading

Angluin D. (1978). On the complexity of minimum inference of regular sets. *Information and Control, 39*, 337–350.

Korfiatis, G., & Paliouras, G. (2008). Modeling web navogation using grammatical inference. *Applied Artificial Intelligence, 22*(1–2), 116–138.

Lang, K. J., Pearlmutter, B. A., & Price, R. A. (1998). Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *ICGI '98: Proceedings of the 4th international colloquium on grammatical inference* (pp. 1–12). Berlin: Springer.

Oncina, J., & Garcia, P. (1992). Inferring regular languages in polynomial update time. In *Pattern recognition and image analysis*, (Vol. 1, pp. 49–61). World Scientific.

Sakakibara, Y. (2005). Grammatical inference in bioinformatics. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*(7), 1051–1062.

Grammatical Tagging

►POS Tagging

Graph Clustering

Charu C. Aggarwal
IBM T. J. Watson Research Center, Hawthorne, NY, USA

Synonyms

Minimum cuts; Network clustering; Spectral clustering; Structured data clustering

Definition

Graph clustering refers to ►clustering of data in the form of graphs. Two distinct forms of clustering can be performed on graph data. Vertex clustering seeks to cluster the nodes of the graph into groups of densely connected regions based on either edge weights or edge distances. The second form of graph clustering treats the graphs as the objects to be clustered and clusters these objects on the basis of similarity. The second approach is often encountered in the context of structured or XML data.

Motivation and Background

Graph clustering is a form of ►graph mining that is useful in a number of practical applications including marketing, customer segmentation, congestion detection, facility location, and XML data integration (Lee, Hsu, Yang, & Yang, 2002). The graph clustering problems are typically defined into two categories:

- *Node clustering algorithms*: Node clustering algorithms are generalizations of multidimensional clustering algorithms in which we use functions of the multidimensional data points in order to define the distances. In the case of graph clustering algorithms, we associate numerical values with the edges. These numerical values need not satisfy traditional properties of distance functions such as the triangle inequality. We use these distance values in order to create clusters of nodes. We note that the numerical value associated with a given node may either be a distance value or a similarity value. Correspondingly, the objective function associated with the partitioning may either be minimized or maximized. We note that the problem of minimizing the intercluster similarity for a fixed number of clusters essentially reduces to the problem of *graph partitioning* or the *minimum multiway cut problem*. This is also referred to the problem of mining dense graphs and pseudo-cliques. Recently, the problem has also been studied in the database literature as that of *quasi-clique determination*. In this problem, we determine groups of nodes which are "almost cliques." In other words, an edge exists between any pair of nodes in the set with a high probability. A closely related problem is that of determining *shingles* (Gibson, Kumar, & Tomkins, 2005). Shingles are defined as those subgraphs which have a large number of common links. This is particularly useful for massive graphs which contain a large number of nodes. In such cases, a min-hash approach (Gibson et al., 2005) can be used in order to summarize the structural behavior of the underlying graph.
- *Graph clustering algorithms*: In this case, we have a (possibly large) number of graphs which need to be clustered based on their underlying structural behavior. This problem is challenging because of the need to match the structures of the underlying graphs and use these structures for clustering purposes. Such algorithms are discussed both in the context of classical graph data sets as well as semistructured data. In the case of semistructured data, the problem arises in the context of a large number of documents which need to be clustered on the basis of the underlying structure and attributes. It has been shown by Aggarwal, Ta, Feng, Wang, and Zaki (2007) that the use of the underlying document structure leads to significantly more effective algorithms.

This chapter will discuss the different kinds of clustering algorithms and their applications. Each section will discuss a particular class of clustering algorithms and the different approaches which are commonly used for this class.

Graph Clustering as Minimum Cut

The graph clustering problem can be related to the minimum-cut and graph partitioning problems. In this case, it is assumed that the underlying graphs have weights on the edges. It is desired to partition the graphs in such a way so as to minimize the weights of the edges across the partitions. In general, we would like to partition the graphs into k groups of nodes. However, since the special case $k = 2$ is efficiently solvable, we would like to first provide a special discussion for this case. This version is polynomially solvable, since it is the mathematical dual of the maximum-flow problem. This problem is also referred to as the *minimum-cut problem*.

The minimum-cut problem is defined as follows. Consider a graph $G = (N, A)$ with node set N and edge set A. The node set N contains the source s and sink t. Each edge $(i, j) \in A$ has a weight associated with it which is denoted by u_{ij}. We note that the edges may be either undirected or directed, though the undirected case is often much more relevant for connectivity applications. We would like to partition the node set N into two groups S and $N-S$. The set of edges such that one end lies in S and the other lies in $N-S$ is denoted by $C(S, N-S)$. We would like to partition the node set N into two sets S and $N - S$, such that the sum of the weights in $C(S, N-S)$ is minimized. In other words, we would like to minimize $\sum_{(i,j)\in C(S,N-S)} u_{ij}$. This is the unrestricted version of the minimum-cut problem. We will examine two variations of the minimum-cut problem:

- We wish to determine the global minimum s-t cut with no restrictions on the membership of nodes to different partitions.
- We wish to determine the minimum s-t cut, in which one partition contains the source node s and the other partition contains the sink node t.

It is easy to see that the former problem can be solved by using repeated applications of the latter algorithm. By fixing s and choosing different values of the sink t, it can be shown that the global minimum cut may be effectively determined.

It turns out that the maximum-flow problem is the mathematical dual of the minimum-cut problem. In the maximum-flow problem, we assume that the weight u_{ij} is a capacity of the edge (i, j). Each edge is allowed to have a *flow* x_{ij} which is at most equal to the capacity u_{ij}. Each node other than the source s and sink t is assumed to satisfy the *flow conservation property*. In other words, for each node $i \in N$ we have

$$\sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji}.$$

We would like to maximize the total flow originating from the source and reaching the sink t, subject to the above constraints. The maximum-flow problem is solved with the use of a variety of *augmenting path* and *preflow push algorithms*. Details of different kinds of algorithms may be found in the work by Ahuja, Orlin, and Magnanti (1992).

A closely related problem to the minimum s-t cut problem is that of determining a *global minimum cut* in an undirected graph. This particular case is more efficient than that of finding the s-t minimum cut. One way of determining a minimum cut is by using a contraction-based edge-sampling approach. While the previous technique is applicable to both the directed and undirected versions of the problem, the contraction-based approach is applicable only to the undirected version of the problem. Furthermore, the contraction-based approach is applicable only for the case in which the weight of each edge is $u_{ij} = 1$. While the method can easily be extended to the weighted version by varying the edge-sampling probability, the polynomial running time bounds discussed by Tsay, Lovejoy, and Karger (1999) do not apply to this case. The contraction approach is a probabilistic technique in which we successively sample the edges in order to collapse nodes into larger sets of nodes. By successively sampling different sequences of edges and picking the optimum value (Tsay et al., 1999), it is possible to determine a global minimum cut. The broad idea of the contraction-based approach is as follows. We pick an edge randomly in the graph and contract its two end points into a single node. We remove all the self-loops which are created as a result of the contraction. We may also create some parallel edges, which are allowed to remain, since they influence the sampling probability (Alternatively, we may replace parallel edgesby a single edge of weight which is equal to the number of parallel edges. We use this weight in order to bias the sampling process.) of contractions. The process of contraction is repeated until we are left with two nodes. We note that

each of this pair of "super-nodes" corresponds to a set of nodes in the original data. These two sets of nodes provide us with the final minimum cut. We note that the minimum cut will survive in this approach, if none of the edges in the minimum cut are sampled during the contraction. It has been shown by Tsay et al. that by using repeated contraction of the graph to a size of $\sqrt{n}$ nodes, it is possible to obtain a correct solution with high probability in $O(n^2)$ time.

Graph Clustering as Multiway Graph Partitioning

The *multiway graph partitioning problem* is significantly more difficult, and is NP-hard (Kernighan & Lin, 1970). In this case, we wish to partition a graph into $k > 2$ components, so that the total weight of the edges whose ends lie in different partitions is minimized. A well-known technique for graph partitioning is the Kerninghan-Lin algorithm (Kernighan & Lin, 1970). This classical algorithm is based on hill climbing (or more generally neighborhood-search technique) for determining the optimal graph partitioning. Initially, we start off with a random cut of the graph. In each iteration, we exchange a pair of vertices in two partitions to see if the overall cut value is reduced. In the event that the cut value is reduced, then the interchange is performed. Otherwise, we pick another pair of vertices in order to perform the interchange. This process is repeated until we converge to a optimal solution. We note that this optimum may not be a global optimum, but may only be a local optimum of the underlying data. The main variation in different versions of the Kerninghan-Lin algorithm is the policy which is used for performing the interchanges on the vertices. Some examples of strategies which may be used in order to perform the interchange are as follows:

- We randomly pick a pair of vertices and perform the interchange, if it improves the underlying solution quality.
- We test all possible vertex-pair interchanges (or a sample of possible interchanges), and pick the interchange which improves the solution by the greatest amount.
- A k-interchange is one in which a sequence of k interchanges are performed at one time. We can test any k-interchange and perform it, if it improves the underlying solution quality.
- We can pick the optimal k-interchange from a sample of possibilities.

We note that the use of more sophisticated strategies allows a better improvement in the objective function for each interchange, but also requires more time for each interchange. For example, the determination of an optimal k-interchange requires much more time than a straightforward interchange. This is a natural trade-off which may work out differently depending upon the nature of the application at hand. Furthermore, the choice of the policy also affects the likelihood of getting stuck at a local optimum. For example, the use of k-interchange techniques are far less likely to result in local optimum for larger values of k. In fact, by choosing the best interchange across all possible values of k it is possible to ensure that a global optimum is always reached. On the other hand, it is increasingly difficult to implement the algorithm efficiently with increasing value of k. This is because the time complexity of the interchange increases exponentially with the value of k.

Graph Clustering with *k*-Means

Two well-known (and related) techniques for clustering in the context of multidimensional data (Jain & Dubes, 1998) are the k-medoid and k-means algorithms. In the k-medoid algorithm (for multidimensional data), we sample a small number of points from the original data as *seeds* and assign every other data point from the clusters to the closest of these seeds. The closeness may be defined based on a user-defined objective function. The objective function for the clustering is defined as the sum of the corresponding distances of data points to the corresponding seeds. In the next iteration, the algorithm interchanges one of the seeds for another randomly selected seed from the data, and checks if the quality of the objective function improves upon performing the interchange. If this is indeed the case, then the interchange is accepted. Otherwise, we do not accept the interchange and try another sample interchange. This process is repeated, until the objective function does not improve over a predefined number of interchanges. A closely related method is the k-means method. The main difference

with the k-medoid method is that we do not use representative points from the original data after the first iteration of picking the original seeds. In subsequent iterations, we use the centroid of each cluster as the seed set for the next iteration. This process is repeated until the cluster membership stabilizes.

A method has been proposed by Rattigan, Maier, and Jensen (2007), which uses the characteristics of both the k-means and k-medoids algorithms. As in the case of the conventional partitioning algorithms, it picks k graph nodes as seeds. The main differences from the conventional algorithms are in terms of computation of distances (for assignment purposes), and in determination of subsequent seeds. A natural distance function for graphs is the *geodesic distance*, or the smallest number of hops between a pair of nodes. In order to determine the seed set for the next iteration, we compute the *local closeness centrality* for each cluster, and use the corresponding node as the sample seed. Thus, while this algorithm continues to use seeds from the original data set (as in the k-medoids algorithm), it uses intuitive ideas from the k-means algorithms in order to determine the identity of these seeds.

Graph Clustering with the Spectral Method

Eigenvector techniques are often used in multidimensional data in order to determine the underlying correlation structure in the data. It is natural to question as to whether such techniques can also be used for the more general case of graph data. It turns out that this is indeed possible with the use of a method called *spectral clustering*.

In the spectral clustering method, we make use of the node-node adjacency matrix of the graph. For a graph containing n nodes, let us assume that we have an $n \times n$ adjacency matrix, in which the entry (i, j) correspond to the weight of the edge between the nodes i and j. This essentially corresponds to the similarity between nodes i and j. This entry is denoted by w_{ij}, and the corresponding matrix is denoted by W. This matrix is assumed to be symmetric, since we are working with undirected graphs. Therefore, we assume that $w_{ij} = w_{ji}$ for any pair (i, j). All diagonal entries of the matrix W are assumed to be 0. As discussed earlier, the aim of any node partitioning algorithm is to minimize (a function of) the weights across the partitions. The spectral clustering method constructs this minimization function in terms of the matrix structure of the adjacency matrix and another matrix which is referred to as the *degree matrix*.

The *degree matrix* D is simply a diagonal matrix in which all entries are zero except for the diagonal values. The diagonal entry d_{ii} is equal to the sum of the weights of the incident edges. In other words, the entry d_{ij} is defined as follows:

$$d_{ij} = \begin{cases} \sum_{j=1}^{n} w_{ij}, & i = j, \\ 0, & i \neq j. \end{cases}$$

We formally define the *Laplacian matrix* as follows: (*Laplacian matrix*): The Laplacian matrix L is defined by subtracting the weighted adjacency matrix from the degree matrix. In other words, we have

$$L = D - W.$$

This matrix encodes the structural behavior of the graph effectively and its eigenvector behavior can be used in order to determine the important clusters in the underlying graph structure. It can be shown that the Laplacian matrix L is positive semidefinite i.e., for any n-dimensional row vector $f = [f_1 \ldots f_n]$ we have $f \cdot L \cdot f^{\mathrm{T}} \geq 0$. This can be easily shown by expressing L in terms of its constituent entries which are a function of the corresponding weights w_{ij}. Upon expansion, it can be shown that

$$f \cdot L \cdot f^{\mathrm{T}} = (1/2) \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot (f_i - f_j)^2.$$

The Laplacian matrix L is positive semidefinite. Specifically, for any n-dimensional row vector $f = [f_1 \ldots f_n]$, we have

$$f \cdot L \cdot f^{T} = (1/2) \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot (f_i - f_j)^2.$$

At this point, let us examine some *interpretations* of the vector f in terms of the underlying graph partitioning. Let us consider the case in which each f_i is drawn from the set $\{0, 1\}$, and this determines a two-way partition by labeling each node either 0 or 1. The particular partition to which the node i belongs is defined by the corresponding label. Note that the expansion of the

expression $f \cdot L \cdot f^T$ from the above relationship simply represents the sum of the weights of the edges across the partition defined by f. Thus, the determination of an appropriate value of f for which the function $f \cdot L \cdot f^T$ is minimized also provides us with a good node partitioning. Unfortunately, it is not easy to determine the *discrete values* of f which determine this optimum partitioning. Nevertheless, we will see later in this section that even when we restrict f to real values, this provides us with the intuition necessary to create an effective partitioning.

An immediate observation is that the indicator vector $f = [1 \ldots 1]$ is an eigenvector with a corresponding eigenvalue of 0. We note that $f = [1 \ldots 1]$ must be an eigenvector, since L is positive semidefinite and $f \cdot L \cdot f^T$ can be 0 only for eigenvectors with 0 eigenvalues. This observation can be generalized further in order to determine the number of connected components in the graph. We make the following observation.

The number of (*linearly independent*) *eigenvectors with zero eigenvalues for the Laplacian matrix L is equal to the number of connected components in the underlying graph.*

We observe that connected components are the most obvious examples of clusters in the graph. Therefore, the determination of eigenvectors corresponding to zero eigenvalues provides us the information about (relatively rudimentary set of) clusters. Broadly speaking, it may not be possible to glean such clean membership behavior from the other eigenvectors. One of the problems is that other than this particular rudimentary set of eigenvectors (which correspond to the connected components), the vector components of the other eigenvectors are drawn from the real domain rather than the discrete $\{0,1\}$ domain. Nevertheless, because of the nature of the natural interpretation of $f \cdot L \cdot f^T$ in terms of the weights of the edges across nodes with very differing values of f_i, it is natural to cluster together the nodes for which the values of f_i are as similar as possible across any particular eigenvector on an average. This provides us with the intuition necessary to define an effective spectral clustering algorithm, which partitions the data set into k clusters for any arbitrary value of k. The algorithm is as follows:

- Determine the k eigenvectors with the smallest eigenvalues. Note that each eigenvector has as many components as the number of nodes. Let the component of the jth eigenvector for the ith node be denoted by p_{ij}.
- Create a new data set with as many records as the number of nodes. The ith record in this data set corresponds to the ith node and has k components. The record for this node is simply the eigenvector components for that node, which are denoted by $p_{i1} \ldots p_{ik}$.
- Since we would like to cluster nodes with similar eigenvector components, we use any conventional clustering algorithm (e.g., k-means) in order to create k clusters from this data set. Note that the main focus of the approach was to create a *transformation* of a structural clustering algorithm into a more conventional multidimensional clustering algorithm, which is easy to solve. The particular choice of the multidimensional clustering algorithm is orthogonal to the broad spectral approach.

The above algorithm provides a broad framework for the spectral clustering algorithm. The input parameter for the above algorithm is the number of clusters k. In practice, a number of variations are possible in order to tune the quality of the clusters which are found. More details on the different methods which can be used for effective spectral graph clustering may be found in Chung (1997).

Graph Clustering as Quasi-Clique Detection

A different way of determining massive graphs in the underlying data is that of determining *quasi-cliques*. This technique is different from many other partitioning algorithms, in that it focuses on definitions which maximize the edge densities *within a partition*, rather than minimizing the edge densities across partitions. A clique is a graph in which every pair of nodes are connected by an edge. A quasi-clique is a relaxation on this concept, and is defined by imposing a lower bound on the degree of each vertex in the given set of nodes. Specifically, we define a γ-quasi-clique is as follows:

A k-graph ($k \geq 1$) G is a γ-quasi-clique if the degree of each node in the corresponding subgraph of vertices is at least $\gamma \cdot k$.

The value of γ always lies in the range $(0,1]$. We note that by choosing $\gamma = 1$, this definition reverts to that

of standard cliques. Choosing lower values of γ allows for the relaxations which are more true in the case of real applications. This is because we rarely encounter complete cliques in real applications, and at least some edges within a dense subgraph would always be missing. A vertex is said to be critical if its degree in the corresponding subgraph is the smallest integer which is at least equal to $\gamma \cdot k$.

The earliest piece of work on this problem is from Abello, Resende, and Sudarsky (2002). The work of Abello et al. (2002) uses a greedy randomized adaptive search algorithm, GRASP, to find a quasi-clique with the maximum size. A closely related problem is that of finding *frequently occurring cliques* in *multiple data sets*. In other words, when multiple graphs are obtained from different data sets, some dense subgraphs occur frequently together in the different data sets. Such graphs help in determining *important dense patterns of behavior in different data sources*. Such techniques find applicability in mining important patterns in graphical representations of customers. The techniques are also helpful in mining cross-graph quasi-cliques in gene expression data. An efficient algorithm for determining cross graph quasi-cliques was proposed by Pei, Jiang, and Zhang (2005). The main restriction of the work proposed by Pei et al. (2005) is that the support threshold for the algorithms is assumed to be 100%. This restriction has been relaxed in subsequent work (Zeng, Wang, Zhou, & Karypis, 2007). The work by Zeng et al. (2007) examines the problem of mining frequent, closed quasi-cliques from a graph database with arbitrary support thresholds.

Graph Clustering as Dense Subgraph Determination

A closely related problem is that of dense subgraph determination in massive graphs. This problem is frequently encountered in large graph data sets. For example, the problem of determining large subgraphs of web graphs was studied by Gibson et al. (2005). The broad idea in the min-hash approach is to represent the outlinks of a particular node as sets. Two nodes are considered similar if they share many outlinks. Thus, consider a node A with an outlink set S_A, and a node B with outlink set S_B. Then the similarity between the two nodes is defined by the *Jaccard coefficient*, which is defined as $\frac{S_A \cap S_B}{S_A \cup S_B}$. We note that explicit enumeration of all the edges in order to compute this can be computationally inefficient. Rather, a *min-hash approach* is used in order to perform the estimation. This *min-hash approach* is as follows. We sort the universe of nodes in a random order. For any set of nodes in random sorted order, we determine the first node $First(A)$ for which an outlink exists from A to $First(A)$. We also determine the first node $First(B)$ for which an outlink exists from B to $First(B)$. It can be shown that the Jaccard coefficient is an unbiased estimate of the probability that $First(A)$ and $First(B)$ are the same nodes. By repeating this process over different permutations over the universe of nodes, it is possible to accurately estimate the Jaccard coefficient. This is done by using a constant number of permutations c of the node order. The actual permutations are implemented by associated c different randomized hash values with each node. This creates c sets of hash values of size n. The sort-order for any particular set of hash-values defines the corresponding permutation order. For each such permutation, we store the minimum node index of the outlink set. Thus, for each node, there are c such minimum indices. This means that, for each node, a fingerprint of size c can be constructed. By comparing the fingerprints of two nodes, the Jaccard coefficient can be estimated. This approach can be further generalized with the use of every s element set contained entirely with S_A and S_B. Thus, the above description is the special case when s is set to 1. By using different values of s and c, it is possible to design an algorithm which distinguishes between two sets that are above or below a certain threshold of similarity.

The overall technique by Gibson et al. (2005) first generates a set of c shingles of size s for each node. The process of generating the c shingles is extremely straightforward. Each node is processed independently. We use the min-wise hash function approach in order to generate subsets of size s from the outlinks at each node. This results in c subsets for each node. Thus, for each node, we have a set of c shingles. Thus, if the graph contains a total of n nodes, the total size of this shingle fingerprint is $n \times c \times sp$, where sp is the space required for each shingle. Typically, sp will be $O(s)$, since each shingle contains s nodes. For each distinct shingle thus created, we can create a list of nodes which contain it. In general, we would like to determine groups of shingles which contain a large number of common nodes.

In order to do so, the method by Gibson et al. performs a second-order shingling in which the meta-shingles are created from the shingles. Thus, this further compresses the graph in a data structure of size $c \times c$. This is essentially a constant-size data structure. We note that this group of meta-shingles have the the property that they contain a large number of common nodes. The dense subgraphs can then be extracted from these meta-shingles. More details on this approach may be found in the work by Gibson et al.

Clustering Graphs as Objects

In this section, we will discuss the problem of clustering *entire graphs* in a *multigraph database*, rather than examining the node clustering problem within a single graph. Such situations are often encountered in the context of XML data, since each XML document can be regarded as a structural record, and it may be necessary to create clusters from a large number of such objects. We note that XML data is quite similar to graph data in terms of how the data is organized structurally. The attribute values can be treated as graph labels and the corresponding semistructural relationships as the edges. In has been shown by Aggarwal et al. (2007), Dalamagas, Cheng, Winkel, and Sellis (2005), Lee et al. (2002), and Lian, Cheung, Mamoulis, and Yiu (2004) that this structural behavior can be leveraged in order to create effective clusters.

Since we are examining entire graphs in this version of the clustering problem, the problem simply boils down to that of clustering arbitrary *objects*, where the objects in this case have structural characteristics. Many of the conventional algorithms discussed by Jain and Dubes (1998) (such as *k*-means type partitional algorithms and hierarchical algorithms) can be extended to the case of graph data. The main changes required in order to extend these algorithms are as follows:

- Most of the underlying classical algorithms typically use some form of distance function in order to measure similarity. Therefore, we need appropriate measures in order to define similarity (or distances) between structural objects.
- Many of the classical algorithms (such as *k*-means) use *representative objects* such as centroids in critical intermediate steps. While this is straightforward in the case of multidimensional objects, it is much more challenging in the case of graph objects. Therefore, appropriate methods need to be designed in order to create representative objects. Furthermore, in some cases it may be difficult to create representatives in terms of single objects. We will see that it is often more robust to use *representative summaries* of the underlying objects.

There are two main classes of conventional techniques, which have been extended to the case of structural objects. These techniques are as follows:

- *Structural distance-based approach*: This approach computes structural distances between documents and uses them in order to compute clusters of documents. One of the earliest work on clustering tree structured data is the *XClust algorithm* (Lee et al. 2002), which was designed to cluster XML schemas in order for efficient integration of large numbers of document type definitions (DTDs) of XML sources. It adopts the agglomerative hierarchical clustering method which starts with clusters of single DTDs and gradually merges the two most similar clusters into one larger cluster. The similarity between two DTDs is based on their element similarity, which can be computed according to the semantics, structure, and context information of the elements in the corresponding DTDs. One of the shortcomings of the XClust algorithm is that it does not make full use of the structure information of the DTDs, which is quite important in the context of clustering tree-like structures. The method by Chawathe (1999) computes similarity measures based on the structural edit-distance between documents. This edit-distance is used in order to compute the distances between clusters of documents.

  S-GRACE is hierarchical clustering algorithm (Lian et al. 2004). In the work by Lian et al., an XML document is converted to a structure graph (or s-graph), and the distance between two XML documents is defined according to the number of the common element-subelement relationships, which can capture better structural similarity relationships than the tree edit-distance in some cases (Lian et al.).
- *Structural summary-based approach*: In many cases, it is possible to create summaries from the underlying documents. These summaries are used for

creating groups of documents which are similar to these summaries. The first summary-based approach for clustering XML documents was presented by Dalamagas et al. (2005). In the work by Dalamagas et al., the XML documents are modeled as rooted, ordered labeled trees. A framework for clustering XML documents by using structural summaries of trees is presented. The aim is to improve algorithmic efficiency without compromising cluster quality.

A second approach for clustering XML documents is presented by Aggarwal et al. (2007). This technique is a partition-based algorithm. The primary idea in this approach is to use frequent-pattern mining algorithms in order to determine the summaries of frequent structures in the data. The technique uses a *k*-means type approach in which each cluster center comprises a set of frequent patterns which are local to the partition for that cluster. The frequent patterns are mined using the documents assigned to a cluster center in the last iteration. The documents are then further reassigned to a cluster center based on the average similarity between the document and the newly created cluster centers from the local frequent patterns. In each iteration the document assignment and the mined frequent patterns are iteratively reassigned until the cluster centers and document partitions converge to a final state. It has been shown by Aggarwal et al. that such a structural summary-based approach is significantly superior to a similarity function-based approach, as presented by Chawathe (1999). The method is also superior to the structural approach by Dalamagas et al. (2005) because of its use of more robust representations of the underlying structural summaries.

Conclusions and Future Research

In this chapter, we presented a review of the commonly known algorithms for clustering graph data. The problem of clustering graphs has been widely studied in the literature, because of its application to a variety of data mining and data management problems. Graph clustering algorithms are of two types:

- *Node clustering algorithms*: In this case, we attempt to partition the graph into groups of clusters so that each cluster contains groups of nodes which are densely connected. These densely connected groups of nodes may often provide significant information about how the entities in the underlying graph are interconnected with one another.
- *Graph clustering algorithms*: In this case, we have complete graphs available, and we wish to determine the clusters with the use of the structural information in the underlying graphs. Such cases are often encountered in the case of XML data, which are commonly encountered in many real domains.

We provided an overview of the different clustering algorithms available and the trade-offs with the use of different methods. The major challenges that remain in the area of graph clustering are as follows:

- *Clustering massive data sets*: In some cases, the data sets containing the graphs may be so large that they may be held only on disk. For example, if we have a dense graph containing 10^7 nodes, then the number of edges may be as high as 10^{13}. In such cases, it may not even be possible to store the graph effectively on disk. In the cases in which the graph can be stored on disk, it is critical that the algorithm should be designed in order to take the disk-resident behavior of the underlying data into account. This is especially challenging in the case of graph data sets, because the structural behavior of the graph interferes with our ability to process the edges sequentially for many applications. In the cases in which the graph is too large to store on disk, it is essential to design summary structures which can effectively store the underlying structural behavior of the graph. This stored summary can then be used effectively for graph clustering algorithms.
- *Clustering graph streams*: In this case, we have large graphs which are received as edge streams. Such graphs are more challenging, since a given edge cannot be processed more than once during the computation process. In such cases, summary structures need to be designed in order to facilitate an effective clustering process. These summary structures may be utilized in order to determine effective clusters in the underlying data. This approach is similar to the case discussed above in which the size of the graph is too large to store on disk.

In addition, techniques need to be designed for interfacing clustering algorithms with traditional database management techniques. In order to achieve this goal, effective representations and query languages need to be designed for graph data. This is a new and emerging area of research, and can be leveraged upon in order to further improve the effectiveness of graph algorithms.

Cross References

►Group Detection
►Partitional Clustering

Recommended Reading

Abello, J., Resende, M. G., & Sudarsky, S. (2002). Massive quasi-clique detection. In *Proceedings of the 5th Latin American symposium on theoretical informatics (LATIN)* (pp. 598–612). Berlin: Springer.

Aggarwal, C., Ta, N., Feng, J., Wang, J., & Zaki, M. J. (2007). XProj: A framework for projected structural clustering of XML documents. In *KDD conference* (pp. 46–55). San Jose, CA.

Ahuja, R., Orlin, J., & Magnanti, T. (1992). *Network flows: Theory, algorithms, and applications*. Englewood Cliffs, NJ: Prentice-Hall.

Chawathe, S. S. (1999). Comparing hierachical data in external memory. In *Very large data bases conference* (pp. 90–101). San Francisco: Morgan Kaufmann.

Chung, F. (1997). *Spectral graph theory*. Washington, DC: Conference Board of the Mathematical Sciences.

Dalamagas, T., Cheng, T., Winkel, K., & Sellis, T. (2005). Clustering XML documents using structural summaries. In *Information systems*. Elsevier, January 2005.

Gibson, D., Kumar, R., & Tomkins, A. ( ). Discovering large dense subgraphs in massive graphs. In *VLDB conference* (pp. 721-732). http://www.vldb2005.org/program/paper/thu/p721-gibson.pdf

Jain, A., & Dubes, R. (1998). *Algorithms for clustering data*. Englewood, NJ: Prentice-Hall.

Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs, *Bell System Technical Journal, 49*, 291–307.

Lee, M., Hsu, W., Yang, L., & Yang, X. (2002). XClust: Clustering XML schemas for effective integration. In *ACM conference on information and knowledge management*. http://doi.acm.org/10.1145/584792.584841

Lian, W., Cheung, D. W., Mamoulis, N., & Yiu, S. (2004). An efficient and scalable algorithm for clustering XML documents by structure, *IEEE Transactions on Knowledge and Data Engineering, 16*(1), 82–96.

Pei, J., Jiang, D., & Zhang, A. (2005). On mining cross-graph quasi-cliques. In *ACM KDD conference*. Chicago, IL.

Rattigan, M., Maier, M., & Jensen, D. (2007). *Graph clustering with network structure indices. Proceedings of the International Conference on Machine Learning* (783-790). ACM: New York.

Tsay, A. A., Lovejoy, W. S., & Karger, D. R. (1999). Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research, 24*(2), 383–413.

Zeng, Z., Wang, J., Zhou, L., & Karypis, G. (2007). Out-of-core coherent closed quasi-clique mining from large dense graph databases. *ACM Transactions on Database Systems, 32*(2), 13.

Graph Kernels

Thomas Gärtner, Tamás Horváth, Stefan Wrobel
University of Bonn, Fraunhofer IAIS,
Schloss Birlinghoven, Sankt Augustin, Germany

G

Definition

The term *graph kernel* is used in two related but distinct contexts: On the one hand, graph kernels can be defined between graphs, that is, as a *kernel function* $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ where $\mathcal{G}$ denotes the set of all graphs under consideration. In the most common setting $\mathcal{G}$ is the set of all labeled undirected graphs. On the other hand, graph kernels can be defined between the vertices of a single graph, that is, as a kernel function $k : V \times V \to \mathbb{R}$ where V is the vertex set of the graph G under consideration. In the most common setting G is an undirected graph.

Motivation and Background

►Kernel methods are a class of machine learning algorithms that can be applied to any data set on which a valid, that is, positive definite, kernel function has been defined. Many kernel methods are theoretically well founded in statistical learning theory and have shown good predictive performance on many real–world learning problems.

Approaches for Kernels between Graphs

One desireable property of kernels between graphs is that for non-isomorphic graphs $G, G' \in \mathcal{G}$ the functions $k(G, \cdot)$ and $k(G', \cdot)$ are not equivalent. If this property does not hold, the distance is only a pseudometric rather than a metric, that is, non-isomorphic graphs can be mapped to the same point in feature space and no kernel method can ever distinguish between the two graphs. However, it can be seen that computing graph kernels

for which the property does hold is at least as hard as solving graph isomorphism (Gärtner et al., 2003).

For various classes of graphs, special purpose kernels have been defined such as for paths (►string kernels) and trees (Collins & Duffy, 2002). These kernels are typically defined as the number of patterns that two objects have in common or as the inner product in a feature space counting the number of times a particular pattern occurs. The problem of computing a graph kernel where the patterns are all connected graphs, all cycles, or all paths and occurrence is determined by subgraph-isomorphism is, however, NP-hard (Gärtner et al., 2003).

Techniques that have been used to cope with the computational intractability of such graph kernels are (1) to restrict the considered patterns, for example, to bound the pattern size by a constant; (2) to restrict the class of graphs considered, for example, to trees or small graphs; (3) to define occurrence of a pattern differently, that is, not by subgraph-isomorphism; and (4) to approximate the graph kernel. Note that these four techniques can be combined.

While for technique (1) it is not immediately clear if the resulting graph kernel is feasible, technique (2) allows for fixed parameter tractable graph kernels. (Notice that even counting paths or cycles of length k in a graph is #W[1]-complete while the corresponding decision problem is fixed parameter tractable.) Though these will often still have prohibitive runtime requirements, it has been observed that enumerating cycles in real-world databases of small molecules is feasible (Horvath et al., 2004).

With respect to technique (3) it has been proposed to use graph kernels where the patterns are paths but the occurrences are determined by homomorphism (Gärtner et al., 2003; Kashima et al., 2003). Despite the explosion in the number of pattern occurrences (even very simple graphs can contain an infinite number of walks, that is, images of paths under homomorphism), if one downweights the influence of larger patterns appropriately, the kernel takes a finite value and closed form polynomial time computations exist. To increase the practical applicability of these graph kernels, it has been proposed to increase the number of labels by taking neighborhoods into account (Gärtner, 2005) or to avoid "tottering" walks (Mahé et al., 2004).

Various approaches to approximate computation of graph kernels (4) exist. On the one hand, work on computing graph kernels based on restricting the patterns to frequent subgraphs (Deshpande et al., 2002) can be seen as approximations to the intractable all-subgraphs kernel. Computing such graph kernels is still NP-hard and no approximation guarantees are known. On the other hand, a recent graph kernel (Borgwardt et al., 2007) based on sampling small subgraphs of a graph at random is known to have a polynomial time algorithm with approximation guarantees.

The most common application scenario for such graph kernels is the prediction pharmaceutical activity of small molecules.

Approaches for Kernels on a Graph

Learning on the vertices of a graph is inherently *transductive*. Work on kernels between the vertices of a graph began with the "diffusion kernel" (Kondor & Lafferty, 2002) and was later generalized in (Smola and Kondor, 2003) to a framework that contains the diffusion kernel as a special case. Intuitively, these kernels can be understood as comparing the neighborhoods of two vertices in the sense that the more neighbors two vertices have in common, the more similar they are. For classification, this definition is related to making the "cluster assumption", that is, assuming that the decision boundary between classes does not cross "high density" regions of the input space. To compute such graph kernels for increasing sizes of the neighborhood, one needs to compute the limit of a matrix poser series of the (normalized) graph Laplacian or its adjacency matrix. Different graph kernels arise from choosing different coefficients. In general, the limit of such matrix power series can be computed on the eigenvalues. For geometrically decaying parameters, the kernel matrix can also be computed by inverting a sparse matrix obtained by adding a small value to the diagonal of the Laplacian (in which case the kernel is called the "regularized Laplacian kernel") or the adjacency matrix.

In the case of the regularized Laplacian kernel, rather than first computing the kernel matrix and then applying an off-the-shelf implementation of a kernel method, it is often more effective to reformulate the

optimization problem of the kernel method. Several possibilities for such reformulation have been proposed, including changing the variables as in (Gärtner et al., 2006).

The most common application scenario for such graph kernels is the classification of entities in a social network.

Recommended Reading

Borgwardt, K. M., Petri, T., Vishwanathan, S. V. N., & Kriegel, H.-P. (2007). An efficient sampling scheme for comparison of large graphs. In *Mining and learning with graphs* (MLG 2007), Firenze.

Collins, M., & Duffy, N. (2002). Convolution kernel for natural language. In *Advances in neural information proccessing systems (NIPS), 16*, 625–632.

Deshpande, M., Kuramochi, M., & Karypis, G. (2002). Automated approaches for classifying structures. In *Proceedings of the 2nd ACM SIGKDD workshop on data mining in bioinformatics* (BIO KDD 2002).

Gärtner, T. (2005). Predictive graph mining with kernel methods. In S. Bandyopadhyay, U. Maulik, L.B. Holder, and D.J. Cook (Eds.), *Advanced methods for knowledge discovery from complex data*. pp. 95–121, Springer, Heidelberg.

Gärtner, T., Flach, P. A., & Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th annual conference on computational learning theory and the 7th kernel workshop* (COLT 2003), vol. 2777 of LNCS, pp. 129–143, Springer, Heidelberg.

Gärtner, T., Le, Q. V., Burton, S., Smola, A. J., & Vishwanathan, S. V. N. (2006). Large-scale multiclass transduction. In *Advances in neural information processing systems*, vol. 18, pp. 411–418, MIT Press, Cambride, MA.

Horvath, T., Gärtner, T., & Wrobel, S. (2004). Cyclic pattern kernels for predictive graph mining. In *Proceedings of the international conference on knowledge discovery and data mining* (KDD 2004), pp. 158–167, ACM Press, New York, NY.

Kashima, H., Tsuda, K., & Inokuchi, A. (2003). Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning* (ICML 2003), pp. 321–328, AAAI Press, Menlo Park, CA.

Kondor, R. I., & Lafferty, J. (2002). Diffusion kernels on graphs and other discrete input spaces. In C. Sammut & A. Hoffmann (Eds.), *Proceedings of the nineteenth international conference on machine learning* (ICML 2002), pp. 315–322, Morgan Kaufmann, San Fransisco, CA.

Mahé, P., Ueda, N., Akutsu, T., Perret, J.-L., & Vert, J.-P. (2004). Extensions of marginalized graph kernels. In *Proceedings of the 21st international conference on machine learning* (ICML 2004), pp. 70, ACM Press, New York, NY.

Smola, A. J., & Kondor, R. (2003). Kernels and regularization on graphs. In *Proceedings of the 16th annual conference on computational learning theory and the 7th kernel workshop* (COLT 2003), vol. 2777 of LNCS, pp. 144–158, Springer, Heidelberg.

Graph Mining

Deepayan Chakrabarti
Yahoo! Research, Sunnyvale, USA

Definition

Graph Mining is the set of tools and techniques used to (a) analyze the properties of real-world graphs, (b) predict how the structure and properties of a given graph might affect some application, and (c) develop models that can generate realistic graphs that match the patterns found in real-world graphs of interest.

Motivation and Background

A graph $G = (V, E)$ consists of a set of edges, E connecting pairs of nodes from the set V; extensions allow for weights and labels on both nodes and edges. Graphs edges can be used to point *from* one node *to* another, in which case the graph is called directed; in an *undirected* graph, edges must point both ways: $i \rightarrow j \Leftrightarrow j \rightarrow i$. A variant is the bipartite graph $G = (V_1, V_2, E)$ where only edges linking nodes in V_1 to nodes in V_2 are allowed.

A graph provides a representation of the binary relationships between individual entities, and thus is an extremely common data structure. Examples include the graph of hyperlinks linking HTML documents on the Web, the social network graph of friendships between people, the bipartite graphs connecting users to the movies they like, and so on. As such, mining the graph can yield useful patterns (e.g., the communities in a social network) or help in applications (e.g., recommend new movies to a user based on movies liked by other "similar" users). Graph mining can also yield patterns that are common in many real-world graphs, which can then be used to design graph "generators" (e.g., a generator that simulates the Internet topology, for use in testing next-generation Internet protocols).

Structure of Learning System

We split up this discussion into three parts: the analysis of real-world graphs, realistic graph generators, and

applications on graphs. Detailed surveys can be found in Newman (2003) and Chakrabarti and Faloutsos (2006).

Analysis of Real-World Graphs

Four basic types of large-scale patterns have been detected in real-world graphs. The first is the existence of power-laws, for instance in the degree distribution and eigenvalue distribution. Most nodes have very low degree while a few have huge degree. This has implications for algorithms whose running times are bounded by the highest degree. The second set of patterns is called the "small-world phenomenon," which state that the diameter (or effective diameter) of such graphs are very small with respect to their size. Recall that the diameter of a connected graph is the maximum number of hops needed to travel between any pair of nodes; the effective diameter is a more robust version that specifies the number of hops within which a large fraction (say, 90%) of all pairs can reach each other. Examples include a diameter of around 4 for the Internet Autonomous System graph, around 19 for the entire US power grid, around 4 for the graph of actors who worked together in movies, and so on. Third, many large graphs exhibit "community effects," where each community consists of a set of nodes that are more tightly connected to other nodes in the community compared to nodes outside. One local manifestation of this effect is the relatively high *clustering coefficient* which counts, given all pairs of edges (i,j) and (j,k), the probability of the existence of the "transitive" edge (i,k). High clustering coefficients imply tight connections in neighborhoods, which is the basis of strong community structure. Finally, many large graphs were shown to increase in density as they evolve over time, that is, the number of edges grows according to a power-law on the number of nodes. In addition, even while more nodes and edges are being added, the diameter of the graph tends to decrease.

Graph Generators

Imagine designing an application that works on the Internet graph. Collecting the entire Internet graph in one place is hard, making the testing process for such an application infeasible. In such cases, a realistic graph generator can be used to simulate a large "Internet-like" graph, which can be used in place of the real graph. This synthetic graph must match the patterns typically found in the Internet, including the patterns discussed in the previous paragraph. Apart from generating such graphs, the generators can provide insights into the *process* by which large graphs came to attain their structure.

One example of this is the *preferential attachment* model. Starting with a small initial graph, this model adds one new node every step. The new node is connected to m previous nodes, with the probability of connecting to node i being proportional to its degree. This idea, popularly known as "the rich get richer," can be shown to lead to a power-law degree distribution after a large number of nodes and edges have been added.

Many other models have also been proposed, which demonstrate graph generation as a random process, an optimization process, as a process on nodes embedded in some geographic space, and so on.

Applications

Some graph mining algorithms are meant to solve some application on any graph(s) provided as input to the algorithm. Several basic tools are commonly used in such applications, such as the ▶Greedy Search Approach to Graph Mining the ▶Inductive Database Search Approach to Graph Mining spectral methods, graph partitioning methods, and models based on random walks on graphs. *Tree Mining* is a special case of graph mining where the graphs are constrained to be trees. We will discuss a few such applications here.

Frequent subgraph mining: The aim is to find subgraphs that occur very frequently in the particular graph(s) in question (Kuramochi & Karypis, 2001). This is quite useful in chemical datasets consisting of the graph structures of many different molecules (say, all protein molecules that have a certain chemical property); the frequent subgraphs in such molecules might represent basic structural units responsible for giving the molecules their special property. Unfortunately, the frequent subgraph problem subsumes the problem of subgraph isomorphism, and hence is NP-Hard. However, clever techniques have been devised to represent subgraphs so that checking for isomorphism can be done quickly in many cases.

Community detection: The problem is to detect tightly knit groups of nodes, where all nodes in the

group have "similar" linkage structure. There are many algorithms, each optimizing for a different notion of similarity. Examples include graph partitioning methods such as spectral partitioning (Ng, Jordan, & Weiss , 2002) and METIS that try to minimize the number of edges linking nodes across partitions, and co-clustering methods that aim for homogeneity in links across partitions.

Information diffusion and virus propagation: The spread of a contagious disease or a computer virus can be modeled (somewhat crudely) as a contact process on a graph, where the nodes are individuals who can get infected, and the links allow transmission of the contagion from an infected individual to an uninfected one. Similar models have been proposed to model the diffusion of information in social networks. The topology of the graph can be used to infer the most "influential" nodes in the graph, who are most capable of spreading the information quickly throughout the graph (Kempe, Kleinberg, & Tardos , 2003).

Graph kernels: While subgraph isomorphism is a hard problem, we still need to be able to compare graphs on the basis of some similarity measure that can be computed in polynomial time. In the Kernel-Based Approach to Graph Mining graph kernels perform this task by computing similarities based on numbers of walks, paths, cyclic patterns, trees, etc.

Ranking on graphs: Given a graph (say, the Web hyperlink graph), we often need a ranking of the nodes in the graph. The ranking could be static (as in PageRank (Brin & Page, 1998)) or it could depend on a user-specified query node. Such algorithms typically use some version of random walks on graphs (Lovász, 1993), with the probability of the walk hitting a node being correlated with the importance of the node; such importances in turn yield a ranking of the nodes. Both static and query-dependent rankings can be useful in information retrieval settings, where a user desires information pertinent (i.e., "similar") to her query.

Cross References

►Graph Theory
►Greedy Search Approach of Graph Mining
►Inductive Database Search Approach of Graph Mining
►Kernel-Based Approach of Graph Mining
►Link Mining and Link Discovery
►Tree Mining

Recommended Reading

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems, 30* (1–7), 107–117.

Chakrabarti, D., & Faloutsos, C. (2006). Graph mining: Laws, generators and algorithms. *ACM Computing Surveys, 38*(1).

Kempe, D., Kleinberg, J., & Tardos, E. (2003). Maximizing the spread of influence through a social network. In *KDD*.

Kuramochi, M., & Karypis, G. (2001). Frequent subgraph discovery. In *ICDM* (pp. 313–320).

Lovász, L. (1993). Random walks on graphs: A survey. In *Combinatorics: Paul Erdös is eighty* (Vol. 2, pp. 353–397).

Ng, A., Jordan, M., & Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *NIPS*.

The structure and function of complex networks. (2003). *SIAM Review, 45*, 167–256.

Graphical Models

Julian McAuley, Tibério Caetano, Wray Buntine
Statistical Machine Learning Program, NICTA, Canberra, Australia

Definition

The notation we shall use is defined in Table 1, and some core definitions are presented in Table 2. In each of the examples presented in Fig. 1, we are simply asserting that

$$\underbrace{p(x_A, x_B|x_C)}_{\text{function of three variables}} = \underbrace{p(x_A|x_C)p(x_B|x_C)}_{\text{functions of two variables}}, \quad (1)$$

which arises by a straightforward application of the product rule (Definition 1), along with the fact that X_A and X_B are *conditionally independent*, given X_C (Definition 3). The key observation we make is that while the left-hand side of (Eq. 1) is a function of three variables, its conditional independence properties allow it to be *factored* into functions of two variables.

In general, we shall have a series of conditional independence statements about X:

$$\left\{X_{A_i} \perp\!\!\!\perp X_{B_i} \,\middle|\, X_{C_i}\right\}. \quad (2)$$

It is precisely these statements that define the "structure" of our multivariate distribution, which we shall express in the form of a graphical model.

Graphical Models. Table 1 Notation

Notation	Description
$X = (X_1 \dots X_N)$	A random variable (we shall also use $X = (A, B, C \dots)$ in figures to improve readability)
$x = (x_1 \dots x_N)$	A *realization* of the random variable X
$\mathcal{X}$	The sample space (domain) of X
X_A	X can be indexed by a *set*, where we assume $A \subseteq \{1 \dots N\}$
$p(x)$	The probability that $X = x$
$\tilde{A}$	The *negation* of A, i.e., $\{1 \dots N\} \setminus A$
$X_A \perp\!\!\!\perp X_B$	X_A and X_B are *independent*
$X_A \perp\!\!\!\perp X_B \mid X_C$	X_A and X_B are *conditionally independent*, given X_C

Graphical Models. Table 2 Definitions

Definition 1 (product Rule). $p(x_A, x_B) = p(x_A|x_B)p(x_B)$.
Definition 2 (marginalization). $p(x_A) = \sum_{x_{\tilde{A}} \in \mathcal{X}_{\tilde{A}}} p(x_A, x_{\tilde{A}})$.
Definition 3 *(conditional independence). X_A and X_B are said to be conditionally independent (given X_C) iff $p(x_a|x_b, x_c) = p(x_a|x_c)$, for all x_a, x_b, and x_c; the conventional definition of "independence" is obtained by setting $X_C = \varnothing$.*

Graphical Models. Figure 1. Some examples of conditional independence; we say that X_A and X_B are *conditionally independent*, given X_C, or more compactly $X_A \perp\!\!\!\perp X_B \mid X_C$

Motivation and Background

Graphical models are often used to model multivariate data, since they allow us to represent high-dimensional distributions *compactly*; they do so by exploiting the *interdependencies* that typically exist in such data. Put simply, we can take advantage of the fact that high-dimensional distributions can often be decomposed into *low-dimensional factors* to develop efficient algorithms by making use of the distributive law: $ab + ac = a(b + c)$.

Some motivating examples are presented in Fig. 1; similar examples are ubiquitous in fields ranging from computer vision and pattern recognition, to economics and the social sciences. Although we are dealing with high-dimensional data, we can make certain statements about the *structure* of the variables involved, allowing us to express important properties about the distribution compactly. Some of the properties we would like to compute include the probabilities of particular outcomes, and the outcomes with the highest probability.

Theory

Directed Graphical Models

Due to the product rule (Definition 1), it is clear that *any* probability distribution can be written as

$$p(x) = \prod_{i=1}^{N} p(x_{\pi_i} | x_{<\pi_i}) \tag{3}$$

for an arbitrary permutation π of the labels, where we define $< i := \{1 \dots i - 1\}$. For example any four-dimensional distribution can be written as

$$p(x_a, x_b, x_c, x_d) = p(x_c)p(x_b|x_c)p(x_d|x_c, x_b) \\ p(x_a|x_c, x_b, x_d). \tag{4}$$

With this idea in mind, consider a model $p(x)$ for which we have the conditional independence statements

$$\left\{p(x_{\pi_i}|x_{<\pi_i}) = p(x_{\pi_i}|x_{pa_{\pi_i}})\right\}, \tag{5}$$

where $pa_{\pi_i} \subset < \pi_i$. We now have

$$p(x) = \prod_{i=1}^{N} p(x_{\pi_i}|x_{pa_{\pi_i}}). \tag{6}$$

We can interpret pa_i as referring to the "parents" of the node i. Essentially, we are saying that a variable is conditionally independent on its nondescendants, given its parents.

We can represent (Eq. 6) using a directed acyclic graph (DAG) by representing each variable X_i as a node; an arrow is formed from X_j to X_i if $j \in pa_i$. An example of such a representation is given in Fig. 2. It can easily be shown that the resulting graph is always acyclic.

A *Bayesian Network* (a type of *directed* graphical model) is simply a set of probability distributions of the form $p(x) = \prod_{i=1}^{N} p(x_i|x_{pa_i})$. Every Bayesian Network can be represented as a DAG, though we often simply say that the Bayesian Network "is" the DAG. Some trivial examples, and the type of independence statements they imply are shown in Fig. 3.

We finish this section with a simple lemma:

Lemma 4 (topological sort) *Every DAG has at least one permutation π that "sorts" the nodes such that each node has a larger index than its parents; in other words, the factorization associated to any DAG can be written in the form of (Eq. 6) for at least one π such that $\pi_i > j$ for all i, where $j \in pa_{\pi_i}$.*

Undirected Graphical Models

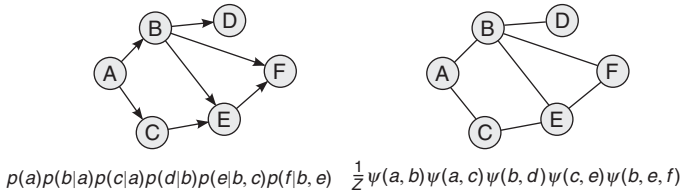
Although we have shown how conditional independence statements in the form of (Eq. 5) can be modeled using a DAG, there exist certain conditional independence statements that are not satisfied by *any* Bayesian Network, such as those in Fig. 4.

Markov random fields (or MRFs) allow for the specification of a different class of conditional independence statements, which are naturally represented by *undirected graphs* (UGs for short). The results associated with MRFs require a few additional definitions:

Definition 5 (clique) *A set of nodes X in a graph $\mathcal{G} = (V, E)$ is said to form a clique if $(X_i, X_j) \in E$ for every $X_i, X_j \in X$ (i.e., the subgraph X is fully connected).*

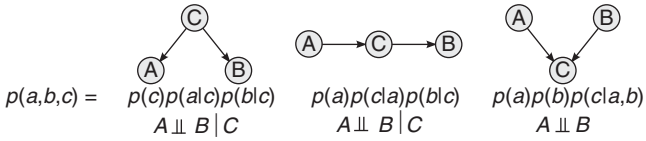
Definition 6 (maximal clique) *A clique X is said to be maximal if there is no clique Y such that $X \subset Y$.*

A Markov random field is a probability distribution of the form $p(x) = \frac{1}{Z}\prod_{c\in\mathcal{C}} \psi_c(x_c)$, where $\mathcal{C}$ is the set of maximal cliques of $\mathcal{G}$, ψ_c is an arbitrary nonnegative real-valued function and Z is simply a normalization constant ensuring that $\sum_x p(x) = 1$.

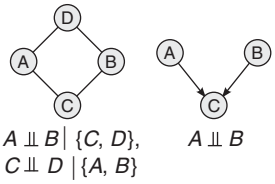


$p(a)p(b|a)p(c|a)p(d|b)p(e|b,c)p(f|b,e)$ $\quad \frac{1}{Z}\psi(a,b)\psi(a,c)\psi(b,d)\psi(c,e)\psi(b,e,f)$

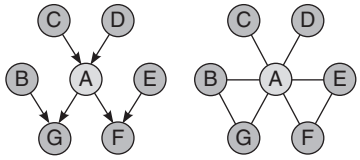
Graphical Models. Figure 2. A directed model (*left*) and an undirected model (*right*). The joint distributions they represent are shown



Graphical Models. Figure 3. Some simple Bayesian Networks, and their implied independence statements. Note in particular that in the rightmost example, we *do not* have $A \perp\!\!\!\perp B \mid C$



Graphical Models. Figure 4. There is no Bayesian Network that captures precisely the conditional independence properties of the Markov random field at *left*; there is no Markov random field that captures precisely the conditional independence properties of the Bayesian Network at *right*



Graphical Models. Figure 5. The Markov Blanket of the node *A* consists of its parents, its children, and the parents of its children (*left*). The corresponding structure for *undirected* models simply consists of the neighbors of *A*. Note that if we convert the directed model to an undirected one (using the procedure described in Section "Conversion from directed to undirected graphical models"), then the Markov Blankets of the two graphs are identical

Conversion from Directed to Undirected Graphical Models

It is possible to convert a directed graphical model to an undirected graphical model via the following simple procedure:

- For every node X_i with parents pa_{X_i}, add undirected edges between every $X_j, X_k \in pa_{X_i}$.
- Replace all directed edges with undirected edges.

In other words, we are replacing statements of the form $p(x_A|x_B)$ with $\psi(x_A, x_B)$, so that the nodes $\{X_i\} \cup pa_{X_i}$ now form a clique in the undirected model. This procedure of "marrying the parents" is referred to as *Moralization*. Naturally, the undirected model formed by this procedure does not precisely capture the conditional independence relationships in the directed version. For example, if it is applied to the graph in Fig. 4 (*right*), then the nodes A, B, and C form a clique in the resulting model, which does not capture the fact that $A \perp\!\!\!\perp B$. However, we note that every term of the form $p(x_i|x_{pa_i})$ appears in some clique of the undirected model, meaning that it can include all of the factors implied by the Bayesian Network.

Characterization of Directed and Undirected Graphical Models

We can now present some theorems that characterize both Bayesian Networks and Markov random fields:

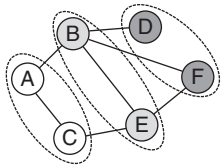
Lemma 7 (Local Markov Property) *A node in a DAG is conditionally independent of its non-descendants, given its parents (this is referred to as the "Directed" Local Markov Property); a node in a UG is conditionally independent of its non-neighbors, given its neighbors.*

Definition 8 (Markov Blanket) *Given a node A, its "Markov Blanket" is the minimal set of nodes C such that $A \perp\!\!\!\perp B \mid C$ for all other nodes B in the model (in other words, the minimal set of nodes that we must know to "predict" the behavior of A).*

Lemma 9 (Markov Blankets of Directed and Undirected Graphs) *In a directed network, the Markov Blanket of a node A (denoted $MB(A)$) consists of its parents, its children, and its children's (other) parents. In an undirected network, it simply consists of the node's neighbors (see Fig. 5).*

Definition 10 (d-separation) *The notion of a Markov Blanket can be generalized to the notion of "d-separation". A set of nodes A is said to be d-separated from a set B by a set C if every (undirected) path between A and B is "blocked" when C is in the conditioning set (i.e., when C is observed). A path is said to be blocked if **either** it contains (p_1, p_2, p_3) with $p_1 \rightarrow p_2 \leftarrow p_3$ (where arrows indicate edge directions) and neither p_2 nor any of its descendants are observed, **or** it contains (p_1, p_2, p_3) with $p_1 \rightarrow p_2 \rightarrow p_3$ and p_2 is observed **or** it contains (p_1, p_2, p_3) with $p_1 \leftarrow p_2 \rightarrow p_3$ and p_2 is observed.*

Applying (Definition 10) to the directed graphs in Fig. 1, we would say that the aqua regions (X_C) *d-separate* the red regions (X_A) from the white regions



Graphical Models. Figure 6. The nodes $\{B, E\}$ form a *clique*; the nodes $\{B, E, F\}$ form a *maximal clique*. The nodes $\{B, E\}$ *separate* the nodes $\{A, C\}$ from $\{D, F\}$

(X_B); *all conditional independence statements can simply be interpreted as d-separation in a DAG.*

The analogous notion of *graph separation* for Markov random fields is simpler than that of d-separation for Bayesian Networks. Given an undirected graph $\mathcal{G}$ and disjoint subsets of nodes A, B, C, if A is only reachable from B via C, this means that A is *separated* from B by C and these semantics encode the probabilistic fact that $A \perp\!\!\!\perp B \mid C$. This is illustrated in Fig. 6.

In both the directed and undirected case, A Markov Blanket of a node is simply the minimal set of nodes that d-separates/graph separates that node from all others.

A complete characterization of the class of probability distributions represented by Bayesian Networks can be obtained naturally once conditional independence statements are mapped to d-separation statements in a DAG. The following theorem settles this characterization.

Theorem 11 *Let p be a probability distribution that satisfies the conditional independence statements implied by d-separation in a DAG. Then p factors according to (Eq. 6). The converse also holds.*

For Markov random fields, an analogous characterization exists:

Theorem 12 (Hammersley-Clifford) *If a strictly positive probability distribution p satisfies the conditional independence statements implied by graph-separation in an undirected graph $\mathcal{G}$, then*

$$p(x) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \psi_c(x_c). \tag{7}$$

The converse also holds, albeit in a more general sense in that p need not be strictly positive.

It can be shown that

directed local Markov property		*local Markov property*
⇕		⇕
d-separation in a DAG	and (for positive p) that	*graph separation in a UG*
⇕		⇕
factorization of p by (Eq. 6)		*factorization of p by* (Eq. 7)

Knowing that directed models can be converted to undirected models, we shall consider inference algorithms in undirected models only.

Applications

Inference Algorithms in Graphical Models

The key observation that we shall rely on in order to do inference efficiently is the *distributive law*:

$$\underbrace{ab + ac}_{\text{three operations}} = \underbrace{a(b + c)}_{\text{two operations}}. \tag{8}$$

By exploiting the factorization in a graphical model, we can use this law to perform certain queries efficiently (such as computing the marginal with respect to a certain variable).

As an example, suppose we wish to compute the marginal $p(x_1)$ in an MRF with the following factorization:

$$p(x) = \frac{1}{Z} \prod_{i=1}^{N-1} \psi(x_i, x_{i+1}). \tag{9}$$

Note that the graph representing this model is simply a *chain*. Computing the sum in the naïve way requires computing

$$p(x_1) = \frac{1}{Z} \sum_{x_{\{2 \ldots N\}}} \prod_{i=1}^{N-1} \psi(x_i, x_{i+1}), \tag{10}$$

whose complexity is $\Theta(\prod_{i=1}^{N}|\mathcal{X}_i|)$. However, due to the distributive law, the same result is simply

$$p(x_1) = \frac{1}{Z}\sum_{x_2}\Big[\psi(x_1,x_2)\sum_{x_3}\Big[\psi(x_2,x_3)\cdots \sum_{x_{N-1}}\Big[\psi(x_{N-2},x_{N-1})\sum_{x_N}\psi(x_{N-1},x_N)\Big]\Big]\Big], \quad (11)$$

whose complexity is $\Theta(\sum_{i=1}^{N-1}|\mathcal{X}_i||\mathcal{X}_{i+1}|)$. As a more involved example, consider computing the marginal with respect to A in the undirected model in Fig. 2; here we wish to compute

$$p(a) = \frac{1}{Z}\sum_{b,c,d,e,f}\psi(a,b)\psi(a,c)\psi(b,d)\psi(c,e)\psi(b,e,f) \quad (12)$$

$$= \frac{1}{Z}\sum_{b}\psi(a,b)\sum_{c}\psi(a,c)\sum_{d}\psi(b,d)\sum_{e}\psi(c,e)\sum_{f}\psi(b,e,f). \quad (13)$$

Exploiting the distributive law in this way is often referred to as the *Elimination Algorithm*. It is useful for computing the marginal with respect to a single variable. However, should we wish to compute the marginal with respect to *each* variable (for example), it is not an efficient algorithm as several operations shall be repeated.

Belief-Propagation In tree-structured models, the elimination algorithm can be adapted to avoid repeated computations, using a message-passing scheme known as *Belief Propagation*, or the *sum-product* algorithm. This is presented in Algorithm 1. Here the "cliques" in the model are simply edges. This algorithm was invented independently by many authors, and is the most efficient amongst many variations.

It can be easily demonstrated that the condition in Algorithm 1, Line 3 is always satisfied by some pair of edges until all messages have been passed: initially, it is satisfied by all of the "leaves" of the model; messages are then propagated inwards until they reach the "root" of the tree; they are then propagated outwards.

Maximum A Posteriori (MAP) Estimation Algorithm 1 allows us to compute the *marginals* of the variables in a graphical model. There are other related properties that we may also wish to compute, such as finding which states have the highest probability (the *Maximum A Posteriori*, or simply "MAP" states). To do so, we note that the operations $(+,\times)$ used in Algorithm 1 can be replaced by $(\max,\times)$. This variant is usually referred to as the *max-product* (as opposed to *sum-product*) algorithm. Indeed, different quantities can be computed by replacing $(+,\times)$ by any pair of operations that form a *semiring* (Aji & McEliece, 2000).

The Junction-Tree Algorithm Algorithm 1 applies only for tree-structured graphs. We can generalize this algorithm to general graphs. We do so by working with a different type of tree-structured graph, whose *nodes* contain the *cliques* in our original graph. We begin with some definitions:

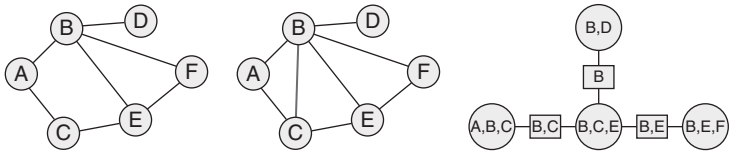
Definition 13 (chordal graph) *A graph $\mathcal{G}$ is said to be chordal if every cycle $(c_1 \ldots c_n)$ in $\mathcal{G}$ contains a chord (i.e., an edge (c_i, c_j) such that $j > (i+1)$).*

Definition 14 (clique-graph, clique-tree) *A clique-graph $\mathcal{H}$ of a graph $\mathcal{G}$ is a graph whose nodes consist of (maximal) cliques in $\mathcal{G}$, and whose edges correspond to intersecting cliques in $\mathcal{G}$. A clique-tree is a clique-graph without cycles.*

Algorithm 1 The *sum-product* algorithm

Input: an undirected, tree-structured graphical model $\mathcal{X}$ with cliques $\mathcal{C}$ {the cliques are simply *edges* in this case}

1: define $m_{A\to B}(x_{A\cap B})$ to be the "message" from an edge A to an adjacent edge B {for example if $A = (a,b)$ and $B = (b,c)$ then we have $m_{(a,b)\to(b,c)}(x_b)$}
2: **while** there exist adjacent edges $A, B \in \mathcal{C}$ for which $m_{A\to B}$ has not been computed **do**
3: find some $A \in \mathcal{C}$ such that $m_{C\to A}$ has been computed for every neighbor $C \in \Gamma(A)$, *except* B {$\Gamma(A)$ returns the edges neighboring A; initially the condition is satisfied by all leaf-edges}
4: $m_{A\to B}(x_{A\cap B}) :=$ $\sum_{x_{A\setminus B}}\{\psi_A(x_A)\prod_{C\in\Gamma(A)\setminus B} m_{C\to A}(x_{A\cap C})\}$
5: **end while**
6: **for** $A \in \mathcal{C}'$ **do**
7: $marginal_A(x_A) :=$ $\psi_A(x_A)\prod_{C\in\Gamma(A)} m_{C\to A}(x_{A\cap C})$
8: **end for**



Graphical Models. Figure 7. The graph at *left* is not chordal, since the cycle (A, B, E, C) does not contain a chord; adding the edge (B, C) results in a chordal (or triangulated) graph (*centre*). The graph at *right* is a Junction-Tree for the graph at *centre*; the cliques of the triangulated graph form the nodes (circles); their *intersection sets* are shown as squares. Note that this is not the only Junction-Tree that we could form – the node $\{B, D\}$ could connect to any of the other three nodes

Definition 15 (Junction-Tree) *A clique-tree $\mathcal{H}$ of $\mathcal{G}$ is said to form a Junction-Tree if for every pair of nodes A, B (i.e., maximal cliques in $\mathcal{G}$), the path between them $(P_1 \ldots P_m)$ satisfies $(A \cap B) \subset P_i$ for all $i \in \{1 \ldots m\}$.*

The algorithms we shall define apply only if the graph in question is *chordal*, or "triangulated" (Definition 13); this can always be achieved by adding additional edges to the graph, as demonstrated in Fig. 7; adding additional edges means increasing the size of the maximal cliques in the graph.

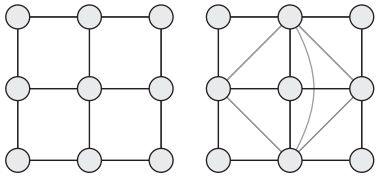
Finding the "optimal" triangulation (i.e., the one that minimizes the size of the maximal cliques) is an NP-complete problem. In practice, triangulation algorithms vary from simple greedy heuristics (e.g., select a node that has as few neighbors as possible), to complex approximation algorithms working within a factor of the optimal solution (Amir, 2001).

The problem of actually *generating* a Junction-Tree from the triangulated graph is easily solved by a maximum spanning tree algorithm (where we prefer edges corresponding to pairs of cliques with large intersections).

Theorem 16 *Let $\mathcal{G}$ be a triangulated graph and $\mathcal{H}$ a corresponding clique-tree. If the sum of the cardinalities of the intersection sets of $\mathcal{H}$ is maximum, then $\mathcal{H}$ is a Junction Tree. The converse also holds.*

If the nodes and edges in Algorithm 1 are replaced by the nodes (maximal cliques in $\mathcal{G}$) and edges (intersecting cliques in $\mathcal{G}$) of $\mathcal{H}$, then we recover the *Junction-Tree Algorithm*.

Approximate Inference The act of triangulating the graph in the Junction-Tree Algorithm may have the effect of increasing the size of its maximal cliques, as in Fig. 8. This may be a problem, as its running time is exponential in the size of the maximal cliques in the *triangulated* graph (this size minus one is referred to as the *tree-width* of the graph, e.g., a chain has a tree-width of 1).



Graphical Models. Figure 8. The graph above at *left* has maximal cliques of size two; in order to triangulate it, we must introduce maximal cliques of size four (*right*)

There are a variety of approximate algorithms that allow us to perform inference more efficiently:

Variational approximation. If doing inference in a graphical model $\mathcal{X}$ is intractable, we might search for a model $\mathcal{Y}$ for which inference is tractable, and which is "similar" to $\mathcal{X}$ in terms of the KL-divergence between $p(x)$ and $p(y)$. (Wainwright & Jordan, 2008).

Loopy belief-propagation. We can build a clique-graph from a graph that has not been triangulated, simply by connecting all cliques that intersect (in which case, the clique-graph will contain loops). If we then propagate messages in some *random* order, we can obtain good approximations under certain conditions (Ihler et al., 2005).

Gibbs sampling. Given an estimate $x_{A \setminus B}$ of a set of variables $X_{A \setminus B}$, we can obtain an estimate of x_B by sampling from the conditional distribution $p(x_B | x_{A \setminus B})$. If we choose $B = \{X_i\}$, and repeat the procedure for

random choices of $i \in \{1 \ldots N\}$, we obtain the procedure known as *Gibbs Sampling* (Geman & Geman, 1984).

There are several excellent books and tutorial papers on graphical models. A selection of tutorial papers includes Aji and McEliece (2000), Kschischang, Frey, and Loeliger (2001), Murphy (1998), Wainwright and Jordan (2008); review articles include Roweis and Ghahramani (1997) and Smyth (1998), to name but a few.

A selection of works includes Koller and Friedman (2009), Jensen (2001) (introductory books), Edwards (2000) (undirected models), Pearl (1988, 2000) (directed models), Cowell, Dawid, Lauritzen, and Spiegelhalter (2003) (exact inference), Jordan (1998) (learning and approximate inference) and Lauritzen (1996, Lauritzen and Spiegelhalter (1988) (a comprehensive mathematical theory).

There is also a variety of closely related models and extensions:

Gaussian graphical models. We have assumed throughout that our probability distributions are *discrete*; however, the only condition we require is that they are *closed under multiplication and marginalization*. This property is also satisfied for *Gaussian* random variables.

Hidden Markov models. In many applications, the variables in our model may be *hidden*. The above algorithms can be adapted to infer properties about our hidden states, given a sequence of observations.

Kalman filters. Kalman filters employ both of the above ideas, in that they include hidden state variables taking values from a *continuous* space using a Gaussian noise model. They are used to estimate the states of *linear dynamic systems* under noise.

Factor graphs. Factor graphs employ an alternate message-passing scheme, which may be preferable for computational reasons. Inference remains approximate in graphs with loops, though approximate solutions may be obtained more efficiently than by Loopy Belief-Propagation (Kschischang et al., 2001).

Relational models. Relational models allow us to explore the relationships between objects in order to predict the behavior and properties of each. Graphical models are used to predict the properties of an object based on others that relate to it (Getoor & Taskar, 2007).

Learning. Often, we would like to *learn* either the parameters or the structure of the model from (possibly incomplete) data. There is an extensive variety of approaches; a collection of papers appears in Jordan (1998).

Cross References

►Bayesian Network
►Expectation Propogation
►Hidden Markov Models
►Markov Random Field

Recommended Reading

Aji, S. M., & McEliece, R. J. (2000). The generalized distributive law. *IEEE transactions on information theory, 46*(2): 325-343.

Amir, E. (2001). Efficient approximation for triangulation of minimum treewidth. In *Proceedings of the 17th conference on uncertainty in artificial intelligence* (pp. 7–15). San Francisco: Morgan Kaufmann.

Cowell, R. G., Dawid, P. A., Lauritzen, S. L., & Spiegelhalter, D. J. (2003). *Probabilistic networks and expert systems*. Berlin: Springer.

Edwards, D. (2000). *Introduction to graphical modelling*. New York: Springer.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions and the bayesian restoration of images. In *IEEE transactions on pattern analysis and machine intelligence, 6*, 721–741.

Getoor, L., & Taskar, B. (Eds.). (2007). *An introduction to statistical relational learning*. Cambridge, MA: MIT Press.

Ihler, A. T., Fischer III, J. W., & Willsky, A. S. (2005). Loopy belief propagation: Convergence and effects of message errors. *Journal of Machine Learning Research, 6*, 905–936.

Jensen, F. V. (2001). *Bayesian networks and decision graphs*. Berlin: Springer.

Jordan, M. (Ed.). (1998). *Learning in graphical models*. Cambridge, MA: MIT Press.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: Principles and techniques*. Cambridge, MA: MIT Press.

Kschischang, F. R., Frey, B. J., & Loeliger, H. A. (2001). Factor graphs and the sum-product algorithm. *IEEE transactions on information theory, 47*(2), 498–519.

Lauritzen, S. L. (1996). *Graphical models*. Oxford: Oxford University Press.

Lauritzen, S. L., & Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application

to expert systems. *Journal of the Royal Statistical Society, Series B, 50*, 157–224.
Murphy, K. (1998). *A brief introduction to graphical models and Bayesian networks*. San Francisco: Morgan Kaufmann.
Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Francisco: Morgan Kaufmann.
Pearl, J. (2000). *Causality*. Cambridge: Cambridge University Press.
Roweis, S., & Ghahramani, Z. (1997). A unifying review of linear Gaussian models. *Neural Computation, 11*, 305–345.
Smyth, P. (1998). Belief networks, hidden Markov models, and Markov random fields: A unifying view. *Pattern Recognition Letters, 18*, 1261–1268.
Wainwright, M. J., & Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning, 1*, 1–305.

Graphs

Tommy R. Jensen
Alpen-Adria-Universität Klagenfurt,
Klagenfurt, Austria

Definition

Graph Theory is (dyadic) relations on collections specified objects. In its most common, a *graph* is a pair $G = (V, E)$ of a (finite) set of *vertices* V and a set of *edges* E (or *links*). Each edge e is a 2-element subset $\{u, v\}$ of V, usually abbreviated as $e = uv$; u and v are called the *endvertices* of e, they are mutually *adjacent* and each is *incident* to e in G. This explains the typical model of a *simple graph*.

A *directed graph* or ▶digraph is a more general structure, in which the edges are replaced by ordered pairs of distinct elements of the vertex set V, each such pair being referred to as an *arc*. Another generalization of a graph is a *hypergraph* or "set-system" on V, in which the *hyperedges* may have any size. Various concepts in graph theory extend naturally to *multigraphs*, in which each pair of (possibly identical) vertices may be adjacent via several edges (respectively *loops*). Also studied are *infinite graphs*, for which the vertex and edge sets are not restricted to be finite.

A graph is conveniently depicted graphically by representing each vertex as a small circle, and representing each edge by a curve that joins its two endvertices. A digraph is similarly depicted by adding an arrow on the curve representing an *arc* showing the direction from its *tail* to its (possibly identical) *head*.

Motivation and Background

One of the very first results in graph theory appeared in Leonhard Euler's paper on Seven Bridges of Königsberg, published in 1736. The paper contained the complete solution to the problem whether, when given a graph, it is possible to locate an *Euler tour*, that is, a sequence of adjacent edges (each edge imagined to be traversed from one end to the other) that uses every edge exactly once. Figure 1 illustrates the four main parts of the city of Königsberg with the seven bridges connecting them; since this graph contains four vertices of odd degree, it does not allow an Euler tour.

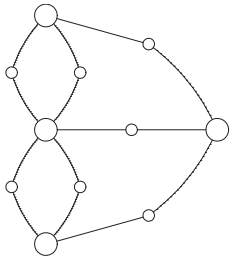
Applications of graphs are numerous and widespread. Much of the success of graph theory is due to the ease at which ideas and proofs may be communicated pictorially in place of, or in conjunction with, the use of purely formal symbolism.

Theory

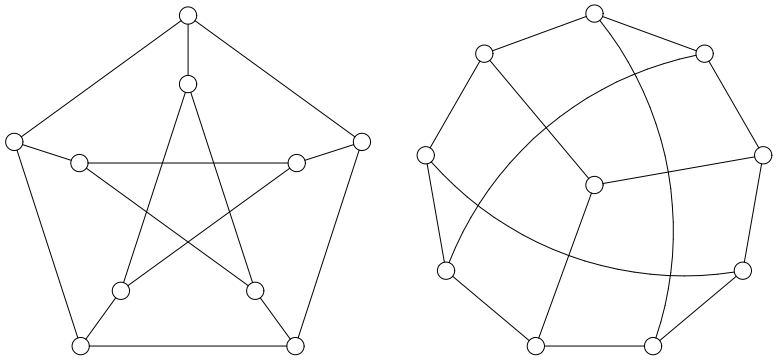
Isomorphism

A graph drawing should not be confused with the graph itself (the underlying abstract structure) as there are several ways to structure the graph drawing. It only matters which vertices are connected to which others by how many edges, the exact layout may be suited for the particular purpose at hand. It is often a problem of independent interest to optimize a drawing of a given graph in terms of aesthetic features.

In practice it is often difficult to decide if two drawings represent the same graph (as in Fig. 2). This



Graphs. Figure 1. A graph of the city of Königsberg



Graphs. Figure 2. Two drawings of the same graph

decision problem has gained increasing status in complexity theory, with growing suspicion that this problem may fall in a new class of problems, which lies between the familar classes of polynomially solvable and NP-complete ▶(NP-completeness) problems (supposing that these classes are indeed distinct; for issues related to the complexities of decision and optimization problems see Garey & Johnson, (1979)). Nonetheless it is customary in the treatment of abstract graphs to consider two graphs identical if they are isomorphic. A closely related problem, the *subgraph isomorphism problem*, an NP-complete problem, consists in finding a given graph as a subgraph of another given graph.

Whereas there seems common agreement in the graph theoretic community on what constitutes a drawing of a graph, it may be considered a weakness, and sometimes a source of confusion, that even the most central general sources on the fundamentals of graph theory, such as the monographs (Berge, 1976; Bondy & Murty, 2007; Diestel, 2005), do not agree on a common formalization of the theory.

Classes of Graphs

Important special classes of graphs are *bipartite graphs*, for which the vertex set is partitionable into two classes A, B with every edge having one end in A and one in B; in particular the *complete bipartite graph* $K_{m,n}$ has $|A| = m$, $|B| = n$, and every vertex in A is joined to every vertex in B. The *complete graph* K_n consists of n vertices that are all pairwise adjacent. A *path* of length n consists of vertices $v_0, v_1, \ldots, v_n$ with edges $v_{i-1}v_i$ for $i = 1, 2, \ldots, n$; such a path *joins* its two endvertices v_0 and v_n. A *circuit* of length n consists of a path of length $n-1$ together with an additional edge between the two endvertices of the path. A graph is *connected* if each pair of its vertices is joined by at least one path within the graph. Of central importance to the study of efficient search procedures in computer science is the class of *trees*, those connected graphs that contain no circuits. Most definitions have various natural counterparts for directed graphs, in particular a *tournament* is a directed graph in which each pair of vertices is joined by exactly one arc.

Properties of Graphs

Finding a complete subgraph of a given order in an input graph is called the *clique problem*. The complementary problem of finding an independent set is called the *independent set problem*. The *longest path problem* and the *longest circuit problem* have as special cases the *Hamilton path problem* and the *Hamilton circuit problem*, the latter two problems asking to find a path, respectively a circuit, that uses all vertices of the given graph. Each of these problems (or a suitable modification of it) belongs to the complexity class of NP-complete problems, hence is generally believed to be very difficult to solve efficiently. The weighted version of the Hamilton circuit problem, the so-called *travelling salesman problem* is of central importance in combinatorial optimization.

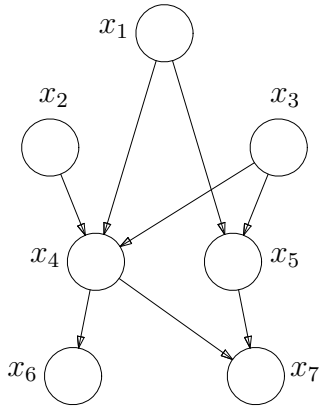
A graph is called *planar* if it may be drawn in the Euclidian plane without any two of its edges crossing except where they meet at a common endvertex. This is often a convenient way of representing a graph, whenever it is doable. A theorem of Kuratowski states that a graph is planar if and only if it contains

homeomorphic copies of neither the complete bipartite graph $K_{3,3}$ (the three-houses-three-utilities-graph) nor the complete graph K_5. A main branch of graph theory is concerned with investigating relationships between the topological and combinatorial properties of graphs Mohar & Thomassen, (2001).

In 1852, Francis Guthrie posed the *four color problem*, asking if it is possible to color the countries of any map, using only four colors, in such a way that all pairs of bordering countries receive different colors. Equivalently, by representing dually every country as a vertex of a graph, with two vertices joined by an edge if their countries share a stretch of common border, the question is whether it is possible to color the vertices of a planar graph using four colors, so that any two adjacent vertices receive distinct colors. This problem, was solved a century later in 1976 by Kenneth Appel, Wolfgang Haken, and John Koch, who invested massive amounts of computing time to complete a graph theoretic approach developed by various mathematicians over a period of most of the preceding part of the twentieth century.

The problem of *coloring* a possibly nonplanar graph with a minimal number of colors, that is, to partition its vertex set into as few independent sets as possible, is a well-studied problem (e.g., see Jensen & Toft, (1995)), though NP-hard in general. In fact it is already an NP-complete problem to ask whether a given planar graph allows a coloring using at most three colors (see Garey, Johnson & Stockmeyer 1976). The recent strong perfect graph theorem provides one of quite few known examples of a fairly rich class of graphs, the *Berge graphs*, for which the coloring problem has a satisfactory solution (see Chudnovsky, Robertson, Seymour & Thomas 2006).

Other well-solved problems include finding a largest *matching* in a given graph; a largest set of edges no two of which share a common endvertex (see Lovász & Plummer (1986) for a thorough treatment of *matching theory*). The most interesting special case asks to find a *perfect matching*, having the property that every vertex is paired up with a unique vertex of the graph adjacent to it. For the special case of bipartite graphs (the *marriage problem*), the problem was solved by Dénes König in 1931. Even when given for every pair of vertices a measure of the desirability of pairing up these particular vertices (the *weighted matching problem*), there exists an efficient solution to the problem of finding an optimum matching of maximal total weight, discovered by Jack Edmonds in 1959.



Graphs. Figure 3. Reproduced from Bishop (2006, p. 362)

Applications

As an example of a visualization application, Fig. 3 shows a digraph to symbolize for a collection of seven stochastic variables $x_1, \ldots, x_7$ that their joint distribution is given by the product

$$p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3)p(x_5|x_1, x_3) \times p(x_6|x_4)p(x_7|x_4, x_5)$$

In addition to visualization of a network, a process, a search procedure, or any hierarchical structure, there are many applications using implementations of known graph algorithms on computers, so that the graph in question will only exist as an abstract datastructure within a program and thus remains invisible to the user.

There are different ways to store graphs in a computer. Often a combination of list and matrix structures will be preferred for storage and dynamic manipulation of a graph by an algorithm. List structures are often preferred for sparse graphs as they have smaller memory requirements. Matrix structures on the other hand provide faster access but can consume a large amount of memory if a graph contains many vertices. In most cases it is convenient to represent a graph or digraph by an array containing, for each edge or arc, the pair of vertices that it joins, together with additional information, such as the weight of the edge, as appropriate.

It may be an advantage in addition to store for each vertex a list of the vertices adjacent to it, or alternatively, a list of the edges incident to it, depending on the application.

The adjacency matrix of a graph, multigraph, or digraph on n vertices is an $n \times n$ matrix in which the ij-entry is the number of edges or arcs that join vertex i to vertex j (or more generally, the weight of a single such edge or arc). As a storage device this is inferior for sparse graphs, those with relatively few edges, but gains in importance when an application naturally deals with very dense graphs or multigraphs.

Future Directions

In recent years the theory of *graph minors* has been an important focus of graph theoretic research. A graph H is said to be a *minor* of a graph G if there exists a subgraph of G from which H can be obtained through a sequence of *edge contractions*, each consisting of the identification of the two ends of an edge e followed by the removal of e. A monumental effort by Neil Robertson and Paul Seymour has resulted in a proof of the Robertson–Seymour theorem (Robertson & Seymour, 2004; see also Diestel, 2005), with the important consequence that for any set $\mathcal{G}$ of graphs that is closed under taking minors, there exists a finite set of obstruction graphs, such that G is an element of $\mathcal{G}$ precisely if G does not contain any minor that belongs to the obstruction set. This theorem has several important algorithmic consequences, many still waiting to be fully explored.

A particularly challenging unsolved problem is the *Hadwiger conjecture* (see Jensen & Toft, 1995), stating that any graph G that does not allow a vertex coloring with as few as k colors will have to contain the complete graph K_{k+1} as a minor. The special cases of $k \leq 5$ colors have been shown to be consequences of the four color theorem. But the problem remains open for all larger values of k.

Other central areas of research relate to the notoriously hard problems of vertex- and edge-coloring, and of Hamilton paths and circuits. These have important applications, but it is not expected that any satisfactory necessary and sufficient conditions will be found for their existence. Hence the study of sufficient conditions of practical value is lively pursued.

A list of open problems in graph theory can be found in Bondy & Murty (2007).

Recommended Reading

Bang-Jensen, J., & Gutin, G. (2000). *Digraphs: theory, algorithms and applications. Springer monographs in mathematics*, London: Springer. http://www.imada.sdu.dk/Research/Digraphs/

Berge, C. (1976). *Graphs and hypergraphs. North-Holland mathematical library* (Vol. 6).

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Bondy, J. A., & Murty, U. S. R. (2007) *Graph theory*, Springer.

Chudnovsky, M., Robertson, N., Seymour, P., & Thomas, R. (2006). The strong perfect graph theorem. *Annals of Mathematics, 164*, 51–229.

Diestel, R. (2005). *Graph theory* (3rd ed.). Springer. http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/GraphTheoryIII.pdf

Emden-Weinert, T. Graphs: theory–algorithms–complexity. http://people.freenet.de/Emden-Weinert/graphs.html.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A guide to the theory of NP-completeness*. New York: Freeman.

Garey, M. R., Johnson, D. S., & Stockmeyer, L. J. (1976). Some simplified NP-complete graph problems. *Theoretical Computer Science, 1*, 237–267.

Gimbel, J., Kennedy, J. W., & Quintas, L. V. (Eds.). (1993). *Quo Vadis, graph theory?* North-Holland.

Harary, F. (1969). *Graph theory*. Reading: Addison-Wesley.

Jensen, T. R., & Toft, B. (1995). *Graph coloring problems*. Wiley.

Locke, S. C. Graph theory.http://www.math.fau.edu/locke/graphthe.htm.

Lovász, L., & Plummer, M. D. (1986). *Matching theory. Annals of discrete math* (Vol. 29). North Holland.

Mohar, B., & Thomassen, C. (2001). *Graphs on surfaces*. John Hopkins University Press.

Robertson, N., & Seymour, P. D. (2004). Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B, 92*(2), 325–357.

Weisstein, E. W. Books about graph theory. http://www.ericweisstein.com/encyclopedias/books/GraphTheory.html.

Greedy Search

Claude Sammut
University of New South Wales, Sydney, Australia

At each step in its search, a greedy algorithm makes the best decision it can at the time and continues without backtracking. For example, an algorithm may perform a ▶general-to-specific search and at each step, commits itself to the specialization that best fits that training data, so far. It continues without backtracking to change any

of its decisions. Greedy algorithms are used in many machine-learning algorithms, including decision tree learning (Breiman, Friedman, Olshen, & Stone, 1984; Quinlan, 1993) and ►rule learning algorithms, such as ►sequential covering.

Cross References

►Learning as Search
►Rule Learning

Recommended Reading

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth International Group.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Greedy Search Approach of Graph Mining

Lawrence Holder
Washington State University, Pullman, USA

Definition

►Greedy search is an efficient and effective strategy for searching an intractably large space when sufficiently informed heuristics are available to guide the search. The space of all subgraphs of a graph is such a space. Therefore, the greedy search approach of ►graph mining uses heuristics to focus the search toward subgraphs of interest while avoiding search in less interesting portions of the space. One such heuristic is based on the compression afforded by a subgraph; that is, how much is the graph compressed if each instance of the subgraph is replaced by a single vertex. Not only does compression focus the search, but it has also been found to prefer subgraphs of interest in a variety of domains.

Motivation and Background

Many data mining and machine learning methods focus on the attributes of entities in the domain, but the relationships between these entities also represents a significant source of information, and ultimately, knowledge. Mining this relational information is an important challenge both in terms of representing the information and facing the additional computational obstacles of analyzing both entity attributes and relations. One efficient way to represent relational information is as a graph, where vertices in the graph represent entities in the domain, and edges in the graph represent attributes and relations among the entities. Thus, mining graphs is an important approach to extracting relational information. The main alternative to a graph-based representation is first-order logic, and the methods for mining this representation fall under the area of inductive logic programming. Here, the focus is on the graph representation.

Several methods have been developed for mining graphs (Washio & Motoda, 2003), but most of these methods focus on finding the most frequent subgraphs in a set of graph transactions (e.g., FSG (Kuramochi & Karypis, 2001), gSpan (Yan & Han, 2002), Gaston (Nijssen & Kok, 2004)) and use efficient exhaustive, rather than heuristic search. However, there are other properties besides frequency of a subgraph pattern that are relevant to many domains. One such property is the amount of compression afforded by the subgraph pattern, when each instance of the pattern is replaced by a single vertex. Searching for the most frequent subgraphs can be made efficient mainly through the exploitation of the downward closure property, which essentially says one can prune any extension of a subgraph that does not meet the minimum support frequency threshold. Unfortunately, the compression of a subgraph does not satisfy the downward closure property; namely, while a small extension of a subgraph may have less compression, a larger extension may have greater compression. Therefore, one cannot easily prune extensions and must search a larger portion of the space of subgraphs. Thus, one must resort to a greedy search method to search this space efficiently.

As with any greedy search approach, the resulting solution may sometimes be suboptimal, that is, the resulting subgraph pattern is not the pattern with maximum compression. The extent to which optimal solutions are missed depends on the type of greedy search and the strength of the heuristics used to guide the search. One approach is embodied in the graph-based induction (GBI) method (Matsuda, Motoda, Yoshida, & Washio, 2002; Yoshida, Motoda, & Indurkhya, 1994). GBI continually compresses the input graph by identifying frequent triples of vertices, some of which may

represent previously compressed portions of the input graph. Candidate triples are evaluated using a measure similar to information gain.

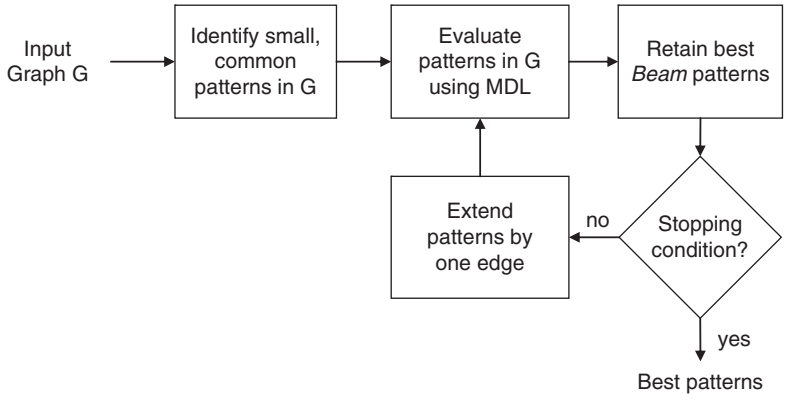
A similar approach recommended here is the use of a beam search strategy coupled with a compression heuristic based on the ►minimum description length (MDL) principle (Rissanen, 1989). The goal is to perform unsupervised discovery of a subgraph pattern that maximizes compression, which is essentially a trade-off between frequency and size. Once the capability to find such a pattern exists, it can be used in an iterative discovery-and-compress fashion to perform hierarchical conceptual clustering, and it can be used to perform supervised learning, that is, find patterns that compress the positive graphs, but not the negative graphs. This approach has been well studied (Cook & Holder, 2000, 2007; Gonzalez, Holder, & Cook, 2002; Holder & Cook, 2003; Jonyer, Cook, & Holder, 2001; Kukluk, Holder, & Cook, 2007) and has proven successful in several domains (Cook, Holder, Su, Maglothin, & Jonyer, 2001; Eberle & Holder, 2006; Holder, Cook, Coble, & Mukherjee, 2005; You, Holder, & Cook, 2006).

Structure of Learning System

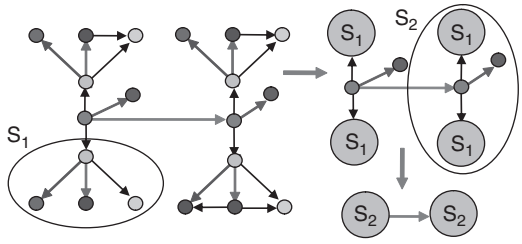
Figure 1 depicts the structure of the greedy search approach of graph mining. The input data is a labeled, directed graph *G*. The search begins by identifying the set of small common patterns in G, that is, all vertices with unique labels having a frequency greater than one. The algorithm then iterates by evaluating the patterns according to the search heuristic, retaining the best patterns, and extending the best patterns by one edge until the stopping condition is met.

The search is guided by the minimum description length (MDL) principle, which seeks to minimize the description length of the entire data set. The evaluation heuristic based on the ►MDL principle assumes that the best pattern is the one that minimizes the description length of the input graph when compressed by the pattern. The description length of the pattern *S* given the input graph *G* is calculated as $DL(G,S) = DL(S) + DL(G|S)$, where *DL(S)* is the description length of the pattern, and $DL(G|S)$ is the description length of the input graph compressed by the pattern. The search seeks a pattern *S* that minimizes *DL(G,S)*.

While several greedy search strategies apply here (e.g., hill climbing, stochastic), the strategy that has been found to work best is the ►beam search. Of the patterns currently under consideration, the system retains only the best *Beam* patterns, where *Beam* is a user-defined parameter. These patterns are then extended by one edge in all possible ways according to the input graph, the extended patterns are evaluated, and then again, all but the best *Beam* patterns are discarded. This process continues until the stopping condition is met. Several stopping conditions are applicable here, including a user-defined limit on the number of patterns considered, the exhaustion of the search space, or the case in which all extensions of a pattern evaluate to a lesser value than their parent pattern. Once meeting the stopping condition, the system returns the best patterns. Note that while the naïve



Greedy Search Approach of Graph Mining. Figure 1. Structure of the greedy search approach of graph mining



Greedy Search Approach of Graph Mining. Figure 2. Example of the greedy search approach of graph mining

approach to implementing this algorithm would require an NP-complete subgraph isomorphism procedure to collect the instances of each pattern, a more efficient approach takes advantage of the fact that new patterns are always one-edge extensions of existing patterns, and, therefore, the instances of the extended patterns can be identified by searching the extensions of the parent's instances. This process does require several isomorphism tests, which is the computational bottleneck of the approach, but avoids the subgraph isomorphism problem.

Once the search terminates, the input graph can be compressed using the best pattern. The compression procedure replaces all instances of the pattern in the input graph by single vertices, which represent the pattern's instances. Incoming and outgoing edges to and from the replaced instances will point to, or originate from the new vertex that represents the instance. The algorithm can then be invoked again on this compressed graph.

Figure 2 illustrates the process on a simple example. The system discovers pattern S_1, which is used to compress the data. A second iteration on the compressed graph discovers pattern S_2. Because instances of a pattern can appear in slightly different forms throughout the data, an inexact graph match, based on graph edit distance, can be used to address noise by identifying similar pattern instances.

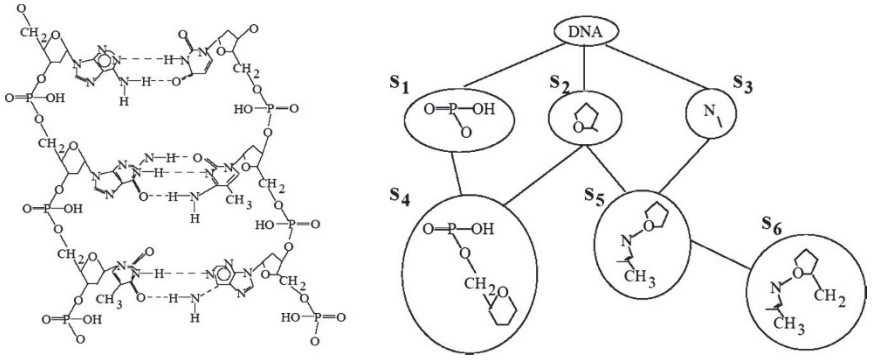
Graph-Based Hierarchical Conceptual Clustering

Given the ability to find a prevalent subgraph pattern in a larger graph and then compress the graph with this pattern, iterating over this process until the graph can no longer be compressed will produce a hierarchical, conceptual clustering of the input data (Jonyer, Cook, & Holder, 2001). On the ith iteration, the best subgraph S_i is used to compress the input graph, introducing new vertices labeled S_i in the graph input to the next iteration. Therefore, any subsequently discovered subgraph S_j can be defined in terms of one or more of S_is, where $i < j$. The result is a *lattice*, where each cluster can be defined in terms of more than one parent subgraph. For example, Fig. 3 shows such a clustering done on a DNA molecule.

Graph-Based Supervised Learning

Extending a graph-based data mining approach to perform ▶supervised learning involves the need to handle negative examples (focusing on the two-class scenario). In the case of a graph the negative information can come in three forms. First, the data may be in the form of numerous smaller graphs, or graph transactions, each labeled either positive or negative. Second, data may be composed of two large graphs: one positive and one negative. Third, the data may be one large graph in which the positive and negative labeling occurs throughout. The first scenario is closest to the standard supervised learning problem in that one has a set of clearly defined examples (Gonzalez et al., 2002). Let G^+ represent the set of positive graphs, and G^- represent the set of negative graphs. Then, one approach to supervised learning is to find a subgraph that appears often in the positive graphs, but not in the negative graphs. This amounts to replacing the information-theoretic measure with simply an error-based measure. This approach will lead the search toward a small subgraph that discriminates well. However, such a subgraph does not necessarily compress well, nor represent a characteristic description of the target concept.

One can bias the search toward a more characteristic description by using the information-theoretic measure to look for a subgraph that compresses the positive examples, but not the negative examples. If $I(G)$ represents the description length (in bits) of the graph G, and $I(G|S)$ represents the description length of graph G compressed by subgraph S, then one can look for an S that minimizes $I(G^+|S) + I(S) + I(G^-) - I(G^-|S)$, where the last two terms represent the portion of the negative graph incorrectly compressed by the subgraph. This approach will lead the search toward a larger subgraph that characterizes the positive examples, but not the negative examples.



Greedy Search Approach of Graph Mining. Figure 3. Iterative application of the greedy search approach of graph mining yields the hierarchical, conceptual clustering on the right given an input graph representing the portion of DNA structure depicted on the left

Finally, this process can be iterated in a set-covering approach to learn a disjunctive hypothesis. If using the error measure, then any positive example containing the learned subgraph would be removed from subsequent iterations. If using the information-theoretic measure, then instances of the learned subgraph in both the positive and negative examples (even multiple instances per example) are compressed to a single vertex. Note that the compression is a lossy one, that is, one does not keep enough information in the compressed graph to know how the instance was connected to the rest of the graph. This approach is consistent with the goal of learning general patterns, rather than mere compression.

Graph Grammar Inference

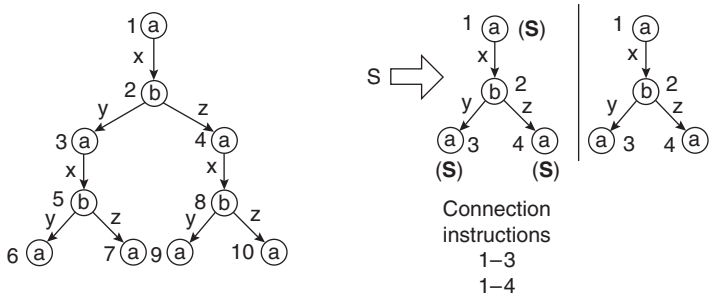
In the above algorithms the patterns are limited to non-recursive structures. In order to learn subgraph motifs, or patterns that can be used as the building blocks to generate arbitrarily large graphs, one needs the ability to learn graph grammars. The key to the inference of a graph grammar is the identification of overlapping structure. One can detect the possibility of a recursive graph-grammar production by checking if the instances of a pattern overlap. If a set of instances overlap by a single vertex, then one can propose a recursive node-replacement graph grammar production. Figure 4 shows an example of a node-replacement graph grammar (right) learned from a simple, repetitive input graph (left). The input graph in Fig. 4 is composed of three overlapping substructures. Based on how the instances overlap, one can also infer connection instructions that describe how the pattern can connect to itself. For example, the connection instructions in Fig. 4 indicate that the graph can grow by connecting vertex 1 of one pattern instance to either vertex 3 or vertex 4 of another pattern instance.

If a set of pattern instances overlap by an edge, then one can propose a recursive edge-replacement graph grammar production. Figure 5 shows an example of an edge-replacement graph grammar (right) learned from the input graph (left). Connection instructions describe how the motifs can connect via the edge labeled "a" or the edge labeled "b."

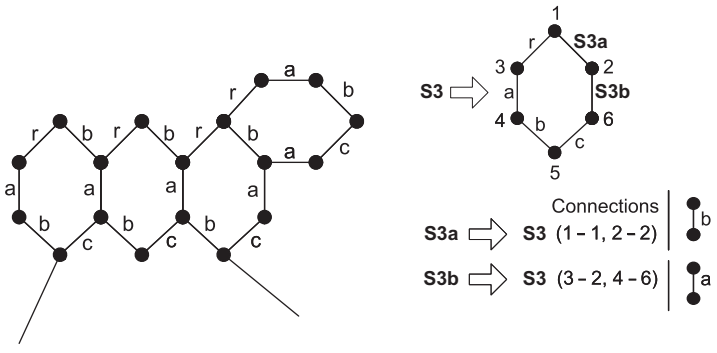
Apart from the inclusion of recursive patterns, the greedy search approach of graph mining is unchanged. Both recursive and non-recursive patterns are evaluated according to their ability to compress the input graph using the ▶MDL heuristic. After several iterations of the approach, the result is a graph grammar consisting of recursive and non-recursive productions that both describe the input graph and provide a mechanism for generating graphs with similar properties.

Programs and Data

Most of the aforementioned functionality has been implemented in the SUBDUE graph-based pattern learning system. The SUBDUE source code and numerous sample graph data files are available at http://www.subdue.org.



Greedy Search Approach of Graph Mining. Figure 4. The node-replacement graph grammar (right) inferred from the input graph (left). The connection instructions indicate how the pattern can connect to itself



Greedy Search Approach of Graph Mining. Figure 5. The edge-replacement graph grammar (right) inferred from the input graph (left). The connection instructions indicate how the pattern can connect to itself

Applications

Many relational domains, from chemical molecules to social networks, are naturally represented as a graph, and a graph mining approach is a natural choice for extracting knowledge from such data. Three such applications are described below.

A huge amount of biological data that has been generated by long-term research encourages one to move one's focus to a systems-level understanding of bio-systems. A biological network, containing various biomolecules and their relationships, is a fundamental way to describe bio-systems. Multi-relational data mining finds the relational patterns in both the entity attributes and relations in the data. A graph consisting of vertices and edges between these vertices is a natural data structure to represent biological networks. The greedy search approach of graph mining has been applied to find patterns in metabolic pathways (You et al., 2006). Graph-based supervised learning finds the unique substructures in a specific type of pathway, which help one understand better how pathways differ. Unsupervised learning shows hierarchical clusters that describe the common substructures in a specific type of pathway, which allow one to better understand the common features in pathways.

Social network analysis is the mapping and measuring of relationships and flows between people, organizations, computers, or other information processing entities. Such analysis is naturally done using a graphical representation of the domain. The greedy approach of graph mining has been applied to distinguish between criminal and legitimate groups based on their mode of communication (Holder et al., 2005). For example, terrorist groups tend to exhibit communications chains; whereas, legitimate groups (e.g., families) tend to exhibit more hub-and-spoke communications.

Anomaly detection is an important problem for detecting fraud or unlawful intrusions. However, anomalies are typically rare and, therefore, present a challenge to most mining algorithms that rely on

regularity and frequency to detect patterns. With the graph mining approach's ability to iteratively compress away regularity in the graph, what is left can be construed as anomalous. To distinguish this residual structure from noise, one can compare its regularity with the probability that such structure would appear randomly. The presence of rare structure that is unlikely to appear by chance suggests an anomaly of interest. Furthermore, most fraudulent activity attempts to disguise itself by mimicking legitimate activity. Therefore, another method for finding such anomalies in graphs is to first find the normative pattern using the greedy search approach of graph mining and then find unexpected deviations to this normative pattern. This approach has been applied to detect anomalies in cargo data (Eberle & Holder, 2006).

Future Directions

One of the main challenges in approaches to graph mining is scalability. Since most relevant graph operations (e.g., graph and subgraph isomorphism) are computationally expensive, they can be applied to only modest-sized graphs that can fit in the main memory. Clearly, there will always be graphs larger than can fit in main memory, so efficient techniques for mining in such graphs are needed. One approach is to keep the graph in a database and translate graph mining operations into database queries. Another approach is to create abstraction hierarchies of large graphs so that mining can occur at higher-level, smaller graphs to identify interesting regions of the graph before descending down into more specific graphs. Traditional high-performance computing techniques of partitioning a problem into subproblems, solving the subproblems, and then recomposing a solution do not always work for graph mining problems, because partitioning the problem means breaking links which may later turn out to be important. New techniques and architectures are needed to improve the scalability of graph mining operations.

Another challenge for graph mining techniques is dynamic graphs. Most graphs represent data that can change over time. For example, a social network can change as people enter and leave the network, new links are established and old links are discarded. First, one would like to be able to mine for static patterns in the presence of the changing data, which will require incremental approaches to graph mining. Second, one would like to mine patterns that describe the evolution of the graph over time, which requires mining of time slice graphs or the stream of graph transaction events. Third, the dynamics can reside in the attributes of entities (e.g., changing concentrations of an enzyme in a metabolic pathway), in the relation structure between entities (e.g., new relationships in a social network), or both. Research is needed on efficient and effective techniques for mining dynamic graphs.

Cross References

►Grammatical Inferences

Recommended Reading

Cook, D., & Holder, L. (March/April 2000). Graph-based data mining. *IEEE Intelligent Systems*, 15(2), 32–41.

Cook, D., & Holder, L. (Eds.). (2007). *Mining graph data*. New Jersey: Wiley.

Cook, D., Holder, L., Su, S., Maglothin, R., & Jonyer, I. (July/August 2001). Structural mining of molecular biology data. *IEEE Engineering in Medicine and Biology, Special Issue on Genomics and Bioinformatics*, 20(4), 67–74.

Eberle, W., & Holder, L. (2006). Detecting anomalies in cargo shipments using graph properties. In *Proceedings of the IEEE intelligence and security informatics conference*, San Diego, CA, May 2006.

Gonzalez, J., Holder, L., & Cook D. (2002). Graph-based relational concept learning. In: *Proceedings of the nineteenth international conference on machine learning*, Sydney, Australia, July 2002.

Holder, L., & Cook, D. (July 2003). Graph-based relational learning: Current and future directions. *ACM SIGKDD Explorations*, 5(1), 90–93.

Holder, L., Cook, D., Coble, J., & Mukherjee, M. (March 2005). Graph-based relational learning with application to security. *Fundamenta Informaticae, Special Issue on Mining Graphs, Trees and Sequences*, 66(1–2), 83–101.

Jonyer, I., Cook, D., & Holder, L. (October 2001). Graph-based hierarchical conceptual clustering. *Journal of Machine Learning Research*, 2, 19–43.

Kukluk, J., Holder, L., & Cook, D. (2007). Inference of node replacement graph grammars. *Intelligent Data Analysis*, 11(4), 377–400.

Kuramochi, M., & Karypis, G. (2001). Frequent subgraph discovery. In *Proceedings of the IEEE international conference on data mining (ICDM)* (pp. 313–320), San Jose, CA.

Matsuda, T., Motoda, H., Yoshida, T., & Washio, T. (2002). Mining patterns from structured data by beam-wise graph-based induction. In *Proceedings of the fifth international conference on discovery science* (pp. 323–338), Lubeck, Germany.

Nijssen, S., & Kok, J. N. (2004). A quickstart in frequent structure mining can make a difference. In *Proceedings of the tenth ACM*

SIGKDD international conference on knowledge discovery and data mining (KDD) (pp. 647–652), Seattle, WA.
Rissanen, J. (1989). *Stochastic complexity in statistical inquiry*. New Jersey: World Scientific.
Washio, T., & Motoda H. (July 2003). State of the art of graph-based data mining. *ACM SIGKDD Explorations*, 5(1), 59–68.
Yan, X., & Han, J. (2002). gSpan: Graph-based substructure pattern mining. In *Proceedings of the IEEE international conference on data mining (ICDM)* (pp. 721–724), Maebashi City, Japan.
Yoshida, K., Motoda, H., & Indurkhya, N. (1994). Graph-based induction as a unified learning framework. *Journal of Applied Intelligence*, 4, 297–328.
You, C., Holder, L., & Cook, D. (2006). Application of graph-based data mining to metabolic pathways. In *Workshop on data mining in bioinformatics, IEEE international conference on data mining*, Hong Kong, China, December 2006.

Group Detection

Hossam Sharara, Lise Getoor
University of Maryland, Maryland, USA

Synonyms

Community detection; Graph clustering; Modularity detection

Definition

Group detection can defined as the clustering of nodes in a graph into groups or communities. This may be a hard partitioning of the nodes, or may allow for overlapping group memberships. A community can be defined as a group of nodes that share dense connections among each other, while being less tightly connected to nodes in different communities in the network. The importance of communities lies in the fact that they can often be closely related to modular units in the system that have a common function, e.g., groups of individuals interacting with each other in a society (Girvan & Newman, 2002), WWW pages related to similar topics (Flake, Lawrence, Giles, & Coetzee, 2002), or proteins having the same biological function within the cell (Chen & Yuan, 2006).

Motivation and Background

The work done in group detection goes back as early as the 1920s when Stuart Rice clustered data by hand to investigate political blocks (Rice, 1927). Another early example is the work of George (Homans, 1950) who illustrated how simple rearrangement of the rows and columns of data matrices helped to reveal their underlying structure. Since then, group detection has attracted researchers from different areas such as sociology, mathematics, physics, marketing, statistics, and computer science.

Group detection techniques vary from simple similarity-based ▶clustering algorithms that follow the classical assumption that the data points are independent and identically distributed, to more advanced techniques that take into consideration the existing relationships between nodes in addition to their attributes, and try to characterize the different distributions present in the data.

Theory Solution

A network is defined as a graph $G = (V, E)$ consisting of a set of nodes $v \in V$, and a set of edges $e \in E$. In the case of weighted networks, $w(v_i, v_j)$ denotes the weight of the edge connection nodes v_i and v_j. A community, or a group, C is a subgraph $C(V', E')$ of the original graph $G(V, E)$ whose nodes and edges are subsets of the original graph's nodes and edges; i.e., $V' \subset V$ and $E' \subset E$.

Following the definition of the community, we can expect that all the vertices in any community must be connected by a path within the same community. This property is referred to in literature as *connectedness*, which implies that in the case of disconnected graphs, we can analyze each connected component separately, as communities cannot span different components.

Another important property that follows from the definition of a community is that the group of vertices within a community should share denser connections among each other, and fewer connections with the other vertices in the network. To quantify this measure, the link density of a group $\delta(C)$ is defined as the ratio between the number of internal edges in that group and the maximum number of possible internal edges:

$$\delta(C) = \frac{|E'|}{|V'| \times (|V'| - 1)/2} \quad (1)$$

Thus, for any community C, we require that $\delta(C) > \delta(G)$; where $\delta(G)$ is the average link density of the whole network. Similarly, the average link density between different communities, calculated using the

ratio between the number of edges emanating from a group and terminating in another, and the maximum number possible of such edges, should generally be low.

Approaches

Beyond the intuitive discussion above, the precise definition of what constitutes a community involves multiple aspects. One important aspect is whether communities form hard partitions of the graph or nodes can belong to several communities. Overlapping communities do commonly occur in natural settings, especially in social networks. Currently, only a few methods are able to handle overlapping communities (Palla, Dernyi, Farkas, & Vicsek, 2005).

Other aspects should also be taken into consideration when defining community structure, such as whether link weights and/or directionalities are utilized, and whether the definition allows for hierarchical community structure, which means that communities may be parts of larger ones. However, one of the most important aspect that comes into consideration in community detection is whether the definition depends on global or local network properties. The main difference between the two approaches is whether the communities are defined in the scope of the whole network structure, such as methods based on centrality measures (Girvan & Newman, 2002), global optimization methods (Newman & Girvan, 2004), spectral methods (Arenas, Daz-Guilera, & Prez-Vicente, 2006), or information-theoretic methods (Rosvall & Bergstrom, 2008). Local methods, on the other hand, define communities based on purely local network structure, such as detecting cliques of different sizes, clique percolation method (Palla et al., 2005), and subgraph fitness method (Lancichinetti, Fortunato, & Kertesz, 2009).

Local Techniques

Local methods for community detection basically rely on defining a set of properties that should exist in a community, then finding maximal subgraphs for which these set of properties hold. This formulation corresponds to finding maximal cliques in the network, where a clique is a subgraph in which all its vertices are directly connected.

However, there are some issues that rises from the previous formulation. First, finding cliques in a graph is an NP-Complete problem, thus most solutions will be approximate based on heuristic methods. Another more semantic issue is the interpretation of communities, especially in the context of social networks, where different individuals have different centralities within their corresponding groups, contradicting with the degree symmetry of the nodes in cliques. To overcome these drawbacks, the notion of a clique is relaxed to n-clique, which is a maximal subgraph where each pair of vertices are at most n-steps apart from each other.

Clustering Techniques

►Data clustering is considered one of the earliest techniques for revealing group structure, where data points are grouped based on the similarity between their corresponding features according to a given similarity measure. The main objective of traditional clustering methods is to obtain clusters or groups of data points possessing high intra-cluster similarity and low inter-cluster similarity. Classical data clustering techniques can be divided into partition-based methods such as k-means clustering (MacQueen, 1967), spectral clustering algorithms (Alpert, Kahng, & Yao, 1999), and hierarchical clustering methods (Hartigan, 1975), which are the most popular and the most commonly used in many fields.

One of the main advantages of the hierarchical clustering techniques is their ability to provide multiple resolutions at which the data can be grouped. In general, hierarchical clustering can be divided into agglomerative and divisive algorithms. The agglomerative algorithm is a greedy bottom-up one that starts with clusters including single data points then successively merge the pairs of clusters with the highest similarity. Divisive algorithms work in an opposite direction, where initially all the data points are regarded as one cluster, which is successively divided into smaller ones by splitting groups of nodes having the lowest similarity. In both algorithms, clusters are represented as a dendrogram, whose depths indicate the steps at which two clusters are joined. This representation clarifies which communities are built up from smaller modules, and how these smaller communities are organized, which can be particularly useful in the case of the presence of a normal hierarchy of community structure in the data. Hierarchical clustering techniques can easily be used in network domains, where data points are replaced by

individual nodes in the network, and the similarity is based on edges between them.

Centrality-Based Techniques

One of the methods for community detection that is based on the global network structure is the one proposed by Girvan and Newman (2002), where they proposed an algorithm based on the betweenness centrality of edges to be able to recover the group structure within the network. Betweenness centrality is a measure of centrality of nodes in networks, defined for each node as the number of shortest paths between pairs of nodes in the network that run through it. The Girvan–Newman algorithm extended this definition for edges in the network as well, where the betweenness centrality of an edge is defined as the number of shortest paths between pairs of nodes that run along it.

The basic idea behind the algorithm is exploiting the fact that the number of edges connecting nodes from different communities is sparse. Following from that, all shortest paths between nodes from different communities should pass along one of these edges, increasing their edge betweenness centrality measure. Therefore, by following a greedy approach and removing edges with highest betweenness centrality from the network successively, the underlying community structure will be revealed. One of the major drawbacks of the algorithm is the time complexity, which is $O(|E|^2|V|)$ generally, and $O(|V|^3)$ for sparse networks. The fact that the edge betweenness needs only to be recalculated only for the edges affected by the edge removal can be factored in, which makes the algorithm efficient in sparse networks with strong community structure, but not very efficient on dense networks.

Modularity-Based Techniques

The concept of modularity was introduced by Newman and Girvan (2004) as a measure to evaluate the quality of a set of extracted communities in a network, and has become one of the most popular quality functions used for community detection. The basic idea is utilizing a *null model*: a network having the same set of nodes as the original one, but with random edges placed between them taking into account preserving the original node degrees. The basic idea is that the created random network is expected to contain no community structure, thus by comparing the number of edges within the extracted communities against the expected number of edges in the same communities from the random network, we can judge the quality of the extracted community structure. More specifically, the modularity Q is defined as follows

$$Q = \frac{1}{2|E|} \sum_{ij} \left[A_{ij} - \frac{deg(i) \times deg(j)}{2|E|} \right] \delta_k(c_i, c_j) \qquad (2)$$

where A_{ij} is the element of the adjacency matrix of the network denoting the number of edges between nodes i and j, $deg(i)$ and $deg(j)$ are the degrees of nodes i and j respectively, c_i and c_j are the communities to which nodes i and j belong respectively, and δ_k refers to the kronecker delta. The summation runs over all pairs of nodes within the same community.

Clearly, a higher modularity value indicates that the average link density within the extracted community is larger than that of the random network where no community structure is present. Thus, modularity maximization can be used as the objective for producing high-quality community structure. However, modularity maximization is an NP-hard problem. Nevertheless, there have been several algorithms for finding fairly good approximations of the modularity maximum in reasonable amount of time.

One of the first modularity maximization algorithms was introduced by Newman in 2004 (Newman, 2004). It is a greedy hierarchical agglomerative clustering algorithm, which starts with individual nodes and merges them in the order of increasing the overall modularity of the resulting configuration. The time complexity of this greedy algorithm is $O(|V|(|E|+|V|))$ or $O(|V|^2)$ for sparse networks, which enables the user to run community detection on large networks in a reasonable amount of time.

Issues

One of the main issues with the methods of group detection in network setting is the focus on the network structure, without taking into consideration other properties of nodes and edges in the network. This issue often results in a lack of correspondence between the extracted communities and the functional groups in the network (Shalizi, Camperi, & Klinkner, 2007). This also

leads to another common problem which is how to validate the resulting communities produced by any of the proposed techniques.

Although in network settings there are often different types of interactions between entities of different natures, most group detection methods work on single-mode networks, which have just a single node and edge type. Fewer works focus on finding groups in more complex, multimodal settings, where nodes from different types have multiple types of interactions with each other. One of the most common approaches to deal with these types of networks is projecting them into a series of individual graphs for each node type. However, this approach results in losing some of the information that could have been retained by operating collectively on the original multi-relational network.

Another issue also gaining interest is developing methods for group detection in dynamic network settings (Tantipathananandh & Berger-Wolf, 2009), where the underlying network structure changes over time. Most of the previous work on group detection focused on static networks, and handles the dynamic case by either analyzing a snapshot of the network at a single point in time, or aggregating all interactions over the whole time period. Both approaches do not capture the dynamics of change in the network structure, which can be an important factor in revealing the underlying communities.

Cross References

►Graph Clustering
►Graph Mining

Recommended Reading

Alpert, C., Kahng, A., & Yao, S. (1999). Spectral partitioning: The more eigenvectors, the better. *Discrete Applied Mathematics, 90*, 3–26.

Arenas, A., Daz-Guilera, A., & Prez-Vicente, C. J. (2006). Synchronization reveals topological scales in complex networks. *Physical Review Letters, 96*(11), 114102.

Chen, J., & Yuan, B. (2006). Detecting functional modules in the yeast protein–protein interaction network. *Bioinformatics, 22*(18), 2283–2290.

Flake, G. W., Lawrence, S., Giles, C. L., & Coetzee, F. (2002). Self-organization and identification of web communities. *IEEE Computer, 35*, 66–71.

Girvan, M., & Newman, M. E. J. (2002). Community structure in social and biological networks. *Proceedings of National Academy of Science, 99*, 7821–7826.

Hartigan, J. A. (1975). *Clustering algorithms*. New York: Wiley.

Homans, G. C. (1950). *The human group*. New York: Harcourt, Brace.

Lancichinetti, A., Fortunato, S., & Kertesz, J. (2009). Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics, 11*, 033015.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of fifth Berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281–297). Berkeley, CA: University of California Press.

Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. *Physical Review E, 69*(6), 066133.

Newman, M. E. J., & Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review E, 69*, 026113.

Palla, G., Dernyi, I., Farkas, I., & Vicsek, T. (2005). Uncovering the overlapping community structure of complex networks in nature and society. *Nature, 435*(7043), 814–818.

Rice, S. A. (1927). The identification of blocs in small political bodies. *American Political Science Review, 21*, 619–627.

Rosvall, M., & Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. *Proceedings of National Academy of Science, 105*, 1118–1123.

Shalizi, C. R., Camperi, M. F., & Klinkner, K. L. (2007). Discovering functional communities in dynamical networks. *Statistical network analysis: Models, issues, and new directions* (pp. 140–157). Berlin: Springer-Verlag.

Tantipathananandh, C., & Berger-Wolf, T. Y. (2009). Algorithms for identifying dynamic communities. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining*, Paris. New York: ACM.

Grouping

►Categorical Data Clustering

Growing Set

Definition

A growing set is a subset of a ►training set containing data that are used by a ►learning system to develop models that are then evaluated against a ►pruning set.

Cross References

►Data Set

Growth Function

►Shattering Coefficient

H

Hebb Rule

►Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Hebbian Learning

Synaptic weight changes depend on the joint activity of the ►presynaptic and postsynaptic neurons.

Cross References

►Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Heuristic Rewards

►Reward Shaping

Hidden Markov Models

Antal van den Bosch
Tilburg University, Tilburg, The Netherlands

Synonyms

HMM

Definition

Hidden Markov models (HMMs) form a class of statistical models in which the system being modeled is assumed to be a Markov process with hidden states. From observed output sequences generated by the Markov process, both the output emission probabilities from the hidden states and the transition probabilities between the hidden states can be estimated by using dynamic programming methods. The estimated model parameters can then be used for various sequence analysis purposes.

Motivation and Background

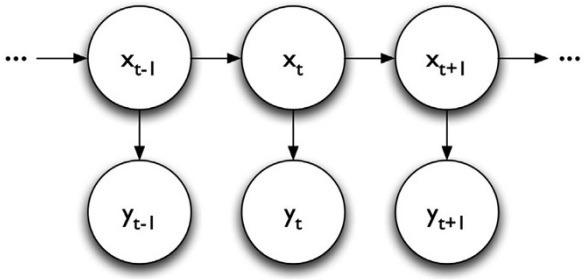
The states of a regular Markov model, named after Russian mathematician Andrey Markov (1865–1922), are directly observable, hence its only parameters are the state transition probabilities. In many real-world cases, however, the states of the system that one wants to model are not directly observable. For instance, in speech recognition the audio is the observable stream, while the goal is to discover the phonemes (the categorical elements of speech) that emitted the audio. HMMs offer the necessary architecture to estimate hidden states through indirect means. Dynamic programming methods have been developed that can estimate both the output emission probabilities and the transition probabilities between the hidden states, either from observations of output sequences only (an unsupervised learning setting), or from pairs of aligned output sequences and gold-standard hidden sequences (a supervised learning setting).

Structure of the Learning System

Figure 1 displays the general architecture of a HMM. Each state (circle) represents a variable x_i or y_i occurring at time i; x_t is the discrete value of the hidden variable at time t. The variable y_t is the output variable observed at the same time t, said to be emitted by x_t. Arrows denote conditional dependencies. Any hidden variable is only dependent on its immediate predecessor; thus, the value of x_t is only dependent on that of x_{t-1} occurring at time $t-1$; this deliberate simplicity is referred to as the Markov assumption. Analogously, observed variables such as y_t are conditionally dependent only on the hidden variables occurring at the same time t, which is x_t in this case.

Typically, a start state x_0 is used as the first hidden state (not conditioned by any previous state), as well as

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,



Hidden Markov Models. Figure 1. Architecture of a hidden Markov model (HMM)

an end state x_{n+1} that closes the hidden state sequence of length n. Start and end states usually emit meta-symbols signifying the "start" and "end" of the sequence.

An important constraint on the data that can, in principle, be modeled in an HMM is that the hidden and output sequences need to be discrete, aligned (i.e., one y_t for each x_t), and of equal length. Sequence pairs that do not conform to these constraints need to be discretized (e.g., in equal-length time slices) or aligned where necessary.

Training and Using Hidden Markov Models

HMMs can be trained both in an unsupervised and a supervised fashion. First, when only observed output sequences are available for training, the model's conditional probabilities from this indirect evidence can be estimated through the Baum–Welch algorithm (Baum, Petrie, Soules, & Weiss, 1970), a form of unsupervised learning, and an instantiation of the expectation-maximization algorithm (Dempster, Laird, & Rubin, 1977).

When instead aligned sequences of gold-standard hidden variables and output variables are given as supervised training data, both the output emission probabilities and the state transition probabilities can be straightforwardly estimated from frequencies of co-occurrence in the training data.

Once trained, it is possible to find the most likely sequence of hidden states that could have generated a particular (test) output sequence using the Viterbi algorithm (Viterbi, 1967).

Applications of Hidden Markov Models

HMMs are known for their successful application in pattern recognition tasks such as speech recognition (Rabiner, 1989) and DNA sequencing (Kulp, Haussler, Reese, & Eeckman, 1996), but also in sequential pattern analysis tasks such as in part-of-speech tagging (Church, 1988).

Their introduction in speech recognition in the 1970s (Jelinek, 1998) led the way toward the introduction of stochastic methods in general in the field of natural language processing in the 1980s and 1990s (Charniak, 1993; Manning & Schütze, 1999) and into text mining and information extraction in the late 1990s and onwards (Freitag & McCallum, 1999). In a similar way, HMMs started to be used in DNA pattern recognition in the mid-1980s, and have gained widespread usage throughout bioinformatics since (Burge & Karlin, 1997; Durbin, Eddy, Krogh, & Mitchison, 1998).

Programs

Many implementations of HMMs exist. Three noteworthy packages are the following:

- UMDHMM by Tapas Kanungo: Implements the forward-backward, Viterbi, and Baum–Welch algorithms (Kanungo, 1999).
- JAHMM by Jean-Marc François: A versatile Java implementation of algorithms related to HMMs (François, 2006).
- HMMER by Sean Eddy: An implementation of profile HMM software for protein sequence analysis (Eddy, 2007).

Cross References

►Baum–Welch Algorithm
►Bayesian Methods
►Expectation-Maximization Algorithm
►Markov Process
►Viterbi Algorithm

Recommended Reading

Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics, 41*(1), 164–171.

Burge, C., & Karlin, S. (1997). Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology, 268*, 78–94.

Charniak, E. (1993). *Statistical language learning*. Cambridge, MA: MIT Press.

Church, K. W. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the second conference on applied natural language processing* (pp. 136–143). Austin, TX.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, *39*(1), 1–38.

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge, UK: Cambridge University Press.

Eddy, S. (2007). *HMMER*. http://hmmer.janelia.org/.

François, J.-M. (2006). *JAHMM*. http://www.run.montefiore.ulg.ac.be/~francois/software/jahmm/.

Freitag, D., & McCallum, A. (1999). Information extraction with HMM structures learned by stochastic optimization. In *Proceedings of the national conference on artificial intelligence* (pp. 584–589). Cambridge, MA: MIT Press.

Jelinek, F. (1998). *Statistical methods for speech recognition*. Cambridge, MA: MIT Press.

Kanungo, T. (1999). UMDHMM: Hidden Markov model toolkit. In A. Kornai (Ed.), *Extended finite state models of language*. Cambridge, UK: Cambridge University Press. URL: http://www.kanungo.us/software/software.html.

Kulp, D., Haussler, D., Reese, M. G., & Eeckman, F. H. (1996). A generalized hidden Markov model for the recognition of human genes in DNA. *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*, 4, 134–142.

Manning, C., & Schütze, H. (1999). *Foundations of statistical natural language processing*. Cambridge, MA: MIT Press.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, *13*(2), 260–269.

Hierarchical Reinforcement Learning

Bernhard Hengst
University of New South Wales
Sydney, Australia

Definition

Hierarchical reinforcement learning (HRL) decomposes a ▶reinforcement learning problem into a hierarchy of subproblems or subtasks such that higher-level parent-tasks invoke lower-level child tasks as if they were primitive actions. A decomposition may have multiple levels of hierarchy. Some or all of the subproblems can themselves be reinforcement learning problems. When a parent-task is formulated as a reinforcement learning problem it is commonly formalized as a semi-Markov decision problem because its actions are child-tasks that persist for an extended period of time. The advantage of hierarchical decomposition is a reduction in computational complexity if the overall problem can be represented more compactly and reusable subtasks learned or provided independently. While the solution to a HRL problem is optimal given the constraints of the hierarchy there are no guarantees in general that the decomposed solution is an optimal solution to the original reinforcement learning problem.

Motivation and Background

Bellman's "curse of dimensionality" beleaguers reinforcement learning because the problem representation grows exponentially in the number of state and action variables. The complexity we encounter in natural environments has a property, near decomposability, that may be exploited using hierarchical models to greatly simplify our understanding and control of behavior. Human societies have used hierarchical organizations to solve complex tasks dating back to at least Egyptian times. It seems natural, therefore, to introduce hierarchical structure into reinforcement learning to solve more complex problems.

When large problems can be decomposed hierarchically there may be improvements in the time and space complexity for both learning and execution of the overall task. Hierarchical decomposition is a divide-and-conquer strategy that solves the smaller subtasks and puts them back together for a more cost-effective solution to the larger problem. The subtasks defined over the larger problem are stochastic macro-operators that execute their policy until termination. If there are multiple ways to terminate a subtask the optimal subtask policy will depend on the context in which the subtask is invoked. Subtask policies usually persist for multiple time-steps and are hence referred to as *temporally extended actions*. Temporally extended actions have the potential to transition through a much smaller "higher-level" state-space, reducing the size of the original problem. For example, navigating through a house may only require room states to represent the abstract problem if room-leaving temporally extended actions are available to move through each room. A room state in this example is referred to as an *abstract state* as the

detail of the exact position in the room is abstracted away. Hierarchical reinforcement learning can also provide opportunities for subtask reuse. If the rooms are similar, the policy to leave a room will only need to be learnt once and can be transferred and reused.

Early developments of hierarchical learning appeal to analogies of boss – subordinate models. Ashby (1956) discusses the "amplification" of regulation in very large systems through hierarchical control – a doctor looks after a set of mechanics who in turn maintain thousands of air-conditioning systems. Watkins (1989) used a navigator – helmsman hierarchical control example to illustrate how reinforcement learning limitations may be overcome. Early examples of hierarchical reinforcement learning include Singh's Hierarchical-DYNA (Dyna, a class of architectures for intelligent systems based on approximating dynamic programming methods. Dyna architectures integrate trial-and-error (reinforcement) learning and execution-time planning into a single process operating alternately on the world and on a learned model of the world (Sutton 1990)) (Singh, 1992), Kaelbling's Hierarchical Distance to Goal (HDG) (Kaelbling, 1993), and Dayan and Hinton's Feudal reinforcement learning (Dayan & Hinton, 1992). The latter explains hierarchical structure in terms of a management hierarchy. The example has four hierarchical levels and employs abstract states, which they refer to as "information hiding".

Close to the turn of the last century three approaches to hierarchical reinforcement learning were developed relatively independently: Hierarchies of Abstract Machines (HAMs) (Parr & Russell, 1997); the Options framework (Sutton, Precup, & Singh, 1999); and MAXQ value function decomposition (Dietterich, 2000). Each approach has different emphases, but a common factor is the use of temporally extended actions and the formalization of HRL in terms of semi-Markov decision process theory (Puterman, 1994) to solve the higher-level abstract reinforcement learning problem.

Hierarchical reinforcement learning is still an active research area. More recent extensions include: continuous state-space; concurrent actions and multi-agency; use of average rewards (Ghavamzadeh & Mahadevan, 2002); continuing problems; policy-gradient methods; partial-observability and hierarchical memory; factored state-spaces and graphical models; and basis functions. Hierarchical reinforcement learning also includes hybrid approaches such as Ryan's reinforcement learning teleo-operators (RL-TOPs) (Ryan & Reid, 2000) that combines planning at the top level and reinforcement learning at the more stochastic lower levels. Please see Barto and Mahadevan (2003) for a survey of recent advances in hierarchical reinforcement learning. More details can be found in the section on recommended reading.

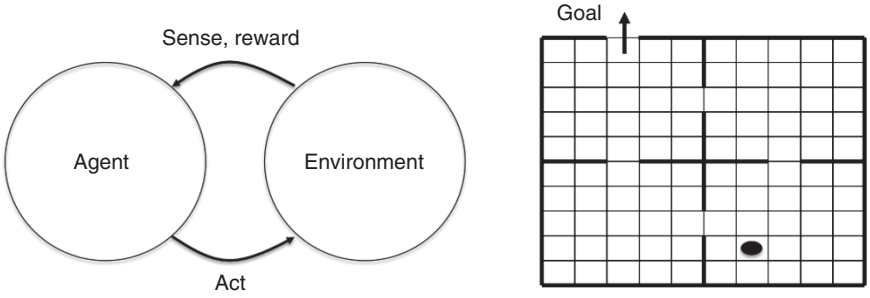
In most applications the structure of the hierarchy is provided as background knowledge by the designer. Some researchers have tried to learn the hierarchical structure from the agent–environment interaction. Most approaches look for subgoals or subtasks that try to partition the problem into near independent reusable subproblems.

Structure of Learning System

Structure of HRL

The agent view of reinforcement learning illustrated on the left in Fig. 1 shows an agent interacting with an environment. At regular time-steps the agent takes actions in the environment and receives sensor observations and rewards from the environment. A hierarchical reinforcement learning agent is given or discovers background knowledge that explicitly or implicitly provides a decomposition of the environment. The agent exploits this knowledge to solve the problem more efficiently by finding an action policy to optimize a measure of future reward, as for reinforcement learning.

We will motivate the machinery of hierarchical reinforcement learning with the simple example shown in Fig. 1 (right). This diagram shows a four-room house with doorways between adjoining rooms and a doorway in the top left room leading outside. Each cell represents a possible position of the agent. We assume the agent always starts in the bottom left room position as shown by the black oval. It is able to sense its position in the room and which room it occupies. It can move one step in any of the four compass directions each time-step. It also receives a reward of −1 at each time-step. The objective is to leave the house via the least-cost route. We assume that the actions are stochastic with an 80% chance of moving in the intended direction and a 20% chance of staying in place. Solving this problem in a straightforward manner using reinforcement learning requires storage for 400 Q values defined over 100 states and 4 actions.



Hierarchical Reinforcement Learning. Figure 1. Left: The agent view of reinforcement learning. Right: A four-room environment with the agent in one of the rooms show as a solid black oval

If the state space is decomposed into the four identical rooms a hierarchical reinforcement learner could solve this problem more efficiently. For example, we could solve two subproblems. One that finds an optimal solution to leave a room to the North and another to leave a room to the West. When learning these subtasks, leaving a room in any other way is disallowed. Each of these subproblems requires storage for 100 Q values – 25 states and 4 actions.

We also formulate and solve a higher-level problem that consists of only the four rooms as states. These are abstract states because, as previously explained, the exact position in the room has been abstracted away. In each abstract state we allow a choice of only one or the other of the learnt room-leaving actions. These are temporally extended actions because, once invoked, they will usually persist for multiple time-steps until the agent exits the room. We proceed to solve this higher-level problem in the usual way using reinforcement learning. The proviso is that the reward on completing a temporally extended action is the sum of rewards accumulated since invocation of the subtask. The higher-level problem requires storage for only 8 Q values – 4 states and 2 actions.

Once learnt, execution of the higher-level policy will determine the optimal room-leaving action to invoke given the current room – in this case to leave the room via the West doorway. Control is passed to the room-leaving subtask that leads the agent out of the room through the chosen doorway. Upon leaving the room, the subtask is terminated and control is passed back to the higher level that chooses the next optimal room-leaving action until the agent finally leaves the house. The total number of Q values required for the hierarchical reinforcement formulation is 200 for the two subtasks and eight for the higher-level problem, a total of 208. This almost halves the storage requirements compared to the "flat" formulation with corresponding savings in time complexity. In this example, hierarchical reinforcement learning finds the same optimal policy that a less efficient reinforcement learner would find, but this is not always the case.

The above example hides many issues that hierarchical reinforcement learning needs to address, including: safe state abstraction; appropriately accounting for accumulated subtask reward when initial conditions change or rewards are discounted; optimality of the solution; and learning of the hierarchical structure itself. In the next sections we will touch on these issues as we discuss the semi-Markov decision problem formalism and review several approaches to hierarchical reinforcement learning.

Semi-Markov Decision Problem Formalism

The common underlying formalism in hierarchical reinforcement learning is the semi-Markov decision process (SMDP). A SMDP generalizes a ▶Markov decision process by allowing actions to be temporally extended. We will state the discrete time equations following Dietterich (2000), recognizing that in general SMDPs are formulated with real-time valued temporally extended actions (Puterman, 1994).

Denoting the random variable N to be the number of time steps that a temporally extended action a takes to complete when it is executed starting in state s, the state transition probability function for the result state s' and the expected reward function are given by (1) and

(2) respectively.

$$T_{ss'}^{N,a} = Pr\{s_{t+N} = s' | s_t = s, a_t = a\} \quad (1)$$

$$R_{ss'}^{N,a} = E\left\{\sum_{n=1}^{N} \gamma^{n-1} r_{t+n} | s_t = s, a_t = a, s_{t+N} = s'\right\} \quad (2)$$

$R_{ss'}^{N,a}$ is the expected sum of N future discounted rewards. The discount factor $\gamma \in [0,1]$. When set to less than 1, γ insures that the value function will converge for continuing or infinite-horizon problems. The Bellman "backup" equations for the value function $V(s)$ for an arbitrary policy π and optimal policies (denoted by $*$) are similar to those for MDPs with the sum taken with respect to s' and N.

$$V_m^{\pi}(s) = \sum_{s',N} T_{ss'}^{N,\pi(s)} \left[R_{ss'}^{N,\pi(s)} + \gamma^N V_m^{\pi}(s')\right] \quad (3)$$

$$V_m^{*}(s) = \max_a \sum_{s',N} T_{ss'}^{N,a} \left[R_{ss'}^{N,a} + \gamma^N V_m^{*}(s')\right] \quad (4)$$

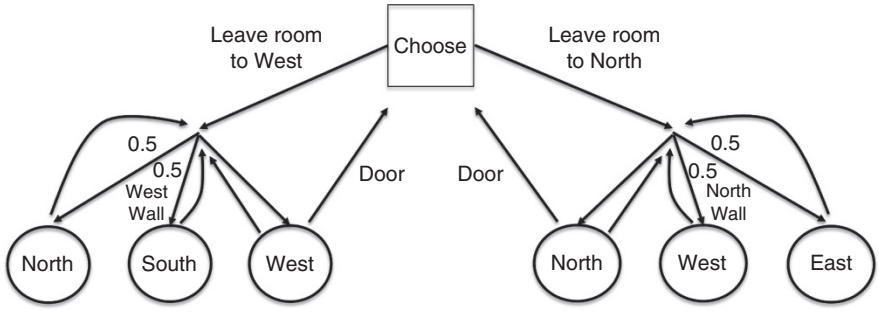
For problems that are guaranteed to terminate, the discount factor γ can be set to 1. In this case the number of steps N can be marginalized out and the sum taken with respect to s alone. The above equations are then similar to the ones for MDPs with the expected primitive reward replaced with the expected sum of rewards to termination of the temporally extended action. All the methods developed for reinforcement learning using primitive actions work equally well for problems using temporally extended actions.

Approaches to Hierarchical Reinforcement Learning

Hierarchies of Abstract Machines (HAMs) In the HAM approach to hierarchical reinforcement learning (Parr & Russell, 1997), the designer specifies subtasks by providing stochastic finite state automata called abstract machines. While in practice several abstract machines may allow some to call others as subroutines (hence hierarchies of abstract machines), in principle this is equivalent to specifying one large abstract machine with two types of states. Action states, that specify the action to be taken given the state of the MDP to be solved and choice states with nondeterministic actions.

An abstract machine is a triple $\langle \mu, I, \delta \rangle$, where μ is a finite set of machine states, I is a stochastic function from states of the MDP to be solved to machine states that determines the initial machine state, and δ is a stochastic next-state function mapping machine states and MDP states to next machine states. The parallel action of the MDP and an abstract machine yields a discrete-time higher-level SMDP with the abstract machine's action states generating a sequence of temporally extended actions between choice states. Only a subset of states of the original MDP are associated with choice-points, potentially reducing the higher-level problem significantly.

Continuing with our four-room example, the abstract machine in Fig. 2 provides choices for leaving a room to the West or the North. In each room it will take actions that move the agent to a wall, and perform a random walk along the wall until it finds the doorway. Only five states of the original MDP are states of the higher-level SMDP. These states are the initial state of the agent and the states on the other side of doorways where the abstract machine enters choice states. Reinforcement learning methods update the value function for these five states in the usual way with rewards accumulated since the last choice state.



Hierarchical Reinforcement Learning. Figure 2. An abstract machine for a HAM that provides routines for leaving rooms to the West and North of the house in Fig. 1 right

The optimal policy consists of the three temporally extended actions sequentially leaving a room to the West, North, and North again.

Solving the SMDP will yield an optimal policy for the agent to leave the house subject to the constraints of the abstract machine. In this case it is not a globally optimal policy because a random walk along walls to find a doorway is inefficient. The HAM approach is predicated on engineers and control theorists being able to design good controllers that will realize specific lower level behaviors. HAMs are a way to partially specify procedural knowledge to transform an MDP to a reduced SMDP. In the most general case HAMs can be Turing machines that execute any computable mapping of the agent's complete sensory-action history.

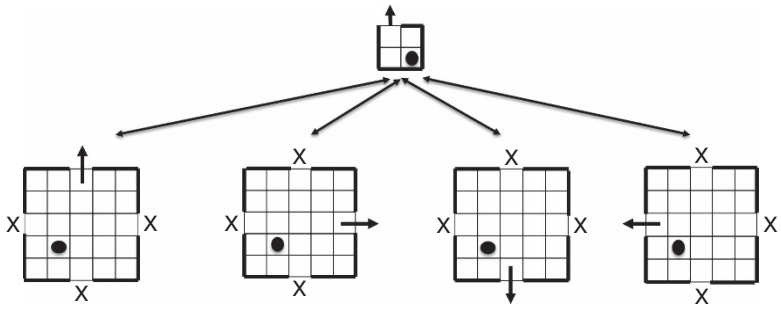
Options For an MDP with finite states S and actions A, *options* generalize one-step primitive actions to include temporally extended actions (Sutton et al., 1999). Options consist of three components: a policy $\pi : S \times A \rightarrow [0,1]$, a termination condition $\beta : S \rightarrow [0,1]$, and an initiation set $I \subseteq S$. An option $\langle I, \pi, \beta \rangle$ is available in state s if and only if $s \in I$. If an option is invoked, actions are selected according to π until the option terminates according to β. These options are called Markov options because intra-option actions taken by policy π depend only on the current state s. It is possible to generalize options to semi-Markov options in which policies and termination conditions make their choices dependent on all prior events since the option was initiated. In this way it is possible, for example, to "time-out" options after some period of time has expired. For their most general interpretation, options and HAMs appear to have similar functionality, but different emphases.

Options were intended to augment the primitive actions available to an MDP. The temporally extended actions executed by the options yield a SMDP. As for HAMs, if options replace primitive actions, the SMDP can be considerably reduced. There is debate as to benefits when primitive actions are retained. Reinforcement learning may be accelerated because the value function can be backed-up over greater distances in the state-space and the inclusion of primitive actions guarantees convergence to the globally optimal policy, but the introduction of additional actions increased the storage and exploration necessary.

In a similar four-room example to that of Fig. 1, the authors (Sutton et al., 1999) show how options can learn significantly faster proceeding on a room-by-room basis, rather than position by position. When the goal is not in a convenient location, able to be reached by the given options, it is possible to include primitive actions as special-case options and still accelerate learning for some problems. For example, with room-leaving options alone, it is not possible to reach a goal in the middle of a room. Primitive actions are required when the room containing the goal state is entered. Although the inclusion of primitive actions guarantees convergence to the globally optimal policy, this may create extra work for the learner.

MAXQ The MAXQ (Dietterich, 2000) approach to hierarchical reinforcement learning restricts subtasks to subsets of states, actions, and policy fragments of the original MDP without introducing extra state, as is possible with HAMs and semi-Markov options. The contribution of MAXQ is the decomposition of the value function over the hierarchy and provision of opportunities for state abstraction. An MDP is manually decomposed into a hierarchical directed acyclic graph of subtasks called a task-hierarchy. Each subtask is a smaller (semi-)MDP. In decomposing the MDP the designer specifies the active states and terminal states for each subtask. Terminal states are typically classed either as goal terminal states or non-goal terminal states. Using disincentives for non-goal terminal states, policies are learned for each subtask to encourage them to terminate in goal terminal states. The actions available in each subtask can be primitive actions or other (child) subtasks. Each sub-task can invoke any of its child subtasks as a temporally extended action. When a task enters a terminal state, it, and all its children, abort and return control to the calling subtask.

Figure 3 shows a task-hierarchy for the previous four-room problem. The four lower-level subtasks are sub-MDPs for a generic room, where a separate policy is learnt to exit a room by each of the four possible doorways. The arrow indicates a transition to a goal terminal state and the "×"s indicate non-goal terminal states. States, actions, transitions, and rewards are inherited from the original MDP. The rewards on transition to terminal states are engineered to encourage the agent to avoid non-goal terminal states and terminate in goal



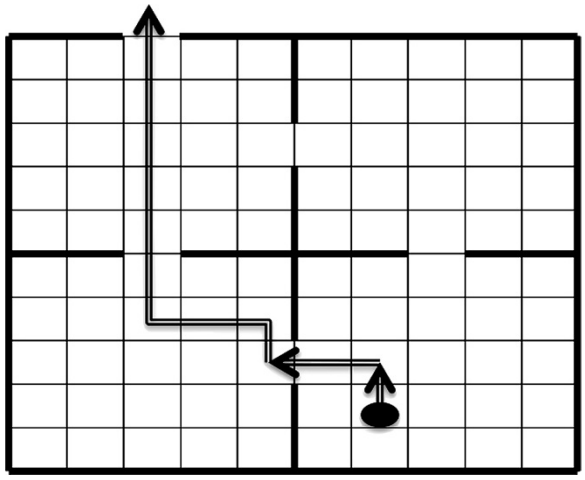
Hierarchical Reinforcement Learning. Figure 3. A task-hierarchy decomposing the four-room problem in Fig. 1. The four lower-level subtasks are generic room-leaving sub-MDPs, one for leaving a room in each compass direction

states. The higher-level problem (SMDP) consists of just four states representing the rooms. Any of the subtasks (room-leaving actions) can be invoked in any of the rooms.

A key feature of MAXQ is that it represents the value of a state as a decomposed sum of subtask completion values plus the value of the immediate primitive action. A completion value is the expected (discounted) cumulative reward to complete the subtask after taking the next (temporally extended) action when following a policy over subtasks. The sum includes all the tasks invoked on the path from the root task in the task hierarchy right down to the primitive action. For a rigorous mathematical treatment the reader is referred to Dietterich (2000). The Q function is expressed recursively (5) as the value for completing the subtask plus the completion value for the overall problem after the subtask has terminated. In this equation, i is the subtask identifier, s is the current state, action a is the child subtask (or primitive action), and π is a policy for each subtask.

$$Q^{\pi}(i,s,a) = V^{\pi}(a,s) + C^{\pi}(i,s,a) \qquad (5)$$

We describe the basic idea for the task-hierarchy shown in Fig. 3 for the optimal policy. The value of the agent's state has three components determined by the two levels in the task-hierarchy plus a primitive action. For the agent state, shown in Fig. 4 by a solid black oval, the value function represents the expected reward for taking the next primitive action to the North, completing the lower-level subtask of leaving the room to the West, and completing the higher-level task of leaving the house. The benefit of decomposing the value function is that it can be represented much more compactly because only the completion values for non-primitive subtasks and primitive actions need be stored.



Hierarchical Reinforcement Learning. Figure 4. The components of the decomposed value function for the agent following an optimal policy for the the four-room problem in Fig. 1. The agent is shown as a solid black oval at the starting state

The example illustrates two types of state abstraction. As all the rooms are similar we can ignore the room identity when we learn intra-room navigation policies. Secondly, when future rewards are not discounted, the completion value after leaving a room is independent of the starting state in that room. These "funnel" actions allow the intra-room states to be abstracted into a single state for each room as far as the completion value is concerned. The effect is that the original problem can be decomposed into a small four-state SMDP at the top level and four smaller subtask MDPs.

Optimality Hierarchical reinforcement learning can at best yield solutions that are hierarchically optimal,

assuming convergence conditions are met, meaning that they are consistent with the task-hierarchy. MAXQ introduces another form of optimality – recursive optimality. MAXQ optimizes subtask policies to reach goal states ignoring the needs of their parent tasks. This has the advantage that subtasks can be reused in various contexts, but they may not therefore be optimal in each situation. A recursively optimal solution cannot be better than a hierarchical optimal solution. Both recursive and hierarchical optimality can be arbitrarily worse than the globally optimal solution if a designer chooses a poor HAM, option or hierarchical decomposition.

The stochastic nature of MDPs means that the condition under which a temporally abstract action is appropriate may have changed after the action's invocation and that another action may become a better choice because of "stochastic drift." A subtask policy proceeding to termination in this situation may be suboptimal. By constantly interrupting the subtask, as for example in HDG (Kaelbling, 1993), a better subtask may be chosen. Dietterich calls this "polling" procedure hierarchical greedy execution. While this is guaranteed to be no worse than the hierarchically optimal or recursively optimal solution and may be considerably better, it still does not provide any global optimality guarantees. Great care is required while learning with hierarchical greedy execution. Hauskrecht, Meuleau, and Kaelbling (1998) discuss decomposition and solution techniques that make optimality guarantees, but unfortunately, unless the MDP can be decomposed into very weakly coupled smaller MDPs, the computational complexity is not necessarily reduced. Benefits will still accrue if the options or subtask policies can be reused and amortized over multiple MDPs.

Automatic Decomposition In the above approaches the programmer is expected to manually decompose the overall problem into a hierarchy of subtasks. Methods to automatically decompose problems include ones that look for subgoal bottleneck or landmark states, and ones that find common behavior trajectories or region policies. For example, in Fig. 1 the agent will exit one of the two starting room doorways on the way to the goal. The states adjacent to each doorway will be visited more frequently in successful trials than other states.

Both NQL (nested Q learning) (Digney, 1998) and McGovern (2002) use this idea to identify subgoals. Moore, Baird, and Kaelbling (1999) suggest that, for some navigation tasks, performance is insensitive to the position of landmarks and an (automatic) randomly generated set of landmarks does not show widely varying results from more purposefully positioned ones. Hengst has explored automatic learning of MAXQ-like task-hierarchies from the agent's interactive experience with the environment, automatically finding common regions and generating subgoals when the agent's prediction fails. Methods include state abstraction with discounting for infinite horizon problems and decompositions of problems to form partial-order task-hierarchies (Hengst, 2008). When there are no cycles in the causal graph the variable influence structure analysis (VISA) algorithm (Jonsson & Barto, 2006) performs hierarchical decomposition of factored Markov decision processes using a given dynamic Bayesian network model of actions. Konidaris and Barto (2009) introduce a skill discovery method for reinforcement learning in continuous domains that constructs chains of skills leading to an end-of-task reward.

Given space limitations we cannot adequately cover all the research in hierarchical reinforcement learning, but we trust that the material above will provide a starting point.

Cross References

- ▶Associative Reinforcement Learning
- ▶Average Reward Reinforcement Learning
- ▶Bayesian Reinforcement Learning
- ▶Credit Assignment
- ▶Markov Decision Process
- ▶Model-Based Reinforcement Learning
- ▶Policy Gradient Methods
- ▶Q Learning
- ▶Reinforcement Learning
- ▶Relational Reinforcement Learning
- ▶Structured Induction
- ▶Temporal Difference Learning

Recommended Reading

Ashby, R. (1956). *Introduction to Cybernetics*. London: Chapman & Hall.

Barto, A., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Special Issue on Reinforcement Learning, Discrete Event Systems Journal, 13*, 41–77.

Dayan, P., & Hinton, G. E. (1992). Feudal reinforcement learning. In *Advances in neural information processing systems 5 NIPS*

Conference, Denver, CO, December 2–5, 1991. San Francisco: Morgan Kaufmann.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research, 13*, 227–303.

Digney, B. L. (1998). Learning hierarchical control structures for multiple tasks and changing environments. In *From animals to animats 5: Proceedings of the fifth international conference on simulation of adaptive behaviour. SAB 98* Zurich, Switzerland, August 17–21, 1998. Cambridge: MIT Press.

Ghavamzadeh, M., & Mahadevan, S. (2002). Hierarchically optimal average reward reinforcement learning. In C. Sammut & Achim Hoffmann (Eds.), *Proceedings of the nineteenth international conference on machine learning*, Sydney, Australia (pp. 195–202). San Francisco: Morgan-Kaufman.

Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Hierarchical solution of Markov decision processes using macro-actions. In *Fourteenth annual conference on uncertainty in artificial intelligence*, Madison, WI (pp. 220–229).

Hengst, B. (2008). Partial order hierarchical reinforcement learning. In *Australasian conference on artificial intelligence* Auckland, New Zealand, December 2008 (pp. 138–149). Berlin: Springer.

Jonsson, A., & Barto, A. (2006). Causal graph based decomposition of factored MDPs. *Journal of Machine Learning Research, 7*, 2259–2301.

Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. In *Machine learning: Proceedings of the tenth international conference* (pp. 167–173). San Mateo: Morgan Kaufmann.

Konidaris, G., & Barto, A. (2009). Skill discovery in continuous reinforcement learning domains using skill chaining. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, & A. Culotta (Eds.), *Advances in neural information processing systems 22* (pp. 1015–1023).

McGovern, A. (2002). Autonomous discovery of abstractions through interaction with an environment. In *SARA* (pp. 338–339). London: Springer.

Moore, A., Baird, L., & Kaelbling, L. P. (1999). Multi-value functions: Efficient automatic action hierarchies for multiple goal MDPs. In *Proceedings of the international joint conference on artificial intelligence*, Stockholm (pp. 1316–1323). San Francisco: Morgan Kaufmann.

Parr, R., & Russell, S. J. (1997). Reinforcement learning with hierarchies of machines. In *NIPS*, Denver, CO, 1997.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. Wiley: New York.

Ryan, M. R. K., & Reid, M. D. (2000). Using ILP to improve planning in hierarchical reinforcement learning. In *Proceedings of the tenth international conference on inductive logic programming*, ILP 2000, London. London: Springer.

Singh, S. (1992). Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the tenth national conference on artificial intelligence*.

Sutton, R. S., Precup, D., & Singh, S. P. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence, 112*(1–2), 181–211.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College.

High-Dimensional Clustering

►Document Clustering

Higher-Order Logic

John Lloyd
The Australian National University
Canberra ACT, Australia

Definition

Higher-order logic is a logic that admits so-called "higher-order functions," which are functions that can have functions as arguments or return a function as a result. The expressive power that comes from higher-order functions makes the logic highly suitable for representing individuals, predicates, features, background theories, and hypotheses, and performing the necessary reasoning, in machine learning applications.

Motivation and Background

Machine learning tasks naturally require knowledge representation and reasoning. The individuals that are the subject of learning, the training examples, the features, the background theory, and the hypothesis languages all have to be represented. Furthermore, reasoning, usually in the form of computation, has to be performed.

Logic is a convenient formalism in which knowledge representation and reasoning can be carried out; indeed, it was developed exactly for this purpose. For machine learning applications, quantification over variables is generally needed, so that, at a minimum, ►first-order logic should be used. Here, the use of higher-order logic for this task is outlined. Higher-order logic admits higher-order functions that can have functions as arguments or return a function as a result. This means that the expressive power of higher-order logic is greater than first-order logic so that some expressions of higher-order logic are difficult or impossible to state directly in first-order logic. For example, sets can be represented by ►predicates which are terms in higher-order logic and operations on sets can be implemented by higher-order functions. Grammars that generate

spaces of predicates can be easily expressed. Also the programming idioms of functional programming languages become available.

The use of higher-order logic in learning applications began around 1990 when researchers argued for the advantages of lifting the concept of ▶least general generalization in the first-order setting to the higher-order setting (Dietzen & Pfenning, 1992; Feng & Muggleton, 1992; Lu, Harao, & Hagiya, 1998). A few years later, Muggleton and Page (1994) advocated the use of higher-order concepts, especially sets, for learning applications. Then the advantages of a type system and also higher-order facilities for concept learning were presented in Flach, Giraud-Carrier, and Lloyd (1998). Higher-order logic is also widely used in other parts of computer science, for example, theoretical computer science, functional programming, and verification of software.

Most treatments of higher-order logic can be traced back to Church's simple theory of types (Church, 1940). Recent accounts can be found, for example, in Andrews (2002), Fitting (2002), and Wolfram (1993). For a highly readable account of the advantages of working in higher-order rather than first-order logic, Farmer (2008) is strongly recommended. An account of higher-order logic specifically intended for learning applications is in Lloyd (2003), which contains much more detail about the knowledge representation and reasoning issues that are discussed below.

Theory

Logic

To begin, here is one formulation of the syntax of higher-order logic which gives prominence to a type system that is useful for machine learning applications, in particular.

An *alphabet* consists of four sets: a set $\mathfrak{T}$ of type constructors; a set $\mathfrak{P}$ of parameters; a set $\mathfrak{C}$ of constants; and a set $\mathfrak{V}$ of variables. Each type constructor in $\mathfrak{T}$ has an arity. The set $\mathfrak{T}$ always includes the type constructor Ω of arity 0. Ω is the type of the booleans. Each constant in $\mathfrak{C}$ has a signature (i.e., type declaration). The set $\mathfrak{V}$ is denumerable. Variables are typically denoted by $x, y, z, \ldots$. The parameters are type variables that provide polymorphism in the logic; they are ignored for the moment.

Here is the definition of a type (for the nonpolymorphic case).

Definition A *type* is defined inductively as follows:

1. If T is a type constructor of arity k and $\alpha_1, \ldots, \alpha_k$ are types, then $T\ \alpha_1 \ldots \alpha_k$ is a type. (Thus a type constructor of arity 0 is a type.)
2. If α and β are types, then $\alpha \to \beta$ is a type.
3. If $\alpha_1, \ldots, \alpha_n$ are types, then $\alpha_1 \times \cdots \times \alpha_n$ is a type.

The set $\mathfrak{C}$ always includes the following constants:

1. $\top$ and $\bot$, having signature Ω.
2. $=_\alpha$, having signature $\alpha \to \alpha \to \Omega$, for each type α.
3. $\neg$, having signature $\Omega \to \Omega$.
4. $\wedge, \vee, \longrightarrow, \longleftarrow$, and $\longleftrightarrow$, having signature $\Omega \to \Omega \to \Omega$.
5. Σ_α and Π_α, having signature $(\alpha \to \Omega) \to \Omega$, for each type α.

The intended meaning of $=_\alpha$ is identity (i.e., $=_\alpha\ x\ y$ is $\top$ if x and y are identical), the intended meaning of $\top$ is true, the intended meaning of $\bot$ is false, and the intended meanings of the connectives $\neg, \wedge, \vee, \longrightarrow, \longleftarrow$, and $\longleftrightarrow$ are as usual. The intended meanings of Σ_α and Π_α are that Σ_α maps a predicate to $\top$ if the predicate maps at least one element to $\top$ and Π_α maps a predicate to $\top$ iff the predicate maps all elements to $\top$.

Here is the definition of a term (for the nonpolymorphic case).

Definition A *term*, together with its type, is defined inductively as follows:

1. A variable in $\mathfrak{V}$ of type α is a term of type α.
2. A constant in $\mathfrak{C}$ having signature α is a term of type α.
3. If t is a term of type β and x a variable of type α, then $\lambda x.t$ is a term of type $\alpha \to \beta$.
4. If s is a term of type $\alpha \to \beta$ and t a term of type α, then $(s\,t)$ is a term of type β.
5. If $t_1, \ldots, t_n$ are terms of type $\alpha_1, \ldots, \alpha_n$, respectively, then $(t_1, \ldots, t_n)$ is a term of type $\alpha_1 \times \cdots \times \alpha_n$.

A *formula* is a term of type Ω. Terms of the form $(\Sigma_\alpha\ \lambda x.t)$ are written as $\exists_\alpha x.t$ and terms of the form $(\Pi_\alpha\ \lambda x.t)$ are written as $\forall_\alpha x.t$ (in accord with the

intended meaning of Σ_α and Π_α). Thus, in higher-order logic, each quantifier is obtained as a combination of an abstraction acted on by a suitable function (Σ_α or Π_α).

The polymorphic version of the logic extends what is given above by also having available parameters. The definition of a type as above is then extended to polymorphic types that may contain parameters and the definition of a term as above is extended to terms that may have polymorphic types.

Reasoning in higher-order logic can consist of theorem proving, via resolution or tableaus, for example, or can consist of equational reasoning, as is embodied in the computational mechanisms of functional programming languages, for example. Theorem proving and equational reasoning can even be combined to produce more flexible reasoning systems. Determining whether a formula is a theorem is, of course, undecidable.

The semantics for higher-order logic is generally based on Henkin (1950) models. Compared with first-order interpretations, the main extra ingredient is that, for each (closed) type of the form $\alpha \to \beta$, there is a domain that consists of some set of functions from the domain corresponding to α to the domain corresponding to β. There exist proof procedures that are sound and complete with respect to this semantics (Andrews, 2002; Fitting, 2002).

The logic includes the λ-calculus. Thus, the rules of λ-conversion are available:

1. (α-Conversion) $\lambda x.t \succ_\alpha \lambda y.(t\{x/y\})$, if y is not free in t.
2. (β-Reduction) $(\lambda x.s\,t) \succ_\beta s\{x/t\}$.
3. (η-Reduction) $\lambda x.(t\,x) \succ_\eta t$, if x is not free in t.

Here $s\{x/t\}$ denotes the result of replacing free occurrences of x in s by t, where free variable capture is avoided by renaming the relevant bound variables in s.

Higher-order generalization is introduced through the concept of least general generalization as follows (Feng & Muggleton, 1992). A term s is *more general* than a term t if there is a substitution θ such that $s\theta$ is λ-convertible to t. A term t is a *common generalization* of a set T of terms if t is more general than each of the terms in T. A term t is a *least general generalization* of a set T of terms if t is a common generalization of T and, for all common generalizations s of T, t is not strictly more general than s.

Knowledge Representation

In machine learning applications, the individuals that are the subject of learning need to be represented. Using logic, individuals are most naturally represented by (closed) terms. In higher-order logic, advantage can be taken of the fact that sets can be identified with predicates (their characteristic functions). Thus, the set $\{1, 2\}$ is the term

$$\lambda x.\mathit{if}\ x = 1\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = 2\ \mathit{then}\ \top\ \mathit{else}\ \bot.$$

This idea generalizes to multisets and similar abstractions. For example,

$$\lambda x.\mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ 0$$

is the multiset with 42 occurrences of A and 21 occurrences of B (and nothing else). Thus abstractions of the form

$$\lambda x.\mathit{if}\ x = t_1\ \mathit{then}\ s_1\ \mathit{else}\ \ldots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0$$

are adopted to represent (extensional) sets, multisets, and so on.

These considerations motivate the introduction of the class of basic terms that are used to represent individuals (Lloyd, 2003). The definition of basic terms is an inductive one consisting of three parts. The first part covers data types such as lists and trees and uses the same constructs for this as are used in functional programming languages. The second part uses abstractions to cover data types such as (finite) sets and multisets, for which the data can be represented by a finite lookup table. The third part covers data types that are product types and therefore allows the representation of tuples. The definition is inductive in the sense that basic terms include lists of sets of tuples, tuples of sets, and so on.

It is common in learning applications to need to generate spaces of predicates. This is because features are typically predicates and logical hypothesis languages contain predicates. Thus, there is a need to specify grammars that can generate spaces of predicates. In addition to first-order approaches based on refinement operators or antecedent description grammars, higher-order logic offers another approach to this task based on the idea of generating predicates by composing certain primitive functions.

Predicate rewrite systems are used to define spaces of standard predicates, where standard predicates are predicates in a particular syntactic form that involves composing certain functions (Lloyd, 2003). A predicate rewrite is an expression of the form $p \rightarrowtail q$, where p and q are standard predicates. The predicate p is called the *head* and q is the *body* of the rewrite. A predicate rewrite system is a finite set of predicate rewrites. One should think of a predicate rewrite system as a kind of grammar for generating a particular class of predicates. Roughly speaking, this works as follows. Starting from the weakest predicate *top*, all predicate rewrites that have *top* (of the appropriate type) in the head are selected to make up child predicates that consist of the bodies of these predicate rewrites. Then, for each child predicate and each redex (i.e., subterm selected for expansion) in that predicate, all child predicates are generated by replacing each redex by the body of the predicate rewrite whose head is identical to the redex. This generation of predicates continues to produce the entire space of predicates given by the predicate rewrite system.

Predicate rewrite systems are a convenient mechanism to specify precise control over the space of predicates that is to be generated. Note that predicate rewrite systems depend essentially on the higher-order nature of the logic since standard predicates are obtained by composition of functions and composition is a higher-order function.

Other ingredients of learning problems, such as background theories and training examples, can also be conveniently represented in higher-order logic.

Reasoning

Machine learning applications require that reasoning tasks be carried out, for example, computing the value of some predicate on some individual. Generally, reasoning in (higher-order) logic can be either theorem proving or purely equational reasoning or a combination of both.

A variety of proof systems have been developed for higher-order logic; these include Hilbert-style systems (Andrews, 2002) and tableau systems (Fitting, 2002).

Purely equational reasoning includes the computational models of functional programming languages and therefore can be usefully thought of as computation. Typical examples of this approach include the declarative programming languages Curry (Hanus, 2006) and Escher (Lloyd, 2003) which are extensions of the functional programming language Haskell (Peyton Jones, 2003). For both Curry and Escher, the Haskell computational model is generalized in such a way as to admit the logic programming idioms.

Alternatively, by suitably restricting the fragment of the logic considered and the proof system, computation systems in the form of declarative programming languages can be developed. A prominent example of this approach is the logic programming language λProlog that was introduced in the 1980s (Nadathur & Miller, 1998). In λProlog, program statements are higher-order hereditary Harrop formulas, a generalization of the definite ►clauses used by ►Prolog. The language provides an elegant use of λ-terms as data structures, meta-programming facilities, universal quantification and implications in goals, among other features.

Applications

Higher-order logic has been used in a variety of machine learning settings including decision-tree learning, kernels, Bayesian networks, and evolutionary computing. Decision-tree learning based on the use of higher-order logic as the knowledge representation and reasoning language is presented in Bowers, Giraud-Carrier, and Lloyd (2000), and further developed in Ng (2005b). Kernels and distances over individuals represented by basic terms are studied in Gärtner, Lloyd, and Flach (2004). In Gyftodimos and Flach (2005), Bayesian networks over basic terms are defined and it is shown there how to construct probabilistic classifiers over such networks. In Ng, Lloyd, and Uther (2008), higher-order logic is used as the setting for studying probabilistic modelling, inference and learning. An evolutionary approach to learning higher-order concepts is demonstrated in Kennedy and Giraud-Carrier (1999). In addition, the learnability of hypothesis languages expressed in higher-order logic is investigated in Ng (2005a, 2006).

Cross References

►First-Order Logic
►Inductive Logic Programming
►Learning from Structured Data
►Propositional Logic

Recommended Reading

Andrews, P. B. (2002). *An introduction to mathematical logic and type theory: To truth through proof* (3rd ed.). Dordrecht: Kluwer Academic Publishers.

Bowers, A. F., Giraud-Carrier, C., & Lloyd, J. W. (2000). Classification of individuals with complex structure. In P. Langley (Ed.), *Machine learning: Proceedings of the seventeenth international conference (ICML 2000)* (pp. 81–88). Stanford, CA: Morgan Kaufmann.

Church, A. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic, 5*, 56–68.

Dietzen, S., & Pfenning, F. (1992). Higher-order and modal logic as a framework for explanation-based generalization. *Machine Learning, 9*, 23–55.

Farmer, W. (2008). The seven virtues of simple type theory. *Journal of Applied Logic, 6*(3), 267–286.

Feng, C., & Muggleton, S. H. (1992). Towards inductive generalisation in higher order logic. In D. Sleeman & P. Edwards (Eds.), *Proceedings of the ninth international workshop on machine learning* (pp. 154–162). San Mateo, CA: Morgan Kaufmann.

Fitting, M. (2002). *Types, tableaus, and Gödel's god*. Dordrecht: Kluwer Academic Publishers.

Flach, P., Giraud-Carrier, C., & Lloyd, J. W. (1998). Strongly typed inductive concept learning. In D. Page (Ed.), *Inductive logic programming, 8th international conference, ILP-98*. Lecture Notes in Artificial Intelligence 1446 (pp. 185–194). Berlin: Springer.

Gärtner, T., Lloyd, J. W., & Flach, P. (2004). Kernels and distances for structured data. *Machine Learning, 57*(3), 205–232.

Gyftodimos, E., & Flach, P. (2005). Combining Bayesian networks with higher-order data representations. In *Proceedings of 6th international symposium on intelligent data analysis (IDA 2005). Lecture notes in computer science* (Vol. 3646, pp. 145–156). Berlin: Springer.

Hanus, M. (Ed.). (2006). *Curry: An integrated functional logic language*. http://www.informatik.uni-kiel.de/~curry. Retrieved 21 December 2009.

Henkin, L. (1950). Completeness in the theory of types. *Journal of Symbolic Logic, 15*(2), 81–91.

Kennedy, C. J., & Giraud-Carrier, C. (1999). An evolutionary approach to concept learning with structured data. In *Proceedings of the fourth international conference on artificial neural networks and genetic algorithms (ICANNGA'99)* (pp. 331–366). Berlin: Springer.

Lloyd, J. W. (2003). *Logic for learning. Cognitive technologies*. Berlin: Springer.

Lu, J., Harao, M., & Hagiya, M. (1998). Higher order generalization. In *JELIA '98: Proceedings of the European workshop on logics in artificial intelligence. Lecture notes in artificial intelligence* (Vol. 1489, pp. 368–381). Berlin: Springer.

Muggleton, S., & Page, C. D. (1994). *Beyond first-order learning: Inductive learning with higher-order logic*. Technical report PRG-TR-13-94, Oxford University Computing Laboratory.

Nadathur, G., & Miller, D. A. (1998). Higher-order logic programming. In D. M. Gabbay, C. J. Hogger, & J. A. Robinson (Eds.), *The handbook of logic in artificial intelligence and logic programming* (Vol. 5, pp. 499–590). Oxford: Oxford University Press.

Ng, K. S. (2005a). Generalization behaviour of alkemic decision trees. In *Inductive logic programming, 15th international conference (ILP 2005). Lecture notes in artificial intelligence* (Vol. 3625, pp. 246–263). Berlin: Springer.

Ng, K. S. (2005b). *Learning comprehensible theories from structured data*. PhD thesis, Computer Sciences Laboratory, The Australian National University.

Ng, K. S. (2006). (Agnostic) PAC learning concepts in higher-order logic. In *European conference on machine learning (ECML 2006). Lecture notes in artificial intelligence* (Vol. 4212, pp. 711–718). Berlin: Springer.

Ng, K. S., Lloyd, J. W., & Uther, W. T. B. (2008). Probabilistic modelling, inference and learning using logical theories. *Annals of Mathematics and Artificial Intelligence*, 54, 159–205. Doi: 10.1007/s 10472-009-9136-7.

Peyton Jones, S. (Ed.) (2003). *Haskell 98 language and libraries: The revised report. Cambridge*: Cambridge University Press.

Wolfram, D. A. (1993). *The clausal theory of types*. Cambridge: Cambridge University Press.

HMM

►Hidden Markov Models

Hold-One-Out Error

►Leave-One-Out Error

Holdout Data

►Holdout Set

Holdout Evaluationl

Definition

Holdout evaluation is an approach to ►out-of-sample evaluation whereby the available data are partitioned into a ►training set and a ►test set. The test set is thus ►out-of-sample data and is sometimes called the *holdout set* or *holdout data*. The purpose of holdout evaluation is to test a model on different data to that from which it is ►learned. This provides an unbiased estimate of learning performance, in contrast to ►in-sample evaluation.

In *repeated holdout evaluation*, repeated holdout evaluation experiments are performed, each time with

a different partition of the data, to create a distribution of ►training and ►test sets with which an algorithm is assessed.

Cross References
►Algorithm Evaluation

Holdout Set

Synonyms
Holdout data

Definition
A holdout set is a ►data set containing data that are not used for learning and that are used for ►evaluation by a ►learning system.

Cross References
►Evaluation Set
►Holdout Evaluation

Hopfield Network

Risto Miikkulainen
The University of Texas at Austin, Austin, TX, USA

Synonyms
Recurrent associative memory

Definition
The Hopfield network is a binary, fully recurrent network that, when started on a random activation state, settles the activation over time into a state that represents a solution (Hopfield & Tank, 1986). This architecture has been analyzed thoroughly using tools from statistical physics. In particular, with symmetric weights, no self-connections, and asynchronous neuron activation updates, a Lyapunov function exists for the network, which means that the network activity will eventually settle. The Hopfield network can be used as an associate memory or as a general optimizer. When used as an associative memory, the weight values are computed from the set of patterns to be stored. During retrieval, part of the pattern to be retrieved is activated, and the network settles into the complete pattern. When used as an optimizer, the function to be optimized is mapped into the Lyapunov function of the network, which is then solved for the weight values. The network then settles to a state that represents the solution. The basic Hopfield architecture can be extended in many ways, including continuous neuron activations. However, it has limited practical value mostly because it is not strong in either of the above task: as an associative memory, its capacity is approximately $0.15N$ in practice (where N is the number of neurons), and as an optimizer, it often settles into local optima instead of the global one. The ►Boltzmann Machine extends the architecture with hidden neurons, allowing for better performance in both tasks. However, the Hopfield network has had a large impact in the field because the theoretical techniques developed for it have inspired theoretical approaches for other architectures as well, especially for those of self-organizing systems (e.g., ►Self Organizing Maps, ►Adaptive Resonance Theory).

Recommended Reading
Hopfield, J. J., & Tank, D. W. (1986). Computing with neural circuits: A model. *Science, 233*, 624–633.

Hypothesis Language

Hendrik Blockeel
Katholieke Universiteit Leuven, Belgium
Leiden Institute of Advanced Computer Science
The Netherlands

Synonyms
Representation language

Definition
The *hypothesis language* used by a machine learning system is the language in which the hypotheses (also referred to as patterns or models) it outputs are described.

Motivation and Background

Most machine learning algorithms can be seen as a procedure for deriving one or more hypotheses from a set of observations. Both the input (the observations) and the output (the hypotheses) need to be described in some particular language. This language is respectively called the ►Observation Language or the hypothesis language. These terms are mostly used in the context of symbolic learning, where these languages are often more complex than in subsymbolic or statistical learning. For instance, hypothesis languages have received a lot of attention in the field of ►Inductive Logic Programming, where systems typically take as one of their input parameters a declarative specification of the hypothesis language they are supposed to use (which is typically a strict subset of full clausal logic). Such a specification is also called a ►Language Bias.

Examples of Hypothesis Languages

The hypothesis language used obviously depends on the learning task that is performed. For instance, in predictive learning, the output is typically a function, and thus the hypothesis language must be able to represent functions; whereas in clustering the language must have constructs for representing clusters (sets of points). Even for one and the same goal, different languages may be used; for instance, decision trees and rule sets can typically represent the same type of functions, so the difference between these two is mostly syntactic.

In the following section, we discuss briefly a few different formalisms for representing hypotheses. For most of these, there are separate entries in this volume that offer more detail on the specifics of that formalism.

Decision Trees and Rule Sets

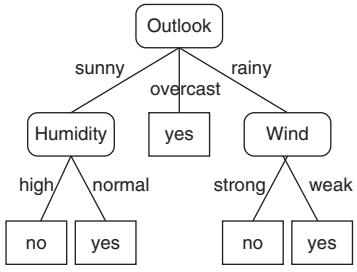
A ►Decision Tree represents a decision process where consecutive tests are performed on an instance to determine the value of its target variable, and at each step in this process, the test that is performed depends on the outcome of previous tests. Each leaf of the tree contains the set of all instances that fulfill the conjunctions of all conditions on the path from the root to this leaf, and as such a tree can easily be written as a set of if-then rules where each rule contains one such conjunction. If the target variable is boolean, this format corresponds to disjunctive normal form.

Figure 1 shows a decision tree and the corresponding rule set. (Inspired by Mitchell, 1997).

Graphical Models

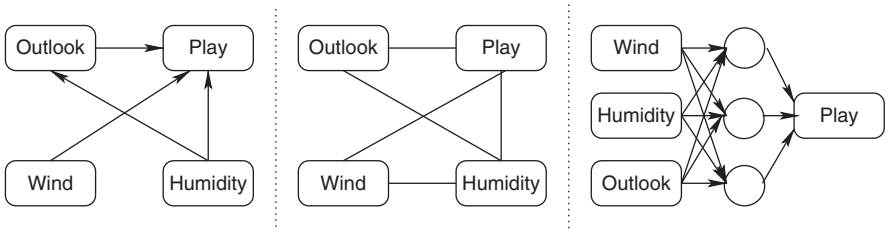
The term "graphical models" usually refers to probabilistic models where the joint distribution over a set of variables is defined as the product of a number of joint distributions over subsets of these variables (i.e., a factorization), and this factorization is defined by a graph structure. The graph may be directed, in which case we speak of a ►Bayesian Network, undirected, in which case we speak of a ►Markov Network, or even a mix of the two (so-called chain graphs). In a Bayesian network, the constituent distributions of the factorization are conditional probability functions associated with each node. In a Markov network, the constituent distributions are potential functions associated with each clique in the graph.

Two learning settings can be distinguished: learning the parameters of a graphical model given the model structure (the graph), and learning both structure and parameters of the model. In the first case, the graph is in fact a language bias specification: the user forces the learner to return a hypothesis that lies within the set



IF Outlook=sunny AND Humidity=high THEN Play=no
IF Outlook=sunny AND Humidity=normal THEN Play=yes
IF Outlook=overcast THEN Play=yes
IF Outlook=rainy AND Wind=strong THEN Play=no
IF Outlook=rainy AND Wind=weak THEN Play=yes

Hypothesis Language. Figure 1. A decision tree and an equivalent rule set



Hypothesis Language. Figure 2. A Bayesian network, a Markov network, and a neural network

of hypotheses representable by this particular structure. In the second case, the structure of the graph makes explicit certain independencies that are hypothesized to exist between the variables (thus it is part of the hypothesis itself).

Figure 2 shows examples of possible graphical models that might be learned from data. For details about the interpretation of such graphical models, we refer to the respective entries in this encyclopedia.

Neural Networks

►Neural networks are typically used to represent complex nonlinear functions. A neural network can be seen as a directed graph where the nodes are variables and edges indicate which variables depend on which other variables. Some nodes represent the observed input variables x_i and output variables y, and some represent new variables introduced by the network. Typically, a variable depends, in a nonlinear way, on a linear combination of those variables that directly precede it in the directed graph. The parameters of the network are numerical edge labels that represent the weight of a parent variable in that linear combination.

As with graphical models, one can learn the parameters of a neural network with a given structure, in which case the structure serves as a language bias; or one can learn both the structure and the parameters of the network.

Figure 2 shows an example of a neural network. We refer to the respective entry for more information on neural networks.

Instance-Based Learning

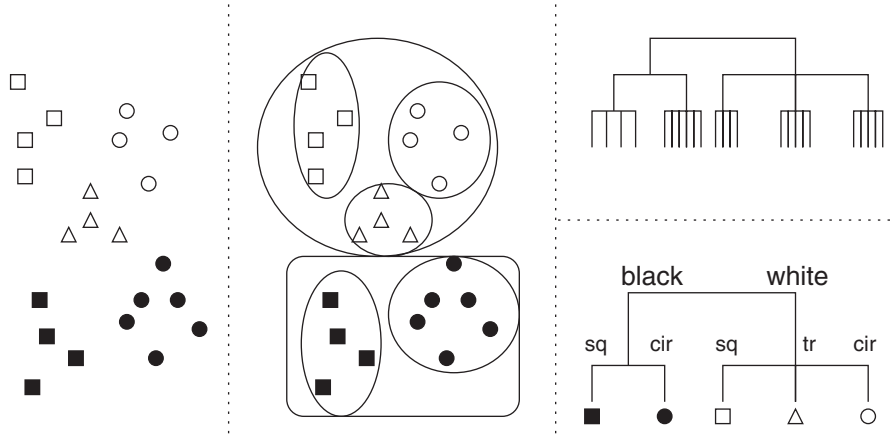
In the most basic version of ►instance-based learning, the training data set itself represents the hypothesis. As such, the hypothesis language is simply the powerset of the observation language. Because many instance-based learners rescale the dimensions of the input space, the vector containing the rescaling factors can be seen as part of the hypothesis. Similarly, some methods derived from instance-based learning build a model in which the training set instances are replaced by prototypes (one prototype being representative for a set of instances) or continuous functions approximating the instances.

Clustering

In clustering tasks, there is an underlying assumption that there is a certain structure in the data set; that is, the data set is really a mixture of elements from different groups or clusters, with each cluster corresponding to a different population. The goal is to describe these clusters or populations and to indicate which data elements belong to which cluster.

Some clustering methods define the clusters extensionally, that is, they describe the different clusters in the dataset by just enumerating the elements in the dataset that belong to them. Other methods add an intensional description to the clusters, defining the properties that an instance should have in order to belong to the cluster; as such, these intensional methods attempt to describe the population that the cluster is a sample from. Some methods recursively group the clusters into larger clusters, building a cluster hierarchy. Figure 3 shows an example of such a cluster hierarchy.

The term "mixture models" typically refers to methods that return a probabilistic model (e.g., a Gaussian distribution with specified parameters) for each separate population identified. Being probabilistic in nature, these methods typically also assign data elements to the populations in a probabilistic, as opposed to deterministic, manner.



Hypothesis Language. Figure 3. A hierarchical clustering: *left*, the data set; *middle*: an extensional clustering shown on the data set; *right*, above: the corresponding extensional clustering tree; *right*, below: a corresponding intensional clustering tree, where the clusters are described based on color and shape of their elements

First-Order Logic Versus Propositional Languages

In symbolic machine learning, a distinction is often made between the so-called attribute-value (or propositional) and relational (or first-order) languages. The terminology "propositional" versus "first-order" originates in logic. In ►Propositional Logic, only the existence of propositions, which can be true or false, is assumed, and these propositions can be combined with the usual logical connectives into logical formulae. In ►First-Order Predicate Logic, the existence of a universe of objects is assumed as well as the existence of predicates that can express certain properties of and relationships between these objects. By adding variables and quantifiers, one can describe deductive reasoning processes in first-order logic that cannot be described in propositional logic. For instance, in propositional logic, one could state propositions *Socrates_is_human* and *all_humans_are_mortal* (both are statements that may be true or false), but there is no inherent relationship between them. In first order logic, the formulae $human(Socrates)$ and $\forall x : human(x) \rightarrow mortal(x)$ allow one to deduce $mortal(Socrates)$. A more extensive explanation of the differences between propositional and first-order logic can be found in the entry on ►First-Order Logic.

Many machine learning approaches use an essentially propositional language for describing observations and hypotheses. In the fields of Inductive Logic Programming and ►Relational Learning, more powerful languages are used, with an expressiveness closer to that of first-order logic. Many of the representation languages mentioned above, which are essentially propositional, have been extended towards the first-order logic context.

The simplest example is that of rule sets. If-then rules have a straightforward counterpart in first-order logic in the form of ►Clauses, which are usually written as logical implications where all variables are interpreted as universally quantified. For instance, the rule "IF Human=true THEN Mortal=true" can be written in clausal form as

$$mortal(x) \leftarrow human(x).$$

Propositional rules correspond to clauses that refer to only one object (and the object reference is implicit). A rule such as

$$grandparent(x, y) \leftarrow parent(x, z), parent(z, y)$$

(expressing that, for any x, y, z, whenever x is a parent of z and z is a parent of y, x is a grandparent of y) has no translation into propositional logic that retains the inference capacity of the first-order logic clause.

Clauses are a natural first-order logic equivalent to the if-then rules typically returned by rule learners, and many of the other representation languages have also been upgraded to the relational or first-order-logic context. For instance, several researchers (e.g., Blockeel & De Raedt, 1998) have upgraded the

formalism of decision trees toward "structural" or "first-order logic" decision trees. Probabilistic relational models (Getoor, Friedman, Koller, & Pfeffer, 2001) and Bayesian logic programs (Kersting & De Raedt, 2001) are examples of how Bayesian networks have been upgraded, while Markov networks have been lifted to "Markov logic" (Richardson & Domingos, 2006).

Further Reading

Most of the literature on hypothesis and observation languages is found in the area of inductive logic programming. Excellent starting points, containing extensive examples of bias specifications, are *Relational Data Mining* by Džeroski & Lavrač (2001), *Logic for Learning* by Lloyd (2003), and *Logical and Relational Learning* by De Raedt (2008).

De Raedt (1998) compares a number of different observation and hypothesis languages with respect to their expressiveness, and indicates relationships between them.

Cross References

►First-Order Logic
►Hypothesis Space
►Inductive Logic Programming
►Observation Language

Recommended Reading

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence, 101*(1–2), 285–297.

De Raedt, L. (1998). Attribute-value learning versus inductive logic programming: the missing links (extended abstract). In D. Page (Ed.), *Proceedings of the eighth international conference on inductive logic programming. Lecture notes in artificial intelligence* (Vol. 1446, pp. 1–8). Berlin: Springer.

De Raedt, L. (2008). *Logical and relational learning*. Berlin: Springer.

Džeroski, S., & Lavrač, N. (Ed.). (2001). *Relational data mining*. Berlin: Springer.

Getoor, L., Friedman, N., Koller, D., & Pfeffer, A. (2001). Learning probabilistic relational models. In S. Dzeroski & N. Lavrac (Eds.), *Relational data mining* (pp. 307–334). Berlin: Springer.

Kersting, K., & De Raedt, L. (2001). Towards combining inductive logic programming and Bayesian networks. In C. Rouveirol & M. Sebag (Eds.), *Proceedings of the 11th international conference on inductive logic programming Lecture notes in computer science* (Vol. 2157, pp. 118–131). Berlin: Springer.

Lloyd, J. W. (2003). *Logic for learning*. Berlin: Springer.

Mitchell, T. (1997). Machine Learning. McGraw Hill.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning, 62*(1–2), 107–136.

Hypothesis Space

Hendrik Blockeel
Katholieke Universiteit Leuven, Belgium
Leiden Institute of Advanced Computer Science
The Netherlands

Synonyms

Model space

Definition

The *hypothesis space* used by a machine learning system is the set of all hypotheses that might possibly be returned by it. It is typically defined by a ►Hypothesis Language, possibly in conjunction with a ►Language Bias.

Motivation and Background

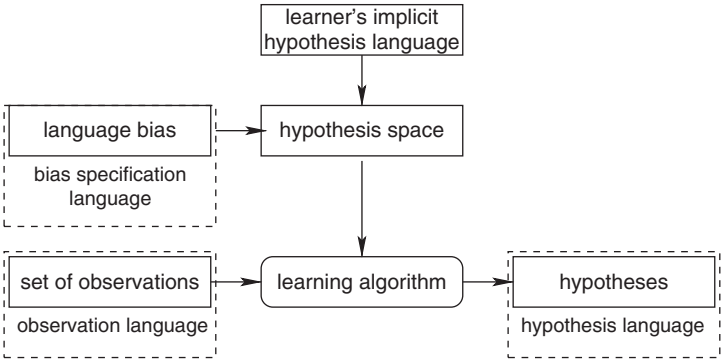
Many machine learning algorithms rely on some kind of search procedure: given a set of observations and a space of all possible hypotheses that might be considered (the "hypothesis space"), they look in this space for those hypotheses that best fit the data (or are optimal with respect to some other quality criterion).

To describe the context of a learning system in more detail, we introduce the following terminology. The key terms have separate entries in this encyclopedia, and we refer to those entries for more detailed definitions.

A learner takes observations as inputs. The ►Observation Language is the language used to describe these observations.

The hypotheses that a learner may produce, will be formulated in a language that is called the Hypothesis Language. The *hypothesis space* is the set of hypotheses that can be described using this hypothesis language.

Often, a learner has an implicit, built-in, hypothesis language, but in addition the set of hypotheses



Hypothesis Space. Figure 1. Structure of learning systems that derive one or more hypotheses from a set of observations

that can be produced can be restricted further by the user by specifying a language bias. This language bias defines a subset of the hypothesis language, and correspondingly a subset of the hypothesis space. A separate language, called the ▶Bias Specification Language, is used to define this language bias. Note that while elements of a hypothesis language refer to a single hypothesis, elements of a bias specification language refer to sets of hypotheses, so these languages are typically quite different. Bias specification languages have been studied in detail in the field of ▶Inductive Logic Programming.

The terms "hypothesis language" and "hypothesis space" are sometimes used in the broad sense (the language that the learner is inherently restricted to, e.g., Horn clauses), and sometimes in a more narrow sense, referring to the smaller language or space defined by the language bias.

The structure of a learner, in terms of the above terminology, is summarized in Fig. 1.

Theory

For a given learning problem, let us denote with $\mathcal{O}$ the set of all possible observations (sometimes also called the instance space), and with $\mathcal{H}$ the hypothesis space, i.e., the set of all possible hypotheses that might be learned. Let 2^X denote the power set of a set X. Most learners can then be described abstractly as a function $T : 2^{\mathcal{O}} \to \mathcal{H}$, which takes as input a set of observations (also called the training set) $S \subseteq \mathcal{O}$, and produces as output a hypothesis $h \in \mathcal{H}$.

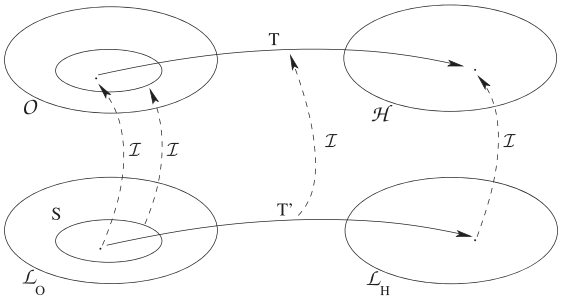
In practice, the observations and hypotheses are represented by elements of the observation language $\mathcal{L}_O$ and the hypothesis language $\mathcal{L}_H$, respectively. The connection between language elements and what they represent is defined by functions $\mathcal{I}_O : \mathcal{L}_O \to \mathcal{O}$ (for observations) and $\mathcal{I}_H : \mathcal{L}_H \to \mathcal{H}$ (for hypotheses). This mapping is often, but not always, bijective. When it is not bijective, different representations for the same hypothesis may exist, possibly leading to redundancy in the learning process.

We will use the symbol $\mathcal{I}$ as a shorthand for $\mathcal{I}_O$ or $\mathcal{I}_H$. We also define the application of $\mathcal{I}$ to any set S as $\mathcal{I}(S) = \{\mathcal{I}(x) | x \in S\}$, and to any function f as $\mathcal{I}(f) = g \Leftrightarrow \forall x : g(\mathcal{I}(x)) = \mathcal{I}(f(x))$.

Thus, a machine learning system really implements a function $T' : 2^{\mathcal{L}_O} \to \mathcal{L}_H$, rather than a function $T : 2^{\mathcal{O}} \to \mathcal{H}$. The connection between T' and T is straightforward: for any $S \subseteq \mathcal{L}_O$ and $h \in \mathcal{L}_H$, $T'(S) = h$ if and only if $T(\mathcal{I}(S)) = \mathcal{I}(h)$; in other words: $T = \mathcal{I}(T')$.

Figure 2 summarizes these languages and spaces and the connections between them. We further illustrate them with a few examples.

Example 1 *In supervised learning, the observations are usually pairs (x, y) with $x \in X$ an instance and $y \in Y$ its label, and the hypotheses are functions mapping X onto Y. Thus $\mathcal{O} = X \times Y$ and $\mathcal{H} \subseteq Y^X$, with Y^X the set of all functions from X to Y. $\mathcal{L}_O$ is typically chosen such that $\mathcal{I}(\mathcal{L}_O) = \mathcal{O}$, i.e., each possible observation can be represented in $\mathcal{L}_O$. In contrast to this, in many cases $\mathcal{I}(\mathcal{L}_H)$ will be a strict subset of Y^X, i.e., $\mathcal{I}(\mathcal{L}_H) \subset Y^X$.*



Hypothesis Space. Figure 2. Illustration of the interpretation function $\mathcal{I}$ mapping $\mathcal{L}_O$, $\mathcal{L}_H$, and T' onto $\mathcal{O}$, $\mathcal{H}$, and T

For instance, $\mathcal{L}_H$ may contain representations of all polynomial functions from X to Y if $X = \mathbf{R}^n$ and $Y = \mathbf{R}$ (with $\mathbf{R}$ the set of real numbers), or may be able to represent all conjunctive concepts over X when $X = \mathbf{B}^n$ and $Y = \mathbf{B}$ (with $\mathbf{B}$ the set of booleans).

When $\mathcal{I}(\mathcal{L}_H) \subset Y^X$, the learner cannot learn every imaginable function. Thus, $\mathcal{L}_H$ reflects an inductive bias that the learner has, called its *language bias*. We can distinguish an implicit language bias, inherent to the learning system, and corresponding to the hypothesis language (space) in the broad sense, and an explicit language bias formulated by the user, corresponding to the hypothesis language (space) in the narrow sense.

Example 2 *Decision tree learners and rule set learners use a different language for representing the functions they learn (call these languages $\mathcal{L}_{DT}$ and $\mathcal{L}_{RS}$, respectively), but their language bias is essentially the same: for instance, if $X = \mathbf{B}^n$ and $Y = \mathbf{B}$, $\mathcal{I}(\mathcal{L}_{DT}) = \mathcal{I}(\mathcal{L}_{RS}) = Y^X$: both trees and rule sets can represent any boolean function from $\mathbf{B}^n$ to $\mathbf{B}$.*

In practice a decision tree learner may employ constraints on the trees that it learns, for instance, it might be restricted to learning trees where each leaf contains at least two training set instances. In this case, the actual hypothesis language used by the tree learner is a subset of the language of all decision trees.

Generally, if the hypothesis language in the broad sense is $\mathcal{L}_H$ and the hypothesis language in the narrow sense is $\mathcal{L}'_H$, then we have $\mathcal{L}'_H \subseteq \mathcal{L}_H$ and the corresponding spaces fulfill (in the case of supervised learning)

$$\mathcal{I}(\mathcal{L}'_H) \subseteq \mathcal{I}(\mathcal{L}_H) \subseteq Y^X.$$

Clearly, the choice of $\mathcal{L}_O$ and $\mathcal{L}_H$ determines the kind of patterns or hypotheses that can be expressed. See the entries on Observation Language and Hypothesis Language for more details on this.

Further Reading

The term "hypothesis space" is ubiquitous in the machine learning literature, but few articles discuss the concept itself. In Inductive Logic Programming, a significant body of work exists on how to define a language bias (and thus a hypothesis space), and on how to automatically weaken the bias (enlarge the hypothesis space) when a given bias turns out to be too strong. The expressiveness of particular types of learners (e.g., classes of ►Neural Networks) has been studied, and this relates directly to the hypothesis space they use. We refer to the respective entries in this volume for more information on these topics.

Cross References

►Bias Specification Language
►Hypothesis Language
►Inductive Logic Programming
►Observation Language

Recommended Reading

De Raedt, L. (1992). *Interactive theory revision: An inductive logic programming approach*. London: Academic Press.

Nédellec, C., Adé, H., Bergadano, F., & Tausend, B. (1996). Declarative bias in ILP. In L. De Raedt (Ed.), *Advances in inductive logic programming. Frontiers in artificial intelligence and applications* (Vol. 32, pp. 82–103). Amsterdam: IOS Press.

Hypothesis Space

Definition

A *hypothesis space* is the space of hypotheses through which a learning algorithm can search for a model. See ►Learning as Search.

ID3

►Decision Tree

Identification

►Classification

Identity Uncertainty

►Entity Resolution

Idiot's Bayes

►Naïve Bayes

Immune Computing

►Artificial Immune Systems

Immune Network

A proposed theory that the immune system is capable of achieving immunological memory by the existence of a mutually reinforcing network of B-cells. This network of B-cells forms due to the ability of the paratopes, located on B-cells, to match against the idiotopes on other B-cells. The binding between the idiotopes and paratopes has the effect of stimulating the B-cells. This is because the paratopes on B-cells react to the idiotopes on similar B-cells, as it would an antigen. However, to counter the reaction there is a certain amount of suppression between the B-cells which acts as a regulatory mechanism. This interaction of the B-cells due to the network was said to contribute to a *stable* memory structure and account for the retainment of memory cells, even in the absence of antigen. This interaction of cells forms the basis of inspiration for a large number of AIS algorithms, for example aiNET.

Immune-Inspired Computing

►Artificial Immune Systems

Immunocomputing

►Artificial Immune Systems

Immunological Computation

►Artificial Immune Systems

Implication

►Entailment

Improvement Curve

►Learning Curves in Machine Learning

Incremental Learning

Paul E. Utgoff
University of Massachusetts, Amherst, USA

Definition

Incremental learning refers to any ►online learning process that learns the same ►model as would be learnt by a ►batch learning algorithm.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Motivation and Background

Incremental learning is useful when the input to a learning process occurs as a stream of distinct observations spread out over time, with the need or desire to be able to use the result of learning at any point in time, based on the input observations received so far. In principle, the stream of observations may be infinitely long, or the next observation long delayed, precluding any hope of waiting until all the observations have been received. Without the ability to forestall learning, one must commit to a sequence of hypotheses or other learned artifacts based on the inputs observed up to the present. One would rather not simply accumulate and store all the inputs and, upon receipt of each new one, apply a batch learning algorithm to the entire sequence of inputs received so far. It would be preferable computationally if the existing hypothesis or other artifact of learning could be updated in response to each newly received input observation.

Theory

Consider the problem of computing the balance in one's checkbook account. Most would say that this does not involve learning, but it illustrates an important point about incremental algorithms. One procedure, a batch algorithm based on the fundamental definition of balance, is to compute the balance as the sum of the deposits less the sum of the checks and fees. As deposit, check, and fee transactions accumulate, this definition remains valid. There is an expectation that there will be more transactions in the future, and there is also a need to compute the balance periodically to ensure that no contemplated check or fee will cause the account to become overdrawn. We cannot wait to receive all of the transactions and then compute the balance just once.

One would prefer an incremental algorithm for this application, to reduce the cost of computing the balance after each transaction. This can be accomplished by recording and maintaining one additional piece of information, the balance after the nth transaction. It is a simple matter to prove that the balance after n transactions added to the amount of transaction $n+1$ provides the balance after $n+1$ transactions. This is because the sums of the fundamental definition for $n+1$ transactions can be rewritten as the sums of the fundamental definition for n transactions plus the amount of the nth transaction. This incremental algorithm reduces the computation necessary to know the balance after each transaction, but it increases the bookkeeping effort somewhat due to the need for an additional variable.

Now consider the problem of learning the mean of a real valued variable from a stream of observed values of this variable. Though simple, most would say that this does involve learning, because one estimates the mean from observations, without ever establishing the mean definitively. The fundamental definition for the mean requires summing the observed values and then dividing by the number of observed values. As each new observation is received, one could compute the new mean. However, one can reduce the computational cost by employing an incremental algorithm. For n observations, we could just as well have observed exactly the n occurences of the mean. The sum of these observations divided by n would produce the mean. If we were to be provided with an $n+1$ observation, we could compute the new sum of the $n+1$ observations as n cases of the mean value plus the new observation, divided by $n+1$. This reduces the cost of computing the mean after each observation to one multiplication, two addition, and one division operations. There is a small increase in bookkeeping in maintaining the counter n of how many observations have been received, and the mean m after n observations.

In both of the above examples, the need to record the fundamental data is eliminated. Only a succinct summary of the data needs to be retained. For the checkbook balance, only the balance after n transactions needs to be stored, making the specific amounts for the individual transactions superfluous. For the mean of a variable, only the mean m after n observations and the number n of observations need to be retained, making the specific values of the individual observations superfluous. Due to this characteristic, incremental algorithms are often characterized as memoryless, not because no memory at all is required, but because no memory of the original data items is needed. An incremental algorithm is not required to be memoryless, but the incremental algorithm must operate by modifying its existing knowledge, not by hiding the application of the corresponding batch algorithm to the accumulated set of observations. The critical issue is the extent to which computation is reduced compared to starting

with all the data observations and nothing more. An essential aspect for an incremental algorithm is that the obtained result be identical to that indicated by the fundamental definition of the computation to be performed.

A point of occasional confusion is whether to call an algorithm incremental when it makes adjustments to its data structures in response to a new data observation. The answer depends on whether the result is the same that one would obtain when starting with all the observations in hand. If the answer is no, then one may have an online learning algorithm that is not an incremental learning algorithm. For example, consider two alternative formulations of the problem mentioned above of learning the mean of a variable. Suppose that the count of observations, held in the variable n, is not permitted to exceed some constant, say 100. Then the mean after n observations coupled with the minimum of n and 100 no longer summarizes all n observations accurately. Consider a second reformulation. Suppose that the most recent 100 observations are held in a queue. When a new observation is received, it replaces the oldest of the 100 observations. Now the algorithm can maintain a moving average, but not the overall overage. These may be desirable, if one wishes to remain responsive to drift in the observations, but that is another matter. The algorithm would not be considered incremental because it does not produce the same result for all n observations that the corresponding batch algorithm would for these same n observations. The algorithm would be online, and it would be memoryless, but it would not be computing the same learned artifact as the batch algorithm.

These two latter reformulations raise the issue of whether the order in which the observations are received is relevant. It is often possible to determine this by looking at the fundamental definition of the computation to be performed. If the operator that aggregates the observations is commutative, then order is not important. For the checking account balance example above, the fundamental aggregation is accomplished in the summations, and addition is commutative, so the order of the transactions is not relevant to the resulting balance. If a fundamental algorithm operates on a set of observations, then aggregation of a new observation into a set of observations is accomplished by the set union operator, which is commutative. Can one have an incremental algorithm for which order of the observations is important? In principle, yes, provided that the result of the incremental algorithm after observation n is the same as that of the fundamental algorithm for the first n observations.

A final seeming concern for an incremental learning algorithm is whether the selection of future observations ($n + 1$ and beyond) is influenced by the first n observations. This is a red herring, because for the n observations, the question of whether the learning based on these observations can be accomplished by a batch algorithm or a corresponding incremental algorithm remains. Of course, if one needs to use the result of learning on the first k instances to help select the $k + 1$ instance, then it would be good sense to choose an incremental learning algorithm. One would rather not apply a batch algorithm to each and every prefix of the input stream. This would require saving the input stream and it would require doing much more computation than is necessary.

We can consider a few learning scenarios which suit incremental learning. An ▶active learner uses its current knowledge to select the next observation. For a learner that is inducing a classifier, the observation would be an unclassified instance. The active learner selects an unclassified instance, which is passed to an oracle that attaches a correct class label. Then the oracle returns the labeled instance as the next observation for the learner. The input sequence is no longer one of instances for which each was drawn independently according to a probability distribution over the possible instances. Instead, the distribution is conditionally dependent on what the learner currently believes. The learning problem is sequential in its nature. The observation can be delivered in sequence, and an incremental learning algorithm can modify its hypothesis accordingly. For the n observations received so far, one could apply a corresponding batch algorithm, but this would be unduly awkward.

▶Reinforcement learning is a kind of online learning in which an agent makes repeated trials in a simulated or abstracted world in order to learn a good, or sometimes optimal, policy that maps states to actions. The learning artifact is typically a function V over states or a function Q over state-action pairs. As the agent moves from state to state, it can improve its function over time. The choice of action depends on the current

V or Q and on the reward or punishment received at each step. Thus, the sequence of observations consists of state-reward pairs or state-action-reward triples. As with active learning, the sequence of observations can be seen as being conditionally dependent on what the learner currently believes at each step. The function V or Q can be modified after each observation, without retaining the observation. When the function is approximated in an unbiased manner, by using a lookup table for discrete points in the function domain, there is an analogy with the problem of computing a checkbook balance, as described above. For each cell of the lookup table, its value is its initial value plus the sum of the changes, analogously for transactions. One can compute the function value by computing this sum, or one can store the sum in the cell, as the net value of all the changes. An incremental algorithm is preferable both for reasons of time and space.

A k-nearest classifier (see ►instance based learning) is defined by a set of training instances, the observations, and a distance metric that returns the numeric distance between any two instances. The difference between the batch algorithm and the incremental algorithm is slight. The batch algorithm accepts all the observations at once, and the incremental algorithm simply adds each new observation to the set of observations. If however, there were data structures kept in the background to speed computation, one could distinguish between building those data structures once (batch) and updating those data structures (incremental). One complaint might be that all of the observations are retained. However, these observations do not need to be revisited when a new one arrives. There is an impact on space, but not on time.

A ►decision tree classifier may be correct for the n observations observed so far. When the n+1 observation is received, an incremental algorithm will restructure the tree as necessary to produce the tree that the batch algorithm would have built for these $n+1$ observations. To do this, it may be that no restructuring is required at all, or that restructuring is needed only in a subtree. This is a case in which memory is required for saving observations in the event that some of them may be needed to be reconsidered from time to time. There is a great savings in time over running the corresponding batch algorithm repeatedly.

Applications

Incremental learning is pervasive, and one can find any number of applications described in the literature and on the web. This is likely due to the fact that incremental learning offers computational savings in both time and space. It is also likely due to the fact that human and animal learning takes place over time. There are sound reasons for incremental learning being essential to development.

Future Directions

Increasingly, machine learning is confronted with the problem of learning from input streams that contain many millions, or more, of observations. Indeed, the stream may produce millions of observations per day. Streams with this many instances need to be handled by methods whose memory requirements do not grow much or at all. Memoryless online algorithms are being developed that are capable of handling this much throughput. Consider transaction streams, say of a telephone company, or a credit card company, or a stock exchange, or a surveillance camera, or eye-tracking data, or mouse movement data. For such a rich input stream, one could sample it, thereby reducing it to a smaller stream. Or, one could maintain a window of observations, giving a finite sample that changes but does not grow over time. There is no shortage of applications that can produce rich input streams. New methods capable of handling such heavy streams have already appeared, and we can expect to see growth in this area.

Cross References

►Active Learning
►Cumulative Learning
►Online Learning

Recommended Reading

Domingos, P., & Hulten, G. (2003). A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics, 12*.

Giraud-Carrier, C. (2000). A note on the utility of incremental learning. *AI Communications, 13*, 215–223.

Utgoff, P. E., Berkman, N. C., & Clouse, J. A. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning, 29*, 5–44.

Indirect Reinforcement Learning

►Model-Based Reinforcement Learning

Induction

James Cussens
University of York, Heslington, UK

Definition

Induction is the process of inferring a general rule from a collection of observed instances. Sometimes it is used more generally to refer to any inference from premises to conclusion where the truth of the conclusion does not follow deductively from the premises, but where the premises provide evidence for the conclusion. In this more general sense, induction includes *abduction* where facts rather than rules are inferred. (The word "induction" also denotes a different, entirely deductive form of argument used in mathematics.)

Theory

Hume's Problem of Induction

The *problem of induction* was famously set out by the great Scottish empiricist philosopher David Hume (1711–1776), although he did not actually use the word "induction" in this context. With characteristic bluntness, he argued that:

- there can be no *demonstrative* arguments to prove *that those instances of which we have had no experience resemble those of which we have had experience* (Hume, 1739, Part 3, Section 6).

Since scientists (and machine-learning algorithms) *do* infer future-predicting general laws from past observations, Hume is led to the following unsettling conclusion concerning human psychology (and statistical inference):

- It is not, therefore, reason, which is the guide of life, but custom. That alone determines the mind, in all instances, to suppose the future conformable to the past (Hume, 1740).

That general laws cannot be demonstrated (i.e., deduced) from data is generally accepted. Hume, however, goes further: he argues that past observations do not even affect the *probability* of future events:

- Nay, I will go farther, and assert, that he could not so much as prove by any *probable* arguments, that the future must be conformable to the past. All probable arguments are built on the supposition, that there is this conformity betwixt the future and the past, and therefore can never prove it. This conformity is a *matter of fact*, and if it must be proved, will admit of no proof but from experience. But our experience in the past can be a proof of nothing for the future, but upon a supposition, that there is a resemblance betwixt them. This therefore is a point, which can admit of no proof at all, and which we take for granted without any proof (Hume, 1740).

Induction and Probabilistic Inference

Hume's unwavering skepticism concerning prediction appears at variance with the predictive accuracy of machine learning algorithms: there is much experimental evidence that ML algorithms, once trained on "past observations," make predictions on unseen cases with an accuracy far in excess of what can be expected by chance. This apparent discrepancy between Hume's philosophy and practical experience of statistical inference can be explored using a familiar example from the literature on induction. Let e be the statement that all swans seen so far have been white and let h be the general rule that all swans are white. Since h implies e it follows that $P(e|h) = 1$ and so, using Bayes' theorem, we have that

$$P(h|e) = \frac{P(h)P(e|h)}{P(e)} = \frac{P(h)}{P(e)}. \quad (1)$$

So $P(h|e) > P(h)$ as long as $P(e) < 1$ and $P(h) > 0$. This provides an explanation for the predictive accuracy of hypotheses supported by data: given supporting data they just have increased probability of being true. Of course, most machine learning outputs are not "noise-free" rules like h; almost always hypotheses claim a certain distribution for future data where no particular observation is ruled out entirely – some are just more likely than others. The same basic argument applies: if $P(h) > 0$

then as long as the observed data is more likely given the hypothesis than it is a priori, that is, as long as $P(e|h)/P(e) > 1$, then the probability of h will increase. Even in the (common) case where each hypothesis in the hypothesis space depends on real-valued parameters and so $P(h) = 0$ for all h, Bayes theorem still produces an increase in the probability *density* in the neighborhoods of hypotheses supported by the data.

In all these cases, it appears that e is giving "inductive support" to h. Consider, however, h' which states that all swans until now have been white and *all future swans will be black*. Even in this case, we have that $P(h'|e) > P(h')$ as long as $P(e) < 1$ and $P(h') > 0$, though h and h' make entirely contradictory future predictions. This is a case of Goodman's paradox. The paradox is the result of confusing probabilistic inference with inductive inference. Probabilistic inference, of which Bayes theorem is an instance, is entirely deductive in nature – the conclusions of all probabilistic inferences follow with absolute certainty from their premises (and the axioms of probability). $P(h|e) > P(h)$ for $P(e) < 1$ and $P(h) > 0$ essentially because e has (deductively) ruled out some data that might have refuted h, not because a "conformity betwixt the future and the past" has been established.

Good performance on unseen data can still be explained. Statistical models (equivalently machine learning algorithms) *make assumptions* about the world. These assumptions (so far!) often turn out to be correct. Hume noted that the principle "that like objects, placed in like circumstances, will always produce like effects" (Hume, 1739, Part 3, Section 8) although not deducible from first principles, has been established by "sufficient custom." This is called the *uniformity of nature* principle in the philosophical literature. It is this principle which informs machine learning systems. Consider the standard problem of predicting class labels for attribute-value data using labeled data as training. If an unlabeled test case has attribute values which are "close" to those of many training examples all of which have the same class label then in most systems the test case will be labeled also with this class. Different systems differ in how they measure "likeness": they differ in their ▶inductive bias. A system which posited h' above on the basis of e would have an inductive bias strongly at variance with the uniformity of nature principle.

These issues resurfaced within the machine learning community in the 1990s. This ML work focused on various "▶*no-free-lunch theorems*." Such a theorem essentially states that a uniformity of nature assumption is required to justify any given inductive bias. This is how Wolpert puts in one of the earliest "no-free-lunch" papers:

> This paper proves that it is impossible to justify a correlation between reproduction of a training set and generalization error off of the training set using only a priori reasoning. As a result, the use in the real world of any generalizer which fits a hypothesis function to a training set (e.g., the use of back-propagation) is implicitly predicated on an assumption about the physical universe (Wolpert, 1992).

Note that in Bayesian approaches inductive bias is encapsulated in the prior distribution: once a prior has been determined all further work in Bayesian statistics is entirely deductive. Therefore it is no surprise that inductivists have sought to find "objective" or "logical" prior distributions to provide a firm basis for inductive inference. Foremost among these is Rudolf Carnap (1891–1970) who followed a logical approach – defining prior distributions over "possible worlds" (first-order models) which were in some sense uniform (Carnap, 1950). A modern extension of this line of thinking can be found in Bacchus, Grove, Halpern, and Koller (1996).

Popper

Karl Popper (1902–1994) accepted the Humean position on induction yet sought to defend science from charges of irrationality (Popper, 1934). Popper *replaced* the problem of induction by the problem of criticism. For Popper, scientific progress proceeds by conjecturing universal laws and then subjecting these laws to severe tests with a view to refuting them. According to the *verifiability principle* of the logical positivist tradition, a theory is scientific if it can be experimentally confirmed, but for Popper confirmation is a hopeless task, instead a hypothesis is only scientific if it is *falsifiable*. All universal laws have prior probability of zero, and thus will eternally have probability zero of being true, no matter how many tests they pass. The value of a law can only be measured by how well-tested it is. The degree to which a law has been tested is called its degree of *corroboration* by Popper. The $P(e|h)/P(e)$ term in Bayes

theorem will be high if a hypothesis h has passed many severe tests.

Popper's critique of inductivism continued throughout his life. In the *Popper–Miller* argument (Popper & Miller, 1984), as it became known, it is observed that a hypothesis h is logically equivalent to:

$$(h \leftarrow e) \wedge (h \vee e)$$

for any evidence e. We have that $e \vdash h \vee e$ (where $\vdash$ means "logically implies") and also that (under weak conditions) $p(h \leftarrow e|e) < p(h \leftarrow e)$. From this Popper and Miller argue that

> …we find that what is left of h once we discard from it everything that is logically implied by e, is a proposition that in general is counterdependent on e (Popper & Miller, 1987)

and so

> Although evidence may raise the probability of a hypothesis above the value it achieves on background knowledge alone, every such increase in probability has to be attributed entirely to the *deductive connections* that exist between the hypothesis and the evidence (Popper & Miller, 1987).

In other words if $P(h|e) > P(h)$ this is only because $e \vdash h \vee e$. The Popper–Miller argument found both critics and supporters. Two basic arguments of the critics were that (1) deductive relations only set limits to probabilistic support; infinitely many probability distributions can still be defined on any given fixed system of propositions and (2) Popper–Miller are mischaracterizing induction as the absence of deductive relations, when it actually means *ampliative inference*: concluding more than the premises entail (Cussens, 1996).

Causality and Hempel's Paradox

The branch of philosophy concerned with how evidence can confirm scientific hypotheses is known as ▶*confirmation theory*. Inductivists take the position (against Popper) that observing data which follows from a hypothesis not only fails to refute the hypothesis, but also *confirms* it to some degree: seeing a white swan confirms the hypothesis that all swans are white, because

$$\forall x : \text{swan}(x) \rightarrow \text{white}(x), \text{swan}(\text{white\_swan}) \vdash \text{white}(\text{white\_swan}).$$

But, by the same argument it follows that observing any nonwhite, nonswan (say a black raven) also confirms that all swans are white, since:

$$\forall x : \text{swan}(x) \rightarrow \text{white}(x), \neg\text{white}(\text{black\_raven}) \vdash \neg\text{swan}(\text{black\_raven}).$$

This is Hempel's paradox to which there are a number of possible responses. One option is to accept that the black raven is a confirming instance, as one object in the universe has been ruled out as a potential refuter. The *degree* of confirmation is however of "a miniscule and negligible degree" (Howson & Urbach, 1989, p. 90). Another option is to reject the formulation of the hypothesis as a material implication where $\forall x : \text{swan}(x) \rightarrow \text{white}(x)$ is just another way of writing $\forall x : \neg\text{swan}(x) \vee \text{white}(x)$. Instead, to be a scientific hypothesis of any interest the statement must be interpreted *causally*. This is the view of Imre Lakatos (1922–1974), and since any causal statement has a (perhaps implicit) *ceteris paribus* ("all other things being equal") clause this has implications for refutation also.

> …"all swans are white," if true, would be a mere curiosity unless it asserted that swanness *causes* whiteness. But then a black swan would not refute this proposition, since it may only indicate *other causes* operating simultaneously. Thus "all swans are white" is either an oddity and easily disprovable or a scientific proposition with a ceteris paribus clause and therefore easily undisprovable (Lakatos, 1970, p. 102).

Cross References

▶Abduction
▶Bayesian Statistics
▶Classification
▶Learning from Analogy
▶No-Free Lunch Theorems
▶Nonmonotonic Logic

Recommended Reading

Bacchus, F., Grove, A., Halpern, J. Y., & Koller, D. (1996). From statistical knowledge bases to degrees of belief. *Artificial Intelligence, 87*(1–2), 75–143.

Carnap, R. (1950). *Logical foundations of probability*. Chicago: University of Chicago Press.

Cussens, J. (1996). Deduction, induction and probabilistic support. *Synthese, 108*(1), 1–10.

Howson, C., & Urbach, P. (1989). *Scientific reasoning: The Bayesian approach*. La Salle, IL: Open Court.

Hume, D. (1739). *A treatise of human nature, book one* (Anonymously published).

Hume, D. (1740). *An abstract of a treatise of human nature* (Anonymously published as a pamphlet). London.

Lakatos, I. (1970). Falsification and the methodology of scientific research programmes. In I. Lakatos & A. Musgrave (Eds.), *Criticism and the growth of knowledge* (pp. 91–196). Cambridge, MA: Cambridge University Press.

Popper, K. R. (1959). *The logic of scientific discovery*. London: Hutchinson (Translation of *Logik der Forschung*, 1934).

Popper, K. R., & Miller, D. (1984). The impossibility of inductive probability. *Nature, 310*, 434.

Popper, K. R., & Miller, D. (1987). Why probabilistic support is not inductive. *Philosophical Transactions of the Royal Society of London, 321*, 569–591.

Wolpert, D. H. (1992). On the connection between in-sample testing and generalization error. *Complex Systems, 6*, 47–94.

Induction as Inverted Deduction

►Logic of Generality

Inductive Bias

Synonyms

Learning bias; Variance hint

Definition

Most ML algorithms make predictions concerning future data which cannot be deduced from already observed data. The inductive bias of an algorithm is what choses between different possible future predictions. A strong form of inductive bias is the learner's choice of hypothesis/model space which is sometimes called *declarative bias*. In the case of Bayesian analysis, the inductive bias is encapsulated in the prior distribution.

Cross References

►Induction, Learning as Search

Inductive Database Approach to Graphmining

Stefan Kramer
Technische Universität München
Garching b. München, Germany

Overview

The inductive database approach to graph mining can be characterized by (1) the concept of querying for (subgraph) patterns in databases of graphs, and (2) the use of specific data structures representing the space of solutions. For the former, a query language for the specification of the patterns of interest is necessary. The latter aims at a compact representation of the solution patterns.

Pattern Domain

In contrast to other graph mining approaches, the inductive database approach to graph mining (De Raedt & Kramer, 2001; Kramer, De Raedt, & Helma, 2001) focuses on simple patterns (paths and trees) and complex queries (see below), not on complex patterns (general subgraphs) and simple queries (minimum frequency only). While the first approaches were restricted to paths as patterns in graph databases, they were later extended toward unrooted trees (Rückert & Kramer, 2003, 2004). Most of the applications are dealing with structures of small molecules and structure–activity relationships (SARs), that is, models predicting the biological activity of chemical compounds.

Query Language

The conditions on the patterns of interest are usually called *constraints* on the solution space. Simple constraints are specified by so-called *query primitives*. Query primitives express frequency-related or syntactic constraints. As an example, consider the frequency-related query primitive $f(p, D) \geq t$, meaning that a subgraph pattern p has to occur with a frequency of at least t in the database of graphs D. Analogously, other frequency-related primitives demand a maximum frequency of occurrence, or a minimum agreement with the target class (e.g., in terms of the information gain or the χ^2 statistic). Answering frequency-related

queries generally requires database access. In contrast to frequency-related primitives, syntax-related primitives only restrict the syntax of solution (subgraph) patterns, and thus do not require database access. For instance, we may demand that a pattern p is more specific than "*c:c-Cl*" (formally $p \geq$ *c:c-Cl*) or more general than "*C-c:c:c:c:c-Cl*" (formally $p \leq$ *C-c:c:c:c:c:c-Cl*). The strings in the primitive contain vertex (e.g., "*C*," "*c*," "*Cl*"...) and edge labels (e.g., ":," "-"...) of a path in a graph. Many constraints on patterns can be categorized as either monotonic or anti-monotonic. Minimum frequency constraints, for instance, are anti-monotonic, because all subpatterns (in our case: subgraphs) are frequent as well, if a pattern is frequent (according to some user-defined threshold) in a database. Vice versa, maximum frequency is monotonic, because if a pattern is not too frequent, then all superpatterns (in our case: supergraphs) are not too frequent either. Anti-monotonic or monotonic constraints can be solved by variants of level-wise search and APriori (De Raedt & Kramer, 2001; Kramer, De Raedt, & Helma, 2001; Mannila & Toivonen, 1997). Other types of constraints involving convex functions, for example, related to the target class, can be solved by branch-and-bound algorithms (Morishita & Sese, 2000). Typical query languages offer the possibility to combine query primitives conjunctively or disjunctively.

Data Structures

It is easy to show that solutions to conjunctions of monotonic and anti-monotonic constraints can be represented by *version spaces*, and in particular, borders of the most general and the most specific patterns satisfying the constraints (De Raedt & Kramer, 2001; Mannila & Toivonen, 1997). Version spaces of patterns can be represented in data structures such as *version space trees* (De Raedt, Jaeger, Lee, & Mannila, 2002; Rückert & Kramer, 2003). For sequences, data structures based on *suffix arrays* are known to be more efficient than data structures based on version spaces (Fischer, Heun, & Kramer, 2006). Query languages allowing disjunctive normal forms of monotonic or anti-monotonic primitives yield multiple version spaces as solutions, represented by generalizations of version space trees (Lee & De Raedt, 2003). The inductive database approach to graph mining can also be categorized as *constraint-based mining*, as the goal is to find solution patterns satisfying user-defined constraints.

Recommended Reading

De Raedt, L., Jaeger, M., Lee, S. D., & Mannila, H. (2002). A theory of inductive query answering. In *Proceedings of the 2002 IEEE international conference on data mining* (*ICDM 2002*). IEEE Computer Society, Washington, DC.

De Raedt, L., & Kramer, S. (2001). The levelwise version space algorithm and its application to molecular fragment finding. In *Proceedings of the seventeenth international joint conference on artificial intelligence* (*IJCAI 2001*). Morgan Kaufmann: San Francisco, CA.

Fischer, J., Heun, V., & Kramer, S. (2006). Optimal string mining under frequency constraints. In *Proceedings of the tenth European conference on the principles and practice of knowledge discovery in databases* (*PKDD 2006*). Springer: Berlin.

Kramer, S., De Raedt, L., & Helma, C. (2001). Molecular feature mining in HIV data. In *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining* (*KDD 2001*). ACM Press: New York, NY.

Lee, S. D., & De Raedt, L. (2003). An algebra for inductive query evaluation. In *Proceedings of the third IEEE international conference on data mining* (*ICDM 2003*). IEEE Computer Society, Washington, DC.

Mannila, H., & Toivonen, H. (1997). Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, *1*(3), 241–258.

Morishita, S., & Sese, J. (2000). Traversing itemset lattice with statistical metric pruning. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems* (*PODS 2000*). ACM Press: New York, NY.

Rückert, U., & Kramer, S. (2003). Generalized version space trees. In J.-F. Boulicaut, S. Dzeroski (Eds.), *Proceedings of the second international workshop on knowledge discovery in inductive databases* (*KDID-2003*). Springer: Berlin.

Rückert, U., & Kramer, S. (2004). Frequent free tree discovery in graph data. In *Proceedings of the ACM symposium on applied computing* (*SAC 2004*). ACM Press: New York, NY.

Inductive Inference

Sanjay Jain, Frank Stephan
National University of Singapore,
Singapore, Republic of Singapore

Definition

Inductive inference is a theoretical framework to model learning in the limit. The typical scenario is that the

learner reads successively datum $d_0, d_1, d_2, \ldots$ about a concept and outputs in parallel hypotheses $e_0, e_1, e_2, \ldots$ such that each hypothesis e_n is based on the preceding data $d_0, d_1, \ldots, d_{n-1}$. The hypotheses are expected to converge to a description for the data observed; here the constraints on how the convergence has to happen depend on the learning paradigm considered. In the most basic case, almost all e_n have to be the same correct index e, which correctly explains the target concept. The learner might have some preknowledge of what the concept might be, that is, there is some class $\mathcal{C}$ of possible target concepts – the learner has only to find out which of the concepts in $\mathcal{C}$ is the target concept; on the other hand the learner has to be able to learn every concept which is in the class $\mathcal{C}$.

Detail

The above given scenario of learning is essentially the paradigm of inductive inference introduced by Gold (1967) and known as **Ex** (explanatory) learning. Usually one considers learning of recursive functions or recursively enumerable languages. Intuitively, using coding, one can code any natural phenomenon into subsets of $\mathbb{N}$, the set of natural numbers. Thus, recursive functions from $\mathbb{N}$ to $\mathbb{N}$ or recursively enumerable subsets of $\mathbb{N}$ (called languages here), are natural concepts to be considered.

Here we will mainly consider language learning. Paradigms related to function learning can be similarly defined and we refer the reader to Osherson, Stob and Weinsten, 1986; Jain, Osherson, Royer, & Sharma, 1999.

One normally considers data provided to the learner to be either full positive data (i.e., the learner is told about every element in the target language, one element at a time, but never told anything about elements not in the target language) or full positive data and full negative data (i.e., the learner is told about every element, whether it belongs or does not belong to the target language). Intuitively, the reason for considering only positive data is that in many natural situations, such as language learning by children, scientific exploration (such as in astronomy) one gets essentially only positive data.

A *text* is a sequence of elements over $\mathbb{N} \cup \{\#\}$. Content of a text T, denoted $\mathrm{ctnt}(T)$ is the set of natural numbers in the range of T. For a finite sequence σ over $\mathbb{N} \cup \{\#\}$, one can similarly define $\mathrm{ctnt}(\sigma)$ as the set of natural numbers in the range of σ. A text T is said to be for a language L iff $\mathrm{ctnt}(T) = L$. Intuitively, a text T for L represents sequential presentation of elements of L, with #'s representing pauses in the presentation. For example, the only text for $\varnothing$ is $\#^\infty$. $T[n]$ denotes the initial sequence of T of length n. That is, $T[n] = T(0)T(1)\ldots T(n-1)$. We let SEQ denote the set of all finite sequences over $\mathbb{N} \cup \{\#\}$. An *informant* I is a sequence of elements over $\mathbb{N} \times \{0,1\} \cup \{\#\}$, where for each $x \in \mathbb{N}$, exactly one of $(x,0)$ or $(x,1)$ is in the range of I. An informant I is for L iff $\mathrm{range}(I) - \{\#\} = \{(x, \chi_L(x)) : x \in \mathbb{N}\}$, where χ_L denotes the characteristic function of L.

A learner W is a mapping from SEQ to $\mathbb{N} \cup \{?\}$. Intuitively, output of ? denotes that the learner does not wish to make a conjecture on the corresponding input. The output of e denotes that the learner conjectures hypothesis W_e, where $W_0, W_1, \ldots$ is some acceptable numbering of all the recursively enumerable languages. We say that a learner M converges on T to e iff, for all but finitely many n, $\mathrm{M}(T[n]) = e$.

Explanatory Learning

A learner M **TxtEx**-identifies a language L iff for all texts T for L, M converges to an index e such that $W_e = L$. Learner M **TxtEx**-identifies a class $\mathcal{L}$ of languages iff M **TxtEx**-identifies each language in the class $\mathcal{L}$. Finally, one says that a class $\mathcal{L}$ is **TxtEx**-learnable if some learner **TxtEx**-identifies $\mathcal{L}$. **TxtEx** denotes the collection of all **TxtEx**-learnable classes. One can similarly define **InfEx**-identification, for learning from informants instead of texts. The following classes are important examples:

$$RE = \{L : L \text{ is recursively enumerable}\};$$
$$FIN = \{L : L \text{ is a finite subset of } \mathbb{N}\};$$
$$KFIN = \{L : L = K \cup H \text{ for some } H \in FIN\};$$
$$SD = \{L : W_{\min(L)} = L\};$$
$$COFIN = \{L : \mathbb{N} - L \text{ is finite}\};$$
$$SDSIZE = \{\{e + x : x = 0 \vee x < |W_e|\} : W_e \text{ is finite}\};$$
$$SDALL = \{\{e + x : x \in \mathbb{N}\} : e \in \mathbb{N}\}.$$

Here, in the definition of *KFIN*, K is the halting problem, that is, some standard example of a set, which is recursively enumerable but not recursive. The classes

FIN, *SD*, *SDSIZE*, and *SDALL* are **TxtEx**-learnable (Case & Smith, 1983; Gold, 1967): The learner for *FIN* always conjectures the set of all data observed so far. The learner for *SD* conjectures the least datum seen so far as, eventually, the least observed datum coincides with the least member of the language to be learnt. The learner for *SDSIZE* as well as the learner for *SDALL* also find in the limit the least datum e to occur and translate it into an index for the e-th set to be learnt. The class *KFIN* is not **TxtEx**-learnable, mainly for computational reasons. It is impossible for the learner to determine if the current input datum belongs to K or not; this forces a supposed learner either to make infinitely many mind changes on some text for K or to make an error on $K \cup \{x\}$, for some $x \notin K$. The union *SDSIZE* ∪ *SDALL* is also not **TxtEx**-learnable, although it is the union of two learnable classes; so it is one example of various nonunion theorems. Gold (1967) gave even a more basic example: $FIN \cup \{\mathbb{N}\}$ is not **TxtEx**-learnable. Furthermore, the class *COFIN* is also not **TxtEx**-learnable. However, except for *RE*, all the classes given above are **InfEx**-learnable, so when being fed the characteristic function in place of only an infinite list of all elements, the learners become, in general, more powerful.

Note that the learner never knows when it has converged to its final hypothesis. If the learner is required to know when it has converged to the final hypothesis, then the criterion of learning is the same as finite learning. Here a finite learner is defined as follows: the learner keeps outputting the symbol ? while waiting for enough data to appear and, when the data observed are sufficient, the learner outputs exactly one conjecture different from ?, which then is required to be an index for the input concept in the hypothesis space. The class of singletons, $\{\{n\} : n \in \mathbb{N}\}$ is finitely learnable; the learner just waits until the unique element n of $\{n\}$ has appeared and then knows the language. In contrast to this, the classes *FIN* and *SD* are not finitely learnable.

Blum and Blum (1975) obtained the following fundamental result: Whenever M learns L explanatorily from text then L has a locking sequence for M. Here, a sequence σ is said to be a locking sequence for M on L iff (a) $\mathrm{ctnt}(\sigma) \subseteq L$, (b) for all τ such that $\mathrm{ctnt}(\tau) \subseteq L$, $\mathrm{M}(\sigma) = \mathrm{M}(\sigma\tau)$ and (c) $W_{\mathrm{M}(\sigma)} = L$. If only the first two conditions are satisfied, then the sequence is called a *stabilizing sequence* for M on L (Fulk, 1990). It was shown by Blum and Blum (1975) that if a learner M **TxtEx**-identifies L then there exists a locking sequence σ for M on L. One can use this result to show that certain classes, such as $FIN \cup \{\mathbb{N}\}$, are not **TxtEx**-learnable.

Beyond Explanatory Learning

While **TxtEx**-learning requires that the learner syntactically converges to a final hypothesis, which correctly explains the concept, this is no longer required for the more general criterion of behaviourally correct learning (called **TxtBc**-learning). Here, the learner may not syntactically converge but it is still required that all its hypothesis after sometime are correct, see (Bārzdiņš, 1974b; Osherson & Weinstein, 1982; Osherson, Stob and Weinsten, 1986; Case & Smith, 1983; Case & Lynes, 1982). So there is semantic convergence to a final hypothesis. Thus, a learner M **TxtBc** identifies a language L iff for all texts T for L, for all but finitely many n, $W_{\mathrm{M}(T[n])} = L$. One can similarly define **TxtBc**-learnability of classes of languages and the collection **TxtBc**. Every **TxtEx**-learnable class is **Bc**-learnable, but the class *KFIN* and *SDSIZE* ∪ *SDALL* are **TxtBc**-learnable but not **TxtEx**-learnable. Furthermore, **InfEx** $\not\subseteq$ **TxtBc**, for example, $FIN \cup \{\mathbb{N}\}$ is **InfEx**-learnable but not **TxtBc**-learnable. On the other hand, every class that is finitely learnable from informant is also **TxtEx**-learnable (Sharma, 1998).

An intermediate learning criterion is **TxtFex**-learning (Case, 1999) or vacillatory learning, which is similar to **TxtBc**-learning except that we require that the number of distinct hypothesis output by the learner on any text is finite. Here one says that the learner **TxtFex**$_n$-learns the language L iff the number of distinct hypothesis that appear infinitely often on any text T for L is bounded by n. Note that **TxtFex**$_*$ = **TxtFex**. Case (1999) showed that

$$\mathbf{TxtEx} = \mathbf{TxtFex}_1 \subset \mathbf{TxtFex}_2 \subset \mathbf{TxtFex}_3 \subset \ldots \subset \mathbf{TxtFex}_* \subset \mathbf{TxtBc}.$$

For example, the class *SD* ∪ *SDALL* is actually **TxtFex**$_2$-learnable and not **TxtEx**-learnable. The corresponding notion has also been considered for function learning, but there the paradigms of explanatory and vacillatory learning coincide (Case & Smith, 1983).

Blum and Blum (1975), Case and Lynes (1982) and Case and Smith (1983) also considered allowing the final (or final sequence of) hypothesis to be anomalous; Blum and Blum (1975) considered $*$-anomalies and (Case & Lynes, 1982; Case & Smith, 1983) considered the general case. Here the final grammar for the input language may not be perfect, but may have up to a anomalies. A grammar n is a anomalous for L (written $W_n = {}^aL$) iff $\text{card}((L - W_n) \cup (W_n - L)) \leq a$. Here one also considers finite anomalies, denoted by $*$-anomalies, where $\text{card}(S) \leq *$ just means that S is finite. Thus, a learner M **TxtEx**^{a}-identifies a language L iff, for all texts T for all L, M on T converges to a hypothesis e such that $W_e = {}^aL$. One can similarly define **TxtBc**^{a}-learning criteria. It can be shown that

$$\textbf{TxtEx} = \textbf{TxtEx}^0 \subset \textbf{TxtEx}^1 \subset \textbf{TxtEx}^2 \subset \ldots \subset \textbf{TxtEx}^*$$

and

$$\textbf{TxtBc} = \textbf{TxtBc}^0 \subset \textbf{TxtBc}^1 \subset \textbf{TxtBc}^2 \subset \ldots \subset \textbf{TxtBc}^*.$$

Let $SD_n = \{L : W_{\min(L)} = {}^nL\}$. Then one can show (Case & Smith, 1983; Case & Lynes, 1982) that $SD_{n+1} \in \textbf{TxtEx}^{n+1} - \textbf{TxtEx}^n$. However, there is a trade-off between behaviourally correct learning and explanatory learning for learning with anomalies. On one hand, $\textbf{TxtBc} \notin \textbf{TxtEx}^*$, but on the other hand $\textbf{TxtEx}^{2n+1} \notin \textbf{TxtBc}^n$ and $\textbf{TxtEx}^{2n} \subseteq \textbf{TxtBc}^n$. However, for learning from informants, we have $\textbf{InfEx}^* \subseteq \textbf{InfBc}$ (Case and Lynes (1982) for the above results).

Consistent and Conservative Learning

Besides the above basic criteria of learning, researchers have also considered several properties that are useful for the learner to satisfy.

A learner M is said to be *consistent* on L iff for all texts T for L, $\text{ctnt}(T[n]) \subseteq W_{\text{M}(T[n])}$. That is, the learner's hypothesis is consistent with the data seen so far. There are three notions of consistency considered in the literature: (a) **TCons**, in which the learner is expected to be consistent on all inputs, irrespective of whether they represent some concept from the target class or not (Wiehagen and Liepe, 1976), (b) **Cons**, in which the learner is just expected to be consistent on the languages in the target class being learnt, though the learner may be inconsistent or even undefined on the input outside the target class (Bārzdiņš, 1974a), and (c) **RCons**, in which the learner is expected to be defined on all inputs, but required to be consistent only on the languages in the target class (Jantke & Beick, 1981). It can be shown that **TCons** $\subset$ **RCons** $\subset$ **Cons** $\subset$ **TxtEx** (Bārzdiņš, 1974a; Jantke and Beick, 1981; Wiehagen and Liepe, 1976).

A learner M is said to be *conservative* (Angluin, 1980) iff it does not change its mind unless the data contradicts its hypothesis. That is, M conservatively learns L iff for all texts T for L, if $\text{M}(T[n]) \neq \text{M}(T[n+1])$, then $\text{ctnt}(T[n+1]) \notin W_{\text{M}(T[n])}$. It can be shown that conservativeness is restrictive, that is there are classes of languages, which can be **TxtEx**-identified but not conservatively identified. An example of a class that can be identified explanatorily but not conservatively is the class containing all sets from *SDALL*, that is, the sets of the form $\{e, e+1, e+2, \ldots\}$, and all sets with minimum k_s and up to s elements where $k_0, k_1, k_2, \ldots$ is a recursive one-one enumeration of K. The general idea why this class is not conservatively learnable is that when the learner reads the data $e, e+1, e+2, \ldots$ it will, after some finite time based on data $e, e+1, e+2, \ldots, e+s$, output a conjecture which contains these data plus $e+s+1$; but conservative learning would then imply that $e \in K$ iff $e = k_r$ for some $r \leq s$, contradicting the non-recursiveness of K.

Monotonicity

Related notions to conservativeness are the various notions on monotonic learning that impose certain conditions on whether the previous hypothesis is a subset of the next hypothesis or not. The following notions are the three main ones.

- A learner M is said to be strongly monotonic (Jantke, 1991) on L iff for all texts T for L, $W_{\text{M}(T[n])} \subseteq W_{\text{M}(T[n+1])}$. Intuitively, strong monotonicity requires that the hypothesis of the learner grows
  with time.
- A learner M is said to be monotonic (Wiehagen, 1990) on L iff for all texts T for L, $W_{\text{M}(T[n])} \cap L \subseteq W_{\text{M}(T[n+1])} \cap L$. In monotonicity, the growth of the hypothesis is required only with respect to the language being learnt.
- A learner M is said to be weakly monotonic (Jantke, 1991) on L iff for all texts T for L, if $\text{ctnt}(T[n+1])) \subseteq$

$W_{M(T[n])}$, then $W_{M(T[n])} \subseteq W_{M(T[n+1])}$. That is, the learner behaves strongly monotonically, as long as the input data is consistent with the hypothesis.

An example for a strong monotonically learnable class is the class *SDALL*. When the learner currently conjectures $\{e, e+1, e+2, \ldots\}$ and it sees a datum $d < e$, then it makes a mind change to $\{d, d+1, d+2, \ldots\}$ which is a superset of the previous conjecture; it is easy to see that all mind changes are of this type. It can be shown that strong monotonic learning implies monotonic learning and weak monotonic learning, though monotonic learning and weak monotonic learning are incomparable (and thus both are proper restrictions of **TxtEx**-learning). For example, consider the class $\mathcal{C}$ consisting of the set $\{0, 2, 4, \ldots\}$ of all even numbers and, for each n, the set $\{0, 2, 4, \ldots, 2n\} \cup \{2n+1\}$ consisting of the even numbers below $2n$ and the odd number $2n+1$. Then, $\mathcal{C}$ is monotonically but not strong monotonically learnable.

Lange, Zeugmann, and Kapur (1992) also considered the dual version of the above criteria, where dual strong monotonicity learning of L requires that for all texts T for L, $W_{M(T[n])} \supseteq W_{M(T[n+1])}$; dual monotonicity requires that for all texts T for L, $W_{M(T[n])} \cap (\mathbb{N} - L) \supseteq W_{M(T[n+1])} \cap (\mathbb{N} - L)$; and dual weak monotonicity requires that if $\mathrm{ctnt}(T[n+1]) \subseteq W_{M(T[n])}$, then $W_{M(T[n])} \supseteq W_{M(T[n+1])}$.

In a similar fashion various other properties of learners have been considered. For example, reliability (Blum & Blum, 1975; Minicozzi, 1976) postulates that the learner does not converge on the input text unless it learns it, prudence (Fulk, 1990; Osherson, Stob and Weinsten, 1986) postulates that the learner outputs only indices of languages, which it also learns and confidence (Osherson, Stob and Weinsten, 1986) postulates that the learner converges on every text to some index, even if the text is for some language outside the class of languages to be learnt.

Indexed Families

Angluin (1980) initiated a study of learning indexed families of recursive languages. A class of languages (along with its indexing) $L_0, L_1, \ldots$ is an indexed family iff membership questions for the languages is uniformly decidable, that is, $x \in L_i$ can be recursively decided in x and i. Angluin gave an important characterization of indexed families that are **TxtEx**-learnable.

Suppose a class $\mathcal{L} = \{L_0, L_1, \ldots\}$ (along with the indexing) is given. Then, S is said to be a *tell-tale* (Angluin, 1980) of L_i iff S is finite and for all j, if $S \subseteq L_j$ and $L_j \subseteq L_i$, then $L_i = L_j$. It can be shown that for any class of languages that are learnable (in **TxtEx** or **TxtBc** sense), there exists a tell-tale for each language in the class. Moreover, Angluin showed that for indexed families, $\mathcal{L} = L_0, L_1, \ldots$, one can **TxtEx**-learn $\mathcal{L}$ iff one can recursively enumerate a tell-tale set for each L_i, effectively from i. Within the framework of learning indexed families, a special emphasis is given to the hypothesis space used; so the following criteria are considered for defining the learnability of a class $\mathcal{L}$ in dependence of the hypothesis space $\mathcal{H} = H_0, H_1, \ldots$. The class $\mathcal{L}$ is

- *Exactly learnable* iff there is a learner using the same hypothesis space as the given class, that is, $H_n = L_n$ for all n;
- *Class-preservingly learnable* iff there is a learner using a hypothesis space $\mathcal{H}$ with $\{L_0, L_1, \ldots\} = \{H_0, H_1, \ldots\}$ – here the order and the number of occurrences in the hypothesis space can differ, but the hypothesis space must consist of the same languages as the class to be learnt, and no other languages are allowed in the hypothesis space;
- *Class-comprisingly learnable* iff there is a learner using a hypothesis space $\mathcal{H}$ with $\{L_0, L_1, \ldots\} \subseteq \{H_0, H_1, \ldots\}$ – here the hypothesis space can also contain some further languages not in the class to be learnt and the learner does not need to identify these additional languages;
- *Prescribed learnable* iff for every hypothesis space $\mathcal{H}$ containing all the languages from $\mathcal{L}$ there is a learner for $\mathcal{L}$ using this hypothesis space;
- *Uniformly learnable* iff for every hypothesis space $\mathcal{H}$ with index e containing all the languages from $\mathcal{L}$ one can synthesize a learner M_e which succeeds to learn $\mathcal{L}$ using the hypothesis space $\mathcal{H}$.

Note that in all five cases $\mathcal{H}$ only ranges over indexed families. This differs from the standard case where $\mathcal{H}$ is an acceptable numbering of all recursively enumerable sets. We refer the reader to the survey of Lange, Zeugmann, and Zilles (2008) for an overview on work done on learning indexed families (**TxtEx**-learning,

learning under various properties of learners as well as characterizations of such learning criteria) and to (Jain, Stephan, & Ye, 2008; Lange & Zeugmann, 1993). While for explanatory learning and every class $\mathcal{L}$, all these five notions coincide, these notions turn out to be different for other learning notions like those of conservative learning, monotonic learning, and strong monotonic learning. For example, the class of all finite sets is not prescribed conservatively learnable: one can make an adversary hypothesis space where some indices contain large spurious elements, so that a learner is forced to do non-conservative mind change to obtain correct indices for the finite sets. The same example as above works for showing the limitations of prescribed learning for monotonic and strong monotonic learning.

The interested reader is referred to the textbook "Systems that Learn" (Jain, Osherson, Royer, & Sharma, 1999; Osherson, Stob and Weinsten, 1986) and the papers below as well as the references found in these papers for further reading. Complexity issues in inductive inference like the number of mind changes necessary to learn a class or oracles needed to learn some class can be found under the entries *Computational Complexity of Learning* and *Query-Based Learning*. The entry *Connections between Inductive Inference and Machine Learning* provides further information on this topic.

Cross References

►Connections Between Inductive Inference and Machine Learning

Recommended Reading

Angluin, D. (1980). Inductive inference of formal languages from positive data. *Information and Control, 45*, 117–135.

Bārzdiņš, J. (1974a). Inductive inference of automata, functions and programs. In *Proceedings of the international congress of mathematics*, Vancouver (pp. 771–776).

Bārzdiņš, J. (1974b). Two theorems on the limiting synthesis of functions. In *Theory of algorithms and programs* (Vol. 1., pp. 82–88). Latvian State University, Riga (In Russian).

Blum, L., & Blum, M. (1975). Toward a mathematical theory of inductive inference. *Information and Control, 28*, 125–155.

Case, J. (1999). The power of vacillation in language learning. *SIAM Journal on Computing, 28*, 1941–1969.

Case, J., & Lynes, C. (1982). Machine inductive inference and language identification. In M. Nielsen & E. M. Schmidt (Eds.), *Proceedings of the 9th international colloquium on automata, languages and programming*, Lecture Notes in Computer Science (Vol. 140., pp. 107–115). Heidelberg: Springer-Verlag.

Case, J., & Smith, C. (1983). Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science, 25*, 193–220.

Fulk, M. (1990). Prudence and other conditions on formal language learning. *Information and Computation, 85*, 1–11.

Gold, E. M. (1967). Language identification in the limit. *Information and Control, 10*, 447–474.

Jain, S., Osherson, D., Royer, J., & Sharma, A. (1999). *Systems that learn: An introduction to learning theory*. (2nd ed.). Cambridge: MIT Press.

Jain, S., Stephan, F., & Ye, N. (2008). Prescribed learning of indexed families. *Fundamenta Informaticae, 83*, 159–175.

Jantke, K. P. (1991). Monotonic and non-monotonic inductive inference. *New Generation Computing, 8*, 349–360.

Jantke, K. P., & Beick, H.-R. (1981). Combining postulates of naturalness in inductive inference. *Journal of Information Processing and Cybernetics (EIK), 17*, 465–484.

Lange, S., & Zeugmann, T. (1993). Language learning in dependence on the space of hypotheses. *Proceedings of the sixth annual conference on computational learning theory*, Santa Cruz, CA, (pp. 127–136).

Lange, S., Zeugmann, T., & Kapur, S. (1992). Class preserving monotonic language learning. *Tech. Rep. 14/92, GOSLER-Report*, FB Mathematik und Informatik, TH Leipzig.

Lange, S., Zeugmann, T., & Zilles, S. (2008). Learning indexed families of recursive languages from positive data: a survey. *Theoretical Computer Science, 397*, 194–232.

Minicozzi, E. (1976). Some natural properties of strong identification in inductive inference. *Theoretical Computer Science, 2*, 345–360.

Osherson, D., Stob, M., & Weinstein, S. (1986). *Systems that learn, an introduction to learning theory for cognitive and computer scientists*. Cambridge: Bradford–The MIT Press.

Osherson, D., & Weinstein, S. (1982). Criteria of language learning. *Information and Control*, 52, 123–138.

Sharma, A. (1998). A note on batch and incremental learnability. *Journal of Computer and System Sciences, 56*, 272–276.

Wiehagen, R. (1990). A thesis in inductive inference. In J. Dix, K. Jantke, & P. Schmitt (Eds.), *Nonmonotonic and inductive logic, 1st international workshop: Vol. 543 of Lecture notes in artificial intelligence* (pp. 184–207). Berlin: Springer-Verlag.

Wiehagen, R., & Liepe, W. (1976). Charakteristische Eigenschaften von erkennbaren Klassen rekursiver Funktionen. *Journal of Information Processing and Cybernetics (EIK), 12*, 421–438.

Inductive Inference

Choice of a model, theory, or hypothesis to express an apparent regularity or pattern in a body of data about many particular instances or events.

Inductive Inference Rules

►Logic of Generality

Inductive Learning

Synonyms
Statistical learning

Definition
Inductive learning is a subclass of machine learning that studies algorithms for learning knowledge based on statistical regularities. The learned knowledge typically has no deductive guarantees of correctness, though there may be statistical forms of guarantees.

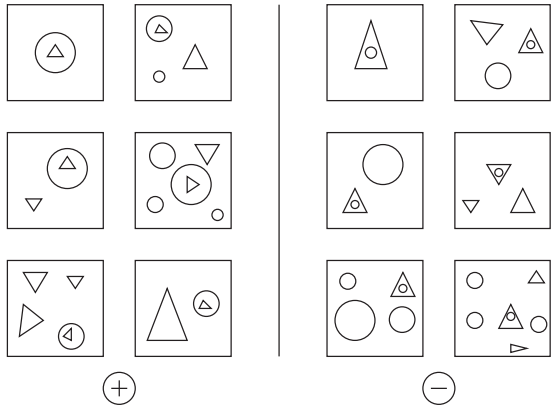
Inductive Logic Programming

Luc De Raedt
Katholieke Universiteit Leuven, Heverlee, Belgium

Synonyms
Learning in logic; Multi-relational data mining; Relational data mining; Relational learning

Definition
Inductive logic programming is the subfield of machine learning that uses ▶first-order logic to represent hypotheses and data. Because first-order logic is expressive and declarative, inductive logic programming specifically targets problems involving structured data and background knowledge. Inductive logic programming tackles a wide variety of problems in machine learning, including classification, regression, clustering, and reinforcement learning, often using "upgrades" of existing propositional machine learning systems. It relies on logic for knowledge representation and reasoning purposes. Notions of coverage, generality, and operators for traversing the space of hypotheses are grounded in logic, see also ▶logic of generality. Inductive logic programming systems have been applied to important applications in bio- and chemo-informatics, natural language processing, and web mining.

Motivation
The first motivation and most important motivation for using inductive logic programming is that it overcomes the representational limitations of attribute-value learning systems. Such systems employ a table-based representations where the instances correspond to rows in the table, the attributes to columns, and for each instance, a single value is assigned to each of the attributes. This is sometimes called the *single-table single-tuple* assumption. Many problems, such as the Bongard problem shown in Fig. 1, cannot elegantly be described in this format. Bongard (1970) introduced about a hundred concept-learning or pattern recognition problems, each containing six positive and six negative examples. Even though Bongard problems are toy-problems, they are similar to real-life problems such as structure–activity relationship prediction, where the goal is to learn to predict whether a given molecule (as represented by its 2D graph structure) is active or not. It is hard — if not, impossible — to squeeze this type of problem into the single-table single-tuple format for various reasons. Attribute-value learning systems employ a fixed number of attributes and also assume that these attributes are present in all of the examples. This assumption does not hold for the Bongard problems as the examples possess a variable number of objects (shapes). The singe-table single-tuple representation imposes an implicit order on the attributes,



Inductive Logic Programming. Figure 1. A complex classification problem: Bongard problem 47, developed by the Russian scientist Bongard (1970). It consists of 12 scenes (or examples), 6 of class ⊕ and 6 of class ⊖. The goal is to discriminate between the two classes

whereas there is no natural order on the objects in the Bongard problem. Finally, the relationships between the objects in the Bongard problem are essential and must be encoded as well. It is unclear how to do this within the single-table single-tuple assumption. First-order logic and relational representations allow one to encode problems involving multiple objects (or entities) as well as the relationships that hold them in a natural way.

The second motivation for using inductive logic programming is that it employs logic, a declarative representation. This implies that hypotheses are understandable and interpretable. By using logic, inductive logic programming systems are also able to employ background knowledge in the induction process. Background knowledge can be provided in the form of definitions of auxiliary relations or predicates that may be used by the learner. Finally, logic provides a well-understood theoretical framework for knowledge representation and reasoning. This framework is also useful for machine learning, in particular for defining and developing notions such as the covers relation, generality, and refinement operators, see also ▶logic of generality.

Theory

Inductive logic programming is usually defined as concept learning using logical representations. It aims at finding a hypothesis (a set of rules) that covers all positive examples and none of the negatives, while taking into account a background theory. This is typically realized by searching a space of possible hypotheses. More formally, the traditional inductive logic programming definition reads as follows:

Given

- a language describing hypotheses $\mathcal{L}_h$,
- a language describing instances $\mathcal{L}_i$,
- possibly a background theory B, usually in the form of a set of (definite) clauses,
- the *covers* relation that specifies the relation between $\mathcal{L}_h$ and $\mathcal{L}_i$, that is when an example e is covered (considered positive) by a hypothesis h, possibly taking into account the background theory B,
- a set of positive and negative examples $E = P \cup N$

Find a hypothesis $h \in \mathcal{L}_h$ such that for all $p \in P$: $covers(B, h, p) = true$ and for all $n \in N$: $covers(B, h, n) = false$.

This definition can, as for ▶concept-learning in general, be extended to cope with noisy data by relaxing the requirement that all examples be classified correctly.

There exist different ways to represent learning problems in logic, resulting in different learning settings. They typically use definite clause logic as the hypothesis language $\mathcal{L}_i$, but differ in the notion of an example. One can learn from entailment, from interpretations, or from proofs, cf. ▶logic of generality. The most popular setting is *learning from entailment*, where each example is a clause and $covers(B, h, e) = true$ if and only if $B \cup h \models e$.

The top leftmost scene in the Bongard problem of Fig.1 can be represented by the clause:

```
positive :- object(o1), object(o2),
            circle(o1), triangle(o2),
            in(o1, o2), large(o2).
```

The other scenes can be encoded in the same way. The following hypothesis then forms a solution to the learning problem:

```
positive :- object(X), object(Y),
                circle(X),
            triangle(Y), in(X,Y).
```

It states that those scenes having a circle inside a triangle are positive. For some more complex Bongard problems it could be useful to employ background knowledge. It could, for instance, state that triangles are polygons.

```
polygon(X) :- triangle(X).
```

Using this clause as background theory, an alternative hypothesis covering all positives and none of the negatives is

```
positive :- object(X), object(Y),
                circle(X),
            polygon(Y), in(X,Y).
```

An alternative for using long clauses as examples is to provide an identifier for each example and to add

the corresponding facts from the condition part of the clause to the background theory. For the above example, the following facts

```
object(e1,o1).
object(e1,o2).
circle(e1,o1).
triangle(e1,o2).
in(e1,o1,o2).
large(e1,o2).
```

would be added to the background theory and the positive example itself would then be represented through the fact `positive(e1)`, where `e1` is the identifier. The inductive logic programming literature typically employs this format for examples and hypotheses.

Whereas inductive logic programming originally focused on concept-learning – as did the whole field of machine learning – it is now being applied to virtually all types of machine learning problems, including regression, clustering, distance-based learning, frequent pattern mining, reinforcement learning, and even kernel methods and graphical models.

A Methodology

Many of the more recently developed inductive logic programming systems have started from an existing attribute-value learner and have upgraded it toward the use of first-order logic (Van Laer & De Raedt, 2001). By examining state-of-the-art inductive logic programming systems one can identify a methodology for realizing this (Van Laer and De Raedt, 2001). It starts from an attribute-value learning problem and system of interest, and takes the following two steps. First, the problem setting is upgraded by changing the representation of the examples, the hypotheses as well as the covers relation toward first-order logic. This step is essentially concerned with defining the learning setting, and possible settings to be considered include the already mentioned learning from *entailment*, *interpretations*, and *proofs* settings. Once the problem is clearly defined, one can attempt to formulate a solution. Thus the second step adapts the original algorithm to deal with the upgraded representations. While doing so, it is advisable to keep the changes as minimal as possible. This step often involves the modification of the operators used to traverse the search space. Different operators for realizing this are introduced in the entry on the ▶logic of generality.

There are many reasons why following the methodology is advantageous. First, by upgrading a learner that is already effective for attribute-value representations, one can benefit from the experiences and results obtained in the propositional setting. In many cases, for instance decision trees, this implies that one can rely on well-established methods and findings, which are the outcomes of several decades of machine learning research. It will be hard to do better starting from scratch. Second, upgrading an existing learner is also easier than starting from scratch as many of the components (such as heuristics and search strategy) can be recycled. It is therefore also economic in terms of man power. Third, the upgraded system will be able to emulate the original one, which provides guarantees that the output hypotheses will perform well on attribute-value learning problems. Even more important is that it will often also be able to emulate extensions of the original systems. For instance, many systems that extend frequent item-set mining toward using richer representations, such as sequences, intervals, the use of taxonomies, graphs, and so on, have been developed over the past decade. Many of them can be emulated using the inductive logic programming upgrade of Apriori (Agrawal, Mannila, Srikant, Toivonen & Verkamo, 1996) called Warmr (Dehaspe & Toivonen, 2001). The upgraded inductive logic programming systems will typically be more flexible than the systems it can emulate but typically also less efficient because there is a price to be paid for expressiveness. Finally, it may be possible to incorporate new features in the attribute-value learner by following the methodology. One feature that is often absent from propositional learners and may be easy to incorporate is the use of a background theory.

It should be mentioned that the methodology is not universal, that is, there exist also approaches, such as Muggleton's Progol (Muggleton, 1995), which have directly been developed in first-order logic and for which no propositional counter part exists. In such cases, however, it can be interesting to follow the inverse methodology, which would specialize the inductive logic programming system.

FOIL: An Illustration

One of the simplest and best-known inductive logic programming systems is FOIL (Quinlan, 1990). It can be regarded as an upgrade of a rule-learner such as CN2 (Clark & Niblett, 1989). FOIL's problem setting is an instance of the learning from entailment setting introduced above (though it restricts the background theory to ground facts only and does not allow functors).

Like most rule-learning systems, FOIL employs a separate-and-conquer approach. It starts from the empty hypothesis, and then repeatedly searches for one rule that covers as many positive examples as possible and no negative example, adds it to the hypothesis, removes the positives covered by the rule, and then iterates. This process is continued until all positives are covered. To find one rule, it performs a hill-climbing search through the space of clauses ordered according to generality. The search starts at the most general rule, the one stating that all examples are positive, and then repeatedly specializes it. Among the specializations it then selects the best one according to a heuristic evaluation based on information gain. A heuristic, based on the minimum description length principle, is then used to decide when to stop specializing clauses.

The key differences between FOIL and its propositional predecessors are the representation and the operators used to compute the specializations of a clause. It employs a refinement operator under θ-subsumption (Plotkin, 1970) (see also ▶logic of generality). Such an operator essentially refines clauses by adding atoms to the condition part of the clause or applying substitutions to a clause. For instance, the clause

```
positive :- triangle(X), in(X,Y),
            color(X,C).
```

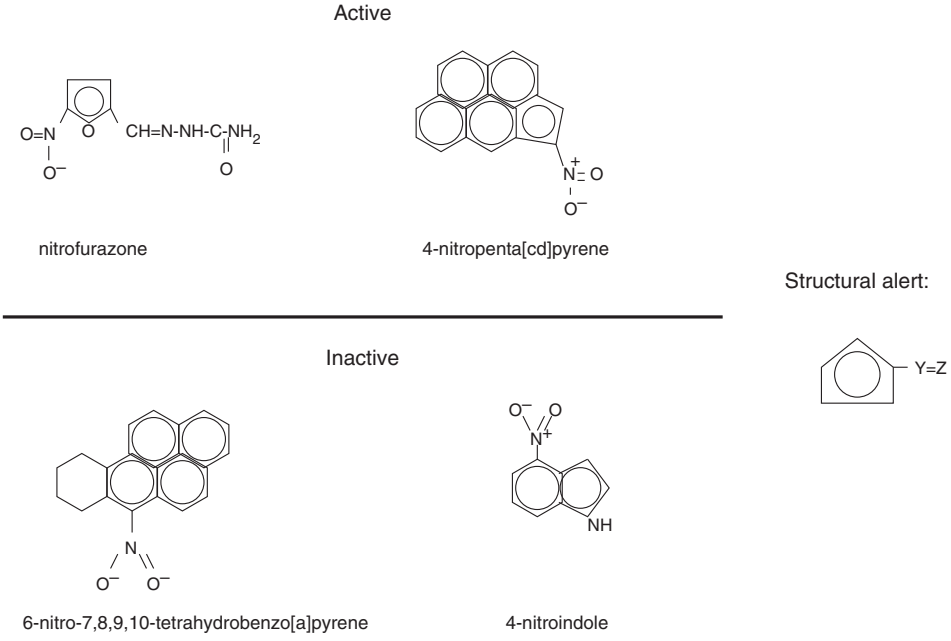
can be specialized to

```
positive :- triangle(X), in(X,Y),
            color(X,red).
positive :- triangle(X), in(X,Y),
            color(X,C), large(X).
positive :- triangle(X), in(X,Y),
            color(X,C),
           rectangle(Y).
...
```

The first specialization is obtained by substituting the variable `C` by the constant `red`, the other two by adding an atom (`large(X)`, `rectangle(Y)`, respectively) to the condition part of the rule. Inductive logic programming systems typically also employ syntactic restrictions – the so-called – that specify which clauses may be used in hypotheses. For instance, in the above example, the second argument of the `color` predicate belongs to the type *Color*, whereas the arguments of `in` are of type *`Object`* and consist of object identifiers.

Application

Inductive logic programming has been successfully applied to many application domains, including bio- and chemo-informatics, ecology, network mining, software engineering, information retrieval, music analysis, web mining, natural language processing, toxicology, robotics, program synthesis, design, architecture, and many others. The best-known applications are in scientific domains. For instance, in structure–activity relationship prediction, one is given a set of molecules together with their activities, and background knowledge encoding functional groups, that is particular components of the molecule, and the task is to learn rules stating when a molecule is active or inactive. This is illustrated in Fig. 2 (after Srinivasan, Muggleton, Sternberg, and King (1996)), where two molecules are active and two are inactive. One then has to find a pattern that discriminates the actives from the inactives. Structure–activity relationship prediction (SAR) is an essential step in, for instance, drug discovery. Using the general purpose inductive logic programming system Progol (Muggleton, 1995) *structural alerts*, such as that shown in Fig. 2, have been discovered. These alerts allow one to distinguish the actives from the inactives – the one shown in the figure matches both of the actives but none of the inactives – and at the same time they are readily interpretable and provide useful insight into the factors determining the activity. To solve structure–activity relationship prediction problems using inductive logic programming one must represent the molecules and hypotheses using the logical formalisms introduced above. The resulting representation is very similar to that employed in the Bongard



Inductive Logic Programming. Figure 2. Predicting mutagenicity (Srinivasan et al., 1996)

problems: the objects are the atoms and relationships the bonds. Particular functional groups are encoded as background predicates.

State-of-the-Art

The upgrading methodology has been applied to a wide variety of machine learning systems and problems. There exist now inductive logic programming systems that

- induce logic programs from examples under various learning settings. This is by far the most popular class of inductive logic programming systems. Well-known systems include Aleph (Srinivasan, 2007) and Progol (Muggleton, 1995) as well as various variants of FOIL (Quinlan, 1990). Some of these systems, especially Progol and Aleph, contain many features that are not present in propositional learning systems. Most of these systems focus on a classification setting, and learn the definition of a *single* predicate.
- induce logical decision trees from examples. These are binary decision trees containing conjunctions of atoms (i.e., queries) as tests. If a query succeeds, then one branch is taken, else the other one. Decision tree methods for both classification and regression exist (see Blockeel & De Raedt, 1998; Kramer & Widmer, 2001).
- mine for frequent queries, where queries are conjunctions of atoms. Such queries can be evaluated on an example. For instance, in the Bongard problem, the query `?- triangle (X), in (X, Y)` succeeds on the leftmost scenes, and fails on the rightmost ones. Therefore, its frequency would be 6. The goal is then to find all queries that are frequent, that is, whose frequencies exceed a certain threshold. Frequent query mining upgrades the popular local pattern mining setting due to Agrawal et al. (1996) to inductive logic programming (see Dehaspe & Toivonen, 2001).
- learn or revise the definitions of theories, which consist of the definitions of multiple predicates, at the same time (cf. Wrobel, 1996), and the entry ▶Theory revision in this encyclopedia. Several of these systems have their origin in the model inference system by Shapiro (1983) or the work by Angluin (1987).

Current Trends and Challenges

There are two major trends and challenges in inductive logic programming. The first challenge is to extend

the inductive logic programming paradigm beyond the purely symbolic one. Important trends in this regard include

- the combination of inductive logic programming principles with graphical and probabilistic models for reasoning about uncertainty. This is a field known as *statistical relational learning*, *probabilistic logic learning*, or *probabilistic inductive logic programming*. At the time of writing, this is a very popular research stream, attracting a lot of attention in the wider artificial intelligence community, cf. the entry ►Statistical Relational Learning in this encyclopedia. It has resulted in many relational or logical upgrades of well-known graphical models including Bayesian networks, Markov networks, hidden Markov models, and stochastic grammars.
- the use of relational distance measures for classification and clustering (Kirsten, Wrobel, & Horvath, 2001; Ramon & Bruynooghe, 1998). These distances measure the similarity between two examples or clauses, while taking into account the underlying structure of the instances. These distances are then combined with standard classification and clustering methods such as k-nearest neighbor and k-means.
- the integration of relational or logical representations in reinforcement learning, known as ►relational reinforcement learning (Džeroski, De Raedt, & Driessens, 2001).

The power of inductive logic programming is also its weakness. The ability to represent complex objects and relations and the ability to make use of background knowledge add to the computational complexity. Therefore, a key challenge of inductive logic programming is tackling this added computational complexity. Even the simplest method for testing whether one hypothesis is more general than another – that is θ-subsumption (Plotkin, 1970) – is NP-complete. Similar tests are used for deciding whether a clause covers a particular example in systems such as FOIL. Therefore, inductive logic programming and relational learning systems are computationally much more expensive than their propositional counterparts. This is an instance of the expressiveness versus efficiency trade-off in computer science. Because of these computational difficulties, inductive logic programming has devoted a lot of attention to efficiency issues. On the theoretical side, there exist various results about the polynomial learnability of certain subclasses of logic programs (cf. Cohen & Page, 1995, for an overview). From a practical perspective, there is quite some work on developing efficient methods for searching the hypothesis space and especially for evaluating the quality of hypotheses. Many of these methods employ optimized inference engines based on Prolog or database technology or constraint-satisfaction methods (cf. Blockeel & Sebag, 2003 for an overview).

Cross References

►Multi-Relational Data Mining

Recommended Reading

A comprehensive introduction to inductive logic programming can be found in the book by De Raedt (2008) on logical and relational learning. Early surveys of inductive logic programming are contained in Muggleton and De Raedt (1994) and Lavrač and Džeroski (1994) and an account of its early history is provided in Sammut (1993). More recent collections on current trends can be found in the proceedings of the annual *Inductive Logic Programming Conference* (published in Springer's *Lectures Notes in Computer Science Series*) and special issues of the *Machine Learning Journal*. An interesting collection of inductive logic programming and multi-relational data mining works are provided in Džeroski and Lavrač (2001). The upgrading methodology is described in detail in Van Laer and De Raedt (2001). More information on logical issues in inductive logic programming are given in the entry ►logic of generality in this encyclopedia, whereas the entries ►statistical relational learning and ►graph mining are recommended for those interested in frameworks tackling similar problems using other types of representations.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307–328). Cambridge, MA: MIT Press.

Angluin, D. (1987). Queries and concept-learning. *Machine Learning, 2*, 319–342.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence, 101*(1–2), 285–297.

Blockeel, H., & Sebag, M. (2003). Scalability and efficiency in multi-relational data mining. *SIGKDD Explorations, 5*(1), 17–30.

Bongard, M. (1970). *Pattern recognition*. New York: Spartan Books.

Clark, P., & Niblett, T. (1989). The CN2 algorithm. *Machine Learning, 3*(4), 261–284.

Cohen, W. W., & Page, D. (1995). Polynomial learnability and inductive logic programming: Methods and results. *New Generation Computing, 13*, 369–409.

De Raedt, L. (2008). *Logical and relational learning*. Springer.

Dehaspe, L., & Toivonen, H. (2001). Discovery of relational association rules. In S. Džeroski & N. Lavrač (Eds.), *Relational data mining* (pp. 189–212). Berlin/Heidelberg: Springer.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning, 43*(1/2), 5–52.

Džeroski, S., & Lavrač, N. (Eds.). (2001). Relational data mining. Springer.

Kirsten, M., Wrobel, S., & Horvath, T. (2001). Distance based approaches to relational learning and clustering. In S. Džeroski and N. Lavrač (Eds.), *Relational data mining* (pp. 213–232). Berlin/Heidelberg: Springer.

Kramer, S., & Widmer, G. (2001). Inducing classification and regression trees in first order logic. In S. Džeroski and N. Lavrač (Eds.), *Relational data mining* (pp. 140–159). Berlin/Heidelberg: Springer.

Lavrač, N., & Džeroski, S. (1994). *Inductive logic programming: techniques and applications*. Chichester, UK: Ellis Horwood.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, 13*, 245–286.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19*(20), 629–679.

Plotkin, G. D. (1970). A note on inductive generalization. In *Machine Intelligence* (vol. 5, pp. 153–163). Edinburgh, Scotland: Edinburgh University Press.

Quinlan, J. R. (1990).
Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Ramon, J., & Bruynooghe, M. (1998). A framework for defining distances between first-order logic objects. In D. Page (Ed.), *Proceedings of the eighth international conference on inductive logic programming. Lecture notes in artificial intelligence*, (vol. 1446, pp. 271–280). Berlin/Heidelberg: Springer.

Sammut, C. (1993). The origins of inductive logic programming: A prehistoric tale. In S. Muggleton (Ed.), *Proceedings of the third international workshop on inductive logic programming* (pp. 127–148). Ljubljana: J. Stefan Institute.

Shapiro, E. Y. (1983). Algorithmic program debugging. MIT Press.

Srinivasan, A. The Aleph Manual, 2007. URL: http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html.

Srinivasan, A., Muggleton, S., Sternberg, M. J. E., & King, R. D. (1996). Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence, 85*(1/2), 277–299.

Van Laer, W., & De Raedt, L. (2001). How to upgrade propositional learners to first order logic: A case study. In S. Džeroski and N. Lavrač (Eds.), *Relational data mining*, (pp. 235–261). Berlin/Heidelberg: Springer.

Wrobel, S. (1996). First-order theory refinement. In L. De Raedt (Ed.), *Advances in inductive logic programming. Frontiers in artificial intelligence and applications* (vol. 32, pp. 14–33). Amsterdam: IOS Press.

Inductive Process Modeling

Ljupčo Todorovski
University of Ljubljana, Ljubljana, Slovenia

Synonyms

Process-based modeling

Definition

Inductive process modeling is a machine learning task that deals with the problem of learning quantitative *process models* from ▶time series data about the behavior of an observed dynamic system. Process models are models based on ordinary differential equations that add an explanatory layer to the equations. Namely, scientists and engineers use models to both predict and explain the behavior of an observed system. In many domains, models commonly refer to processes that govern system dynamics and entities altered by those processes. Ordinary differential equations, often used to cast models of dynamic systems, offer one way to represent these mechanisms and can be used to simulate and predict the system behavior, but fail to make the processes and entities explicit. In response, process models tie the explanatory information about processes and entities to the mathematical formulation, based on equations, that enables simulation.

Table 1 shows a process model for a predator–prey interaction between foxes and rabbits. The three processes explain the dynamic change of the concentrations of both species (represented in the model as two *population* entities) through time. The *rabbit_growth* process states that the reproduction of rabbit is limited by the fixed environmental capacity. Similarly, the *fox_death* process specifies an unlimited exponential mortality function for the fox population. Finally, the *fox_rabbit_predation* process refers to the predator–prey interaction between foxes and rabbits that states that the prey concentration decreases and the predator one increases proportionally with the sizes of the two populations. The process model makes the structure of the model explicit and transparent to scientists; while at the same time it can be easily transformed in to a system of two differential equations by additively combining the equations for the time derivatives of the system

Inductive Process Modeling. Table 1 A Process Model of Predatory–Prey Interaction between Foxes and Rabbits. The Notation $d[X,t]$ Indicates the Time Derivative of Variable X.

```
model predation;
  entities fox{population}, rabbit{population};
  process rabbit_growth;
    entites rabbit;
    equations d[rabbit.conc,t] = 1.81 * rabbit.conc *
    (1 – 0.0003 * rabbit.conc);
  process fox_death;
    entites fox;
    equations d[fox.conc,t] = –1.04 * fox.conc;
  process fox_rabbit_predation;
  entities fox, rabbit;
    equations
    d[fox.conc,t] = 0.03 * rabbit.conc * fox.conc;
    d[rabbit.conc,t] = –1 * 0.3 * rabbit.conc * fox.conc;
```

variables *fox.conc* and *rabbit.conc*. Given initial values for these variables, one can simulate the equations to produce trajectories that correspond to the population dynamics through time.

The processes from Table 1 instantiate more general generic processes, that can be used for modeling any ecological system. For example:

```
generic process predation;
 entities Predator{population}, Prey{population};
 parameters ar[0.01, 10], ef[0.001, 0.8];
 equations
  d[Predator.conc,t] = ef * ar * Prey.conc * Predator.conc;
  d[Prey.conc,t] = –1 * ar * Prey.conc * Predator.conc;
```

is a general form of the *fox_rabbit_predation* process from the example model in Table 1. Note that in the generic process, the parameters are replaced with numeric ranges and the entities with identifiers of generic entities (i.e., *Predator* and *Prey* are identifiers that refer to instances of the generic entity *population*).

Having defined entities and processes on an example, one can define the task of inductive process modeling as: Given

- Time series observations for a set of numeric system variables as they change through time
- A set of entities that the model might include
- Generic processes that specify casual relations among entities
- Constraints that determine plausible relations among processes and entities in the model

Find a specific process model that explains the observed data and the simulation of which closely matches observed time series.

There are two approaches for solving the task of inductive process modeling. The first is the transformational approach that transforms the given knowledge about entities, processes, and constraints to ▶language bias for equation discovery and uses the Lagramge method for ▶equation discovery in turn (Todorovski & Džeroski, 1997, 2007). The second approach performs search through the space of candidate process models to find the one that matches the given time series data best.

Inductive process modeling methods IPM (Bridewell, Langley, Todorovski, & Džeroski, 2008) and HIPM (Todorovski, Bridewell, Shiran, & Langley, 2005) follow the second approach. IPM is a naïve method that exhaustively searches the space of candidate process models following the ▶learning as search paradigm. The search space of candidate process models is defined by the sets of generic processes and of entities in the observed system specified by the user. IPM first matches the type of each entity against the types of entities involved in each generic process and produces a list of all possible instances of that generic process. For example, the generic process *predation*, from the example above, given two *population* entities *fox* and *rabbit*, can be instantiated in four different ways (*fox_fox_predation*, *fox_rabbit_predation*, *rabbit_fox_predation*, and *rabbit_rabbit_predation*). The IPM search procedure collects the set of all possible instances of all the generic processes and uses them as a set of candidate model components. In the search phase, all combinations of these model components are being matched against observed ▶time series. The

matching involves the employment of gradient-descent methods for nonlinear optimization to estimate the optimal values of the process model parameters. As output, IPM reports the process models with the best match.

Trying out all components' combinations is prohibitive in many situations since it obviously leads to combinatorial explosion. HIPM employs constraints that limit the space of combinations by ruling-out implausible or forbidden combinations. Examples of such constraints in the predator–prey example above include rules that a proper process model of population dynamics should include a single growth and a single mortality process per species, the predator–prey process should relate two different species, and different predator–prey interaction should refer to different population pairs. HIPM specifies the rules in a hierarchy of generic processes where each node in the hierarchy specifies a rule for proper combination/selection of process instances.

Cross References

►Equation Discovery

Recommended Reading

Bridewell, W., Langley, P., Todorovski, L., & Džeroski, S. (2008). Inductive process modeling. *Machine Learning*, *71*(1), 1–32.

Todorovski, L., Bridewell, W., Shiran, O., & Langley, P. (2005). Inducing hierarchical process models in dynamic domains. In M.M. Veloso & S. Kambhampati (Eds.), *Proceedings of the twentieth national conference on artificial intelligence*, Pittsburgh, PA, USA.

Todorovski, L., & Džeroski, S. (1997). Declarative bias in equation discovery. In D.H. Fisher (Ed.), *Proceedings of the fourteenth international conference on machine learning*, Nashville, TN, USA.

Todorovski, L., & Džeroski, S. (2007). Integrating domain knowledge in equation discovery. In S. Džeroski & L. Todorovski (Eds.), *Computational discovery of scientific knowledge. LNCS* (Vol. 4660). Berlin: Springer.

Inductive Program Synthesis

►Inductive Programming

Inductive Programming

Pierre Flener[1,2], Ute Schmid[3]
[1]Sabancı University, Orhanlı, Tuzla, İstanbul, Turkey
[2]Uppsala University, Uppsala, Sweden
[3]University of Bamberg, Feldkirchenstr. Bamberg, Germany

Synonyms

Example-based programming; Inductive program synthesis; Inductive synthesis; Program synthesis from examples

Definition

Inductive programming is the inference of an algorithm or program featuring recursive calls or repetition control structures, starting from information that is known to be incomplete, called the *evidence*, such as positive and negative input–output examples or clausal constraints. The inferred program must be correct with respect to the provided evidence, in a ►generalization sense: it should neither be equivalent nor inconsistent to it. Inductive programming is guided explicitly or implicitly by a ►language bias and a ►search bias. The inference may draw on background knowledge or query an oracle. In addition to ►induction, ►abduction may be used. The restriction to algorithms and programs featuring recursive calls or repetition control structures distinguishes inductive programming from ►concept learning or ►classification.

This chapter is restricted to the inference of declarative programs, whether functional or logic, and dispense with repetition control structures in the inferred program in favour of recursive calls.

Motivation and Background

Inductive program synthesis is a branch of the field of *program synthesis*, which addresses a cognitive question as old as computers, namely the understanding of the human act of computer programming, to the point where a computer can be made to help in this task (and ultimately to enhance itself). See Flener (2002) for a recent survey; the other main branches of program synthesis are based on deductive inference,

namely *constructive program synthesis* and *transformational program synthesis*. In such *deductive program synthesis*, the provided information, called the *specification*, is assumed to be complete (in contrast to inductive program synthesis where the provided information is known to be incomplete), and the presence of repetitive or recursive control structures in the synthesized program is not imposed.

Research on the inductive synthesis of recursive *functional* programs started in the early 1970s and was brought onto firm theoretical foundations with the seminal THESYS system of Summers (1977) and work of Biermann (1978), where all the evidence is handled non-incrementally (see ►incremental learning). Essentially, the idea is first to infer computation *traces* from input–output examples (►instances), and then to use a ►trace-based programming method to fold these traces into a recursive program. The main results till the mid 1980s were surveyed in Smith (1984). Due to limited progress with respect to the range of programs that could be synthesized, research activities decreased significantly in the next decades. However, a new approach that formalizes functional program synthesis in the term-rewriting framework and that allows the synthesis of a broader class of programs than the classical approaches is pursued in Kitzelmann and Schmid (2006).

The advent of *logic* programming brought a new elan but also a new direction in the early 1980s, especially due to the MIS system of Shapiro (1983), eventually spawning the new field of ►inductive logic programming (ILP). Most of this ILP work addresses a wider class of problems, as the focus is *not* only on recursive logic programs: more adequate designations are inductive ►theory revision and *declarative program debugging*, as an additional input is a possibly empty initial theory or program that is incrementally revised or debugged according to each newly presented piece of evidence, possibly in the presence of background knowledge or an oracle. The main results on the inductive synthesis of recursive logic programs were surveyed in Flener and Yılmaz (1999).

Structure of Learning System

The core of an inductive programming system is a mechanism for constructing a recursive generalization for a set of input/output examples (instances), say. Although vocabulary of logic programming is used, this method also covers the synthesis of functional programs.

The input, often a set of input/output examples, is called the *evidence*. Further evidence may be queried from an *oracle*. Additional information, in the form of predicate symbols that can be used during the synthesis, can be provided as *background knowledge*. Since the ►hypothesis space – the set of legal recursive programs – is infinite, a ►language bias is introduced. One particularly useful and common approach in inductive programming is to provide a statement bias by means of a *program schema*.

The evidential synthesis of a recursive program starts from the provided evidence for some predicate symbol and works essentially as follows. A program schema is chosen to provide a template for the program structure, where all yet undefined predicate symbols must be instantiated during the synthesis. Predefined predicate symbols of the background knowledge are then chosen for some of these undefined predicate symbols in the template. If it is deemed that the remaining undefined predicate symbols cannot all be instantiated via purely structural generalization by non-recursive definitions, then the method is recursively called to infer recursive definitions for some of them (this is called ►predicate invention and amounts to shifting the *vocabulary bias*); otherwise the synthesis ends successfully right away. This generic method can backtrack to any choice point for synthesizing alternative programs.

In the rest of this section, this basic terminology of inductive programming discussed more precisely. In the next section, instantiations of this generic method by some well-known methods are presented.

The Evidence and the Oracle

The evidence is often limited to ground positive examples of the predicate symbols that are to be defined. Ground negative examples are convenient to prevent overgeneralization, but should be used constructively and not just to reject candidate programs. A useful generalization of ground examples is evidence in the form of a set of (non-recursive) clauses, as variables and additional predicate symbols can then be used.

Example 1 The $delOdds(L, R)$ relation, which holds if and only if R is the integer list L without its odd elements, can be incompletely described by the following clausal evidence:

$$\begin{aligned} delOdds([\,],[\,]) &\leftarrow true \\ delOdds([X],[\,]) &\leftarrow odd(X) \\ delOdds([X],[X]) &\leftarrow \neg odd(X) \\ delOdds([X,Y],[Y]) &\leftarrow odd(X),\ \neg odd(Y) \\ delOdds([X,Y],[X,Y]) &\leftarrow \neg odd(X),\ \neg odd(Y) \\ false &\leftarrow delOdds([X],[X]), \\ &\quad\ odd(X) \end{aligned} \tag{1}$$

The first clause is a ground positive example, whereas the second and third clauses generalize the infinity of ground positive examples, such as $delOdds([5],[\,])$ and $delOdds([22],[22])$, for handling singleton lists, while the fourth and fifth clauses summarize the infinity of ground positive examples for handling lists of two elements, the second one being even: these clauses make explicit the underlying filtering relation (*odd*) that is *intrinsic* to the problem at hand but cannot be provided via ground examples and would otherwise have to be guessed. The sixth clause summarizes an infinity of ground negative examples for handling singleton lists, namely where the only element of the list is odd but not filtered.

In some methods, especially for the induction of functional programs, the first n positive input–output examples with respect to the underlying data type are presented (e.g., for linear lists, what to do with the empty list, with a one-element list, up to a list with three elements); because of this ordering of examples, no explicit presentation of negative examples is then necessary.

Inductive program synthesis should be monotonic in the evidence (more evidence should never yield a less complete program, and less evidence should not yield a more complete program) and should not be sensitive to the order of presentation of the evidence.

Program Schemas

Informally, a *program schema* (Smith, 1985) contains a template program and a set of axioms. The *template* abstracts a class of actual programs, called *instances*, in the sense that it represents their dataflow and control-flow by means of place-holders, but does not make explicit all their actual computations nor all their actual data structures. The *axioms* restrict the possible instances of the place-holders and define their interrelationships. Note that a schema is problem-independent. A ▶first-order-logic approach is taken and templates are considered as open logic programs (programs where some place-holder predicate symbols are left undefined, or *open*; a program with no open predicate symbols is said to be *closed*) and axioms as first-order specifications of these open predicate symbols.

Example 2 Most methods of inductive synthesis are biased by program schemas whose templates have clauses of the forms in the following generic template:

$$r(X,Y,Z) \leftarrow c(X,Y,Z), p(X,Y,Z)$$

$$\begin{aligned} r(X,Y,Z) \leftarrow\ & d(X,H,X_1,\ldots,X_t,Z), \\ & r(X_1,Y_1,Z),\ \ldots,\ r(X_t,Y_t,Z), \\ & q(H,Y_1,\ldots,Y_t,Z,Y) \end{aligned} \tag{2}$$

where c, d, p, and q are open predicate symbols, X is a non-empty sequence of terms, and Y, Z are possibly empty sequences of terms. The intended semantics of this generic template can be described informally as follows. For an arbitrary relation r over parameters X, Y, and Z, an instance of this generic template is to determine the values of *result parameter* Y corresponding to a given value of *induction parameter* X, considering the value of *auxiliary parameter* Z. Two cases arise: either the c test succeeds and X has a value for which Y can easily be directly computed through p, or X has a value for which Y cannot be so easily directly computed and the *divide-and-conquer principle* is applied:

1. *Divide* X through d into a term H and t terms $X_1,\ldots,X_t$ of the same type as X but smaller than X according to some well-founded relation;
2. *Conquer* through t recursive calls to r to determine the values of $Y_1,\ldots,Y_t$ corresponding to $X_1,\ldots,X_t$, respectively, considering the value of Z;

3. *Combine* through q the terms $H, Y_1, \ldots, Y_t, Z$ to build Y.

Enforcing this intended semantics must be done manually, as any instance template by itself has no semantics, in the sense that any program is an instance of it (it suffices to define c by a program that always succeeds, and p by the given program). One way to do this is to attach to a template some axioms (see Smith (1985) for the divide-and-conquer axioms), namely the set of specifications of its open predicate symbols: these specifications refer to each other, including the one of r, and are generic (because even the specification of r is unknown), but can be manually abduced (see ►abduction) once and for all according to the informal semantics of the schema.

Predicate Invention

Another important language bias is the available vocabulary, which is here the set of predicate symbols mentioned in the evidence set, or actually defined in the background knowledge (and possibly mentioned by the oracle). If an inductive synthesis fails, other than backtracking to a different program schema (i.e., shifting the statement bias), one can try and shift the vocabulary bias by inventing new predicate symbols and inducing programs for them in the extended vocabulary; this is also known as performing ►constructive induction. Only the invention of recursively defined predicate symbols is *necessary*, as a non-recursive definition of a predicate symbol can be eliminated by substitution (under ►resolution) for its calls in the ►induced program (even though that might make the program longer).

In general, it is undecidable whether predicate invention is necessary to induce a finite program in the vocabulary of its evidence and background knowledge (as a consequence of Rice's theorem, 1953), but introducing new predicate symbols always allows the induction of a finite program (as a consequence of a result by Kleene), as shown in Stahl (1995). The necessity of shifting the vocabulary bias can only be decided for some restricted languages (but the bias shift attempt might then be unsuccessful), so in practice one often has to resort to heuristics. Note that an inductive synthesiser of recursive algorithms may be recursive itself: it may recursively invoke itself for a necessary new predicate symbol.

Other than the decision problem, the difficulties of predicate invention are as follows. First, adequate formal parameters for a new predicate symbol have to be identified among all the variables in the clause using it. This can be done instantaneously by using precomputations done manually once and for all at the template level. Second, evidence for a new predicate symbol has to be abduced from the current program using the evidence for the old predicate symbol. This usually requires an oracle for the old predicate symbol, whose program is still unfinished at that moment and cannot be used. Third, the abduced evidence may be less numerous than for the old predicate symbol (note that if the new predicate symbol is in a recursive clause, then no new evidence might be abduced from the old evidence that is covered by the base clauses) and can be quite sparse, so that the new synthesis is more difficult. This *sparseness problem* can be illustrated by an example.

Example 3 Given the positive ground examples $factorial(0,1)$, $factorial(1,1)$, $factorial(2,2)$, $factorial(3,6)$, $factorial(4,24)$, and given the still open program:

$$\begin{aligned} factorial(N,F) &\leftarrow N = 0,\ F = 1 \\ factorial(N,F) &\leftarrow add(M,1,N),\ factorial(M,G), \\ &\qquad\quad product(N,G,F) \end{aligned}$$

where *add* is known but *product* was just invented (and named so only for the reader's convenience), the abduceable examples are $product(1,1,1)$, $product(2,1,2)$, $product(3,2,6)$, and $product(4,6,24)$, which is hardly enough for inducing a recursive program for *product*; note that there is one less example than for *factorial*. Indeed, examples such as $product(3,6,18)$, $product(2,6,12)$, $product(1,6,6)$, etc., are missing, which puts the given examples more than one resolution step apart, if not on different resolution paths. This is aggravated by the absence of an oracle for the invented predicate symbol, which is not necessarily intrinsic to the task at hand (although *product* actually is intrinsic to *factorial*).

Background Knowledge

In an inductive programming context, background knowledge is particularly important, as the inference of recursive programs is more difficult than the inference of ▶classifiers. For the efficiency of synthesis, it is crucial that this collection of definitions of the pre-defined predicate symbols be *annotated* with information about the *types* of their arguments and about whether some *well-founded relation* is being enforced between some of their arguments, so that semantically suitable instances for the open predicate symbols of any chosen program schema can be readily spotted. (This requires in turn that the types of the arguments of the predicate symbols in the provided evidence are declared as well.) The background knowledge should be problem-independent, and an inductive programming method should be able to perform *knowledge mobilisation*, namely organizing it dynamically according to relevance to the current task.

In data-driven, analytical approaches, background knowledge is used in combination with ▶explanation-based learning (EBL) methods, such as abduction (see Example 4) or systematic rewriting of input/output examples into computational traces (see Example 5).

Programs and Data

Example 4 The DIALOGS (Dialogue-based Inductive-Abductive LOGic program Synthesiser) method (Flener, 1997) is interactive. The main design objective was to take all extra burden from the specifier by having the method ask for exactly and only the information it needs, default answers being provided wherever possible. As a result, no evidence needs to be prepared in advance, as the method invents its own candidate evidence and queries the oracle about it, with an opportunity to declare (at the oracle/specifier's risk) that enough information has been provided. All answers by the oracle are stored as *judgements*, to prevent asking the same query twice. This is suitable for all levels of expertise of human users, as the queries are formulated in the specifier's initially unknown conceptual language, in a way such that the specifier must know the answers if she really feels the need for the wanted program. The method is schema-biased, and the current implementation has two schemas. The template of the *divide-and-conquer* schema has the generality of the generic template (2). The template of the *accumulate* schema extends this by requiring an accumulator in the sequence Z of auxiliary parameters. The evidence language (▶observation language) is (non-recursive) logic programs with negation. Type declarations are provided as a language bias. The program language (▶hypothesis language) is recursive logic programs with negation, with possibly multiple base cases and recursive cases.

For instance, starting from the empty program for the relation *delOdds* in Example 1, the algorithm design choices of using the *divide-and-conquer* schema with R as result parameter and L as induction parameter decomposed (with $t = 1$) through head-tail decomposition by d lead to the following intermediate open program:

$$\begin{aligned} delOdds(L,R) &\leftarrow c(L,R),\ p(L,R)\\ delOdds(L,R) &\leftarrow d(L,H,L_1),\ delOdds(L_1,R_1),\\ &\qquad q(H,R_1,R)\\ c(\_,\_) &\leftarrow true\\ d(L,H,T) &\leftarrow L = [H|T] \end{aligned} \tag{3}$$

The first five evidential clauses for *delOdds* in (1) are then implicitly interactively acquired from the oracle/specifier by the following question & answer dialogue, leading the specifier to reveal the intrinsic predicate symbol *odd*:

DIALOGS: When does $delOdds([\,],R)$ hold?

Specifier: If $R = [\,]$.

DIALOGS: When does $delOdds([X],R)$ hold?

Specifier: If $(odd(X)$ *and* $R = [\,])$
or $(\neg odd(X)$ *and* $R = [X])$.

DIALOGS: When does $delOdds([X,Y],R)$ hold,
assuming $odd(Y)$?

Specifier: If $(odd(X)$ *and* $R = [\,])$
or $(\neg odd(X)$ *and* $R = [X])$.

DIALOGS: When does $delOdds([X,Y],R)$ hold, assuming $\neg odd(Y)$?

Specifier: If $(odd(X)$ *and* $R = [Y])$ *or* $(\neg odd(X)$ *and* $R = [X,Y])$.

Next, abduction infers the following evidence set for the still open predicate symbols p and q:

$$\begin{aligned}
p([\,],[\,]) &\leftarrow true\\
p([X],[\,]) &\leftarrow odd(X)\\
&\quad\; q(X,[\,],[\,]) \leftarrow odd(X)\\
p([X],[X]) &\leftarrow \neg odd(X)\\
&\quad\; q(X,[\,],[X]) \leftarrow \neg odd(X)\\
p([X,Y],[Y]) &\leftarrow odd(X),\ \neg odd(Y)\\
&\quad\; q(X,[Y],[Y]) \leftarrow odd(X)\\
p([X,Y],[X,Y]) &\leftarrow \neg odd(X),\ \neg odd(Y)\\
&\quad\; q(X,[Y],[X,Y]) \leftarrow \neg odd(X)
\end{aligned}$$

From this, induction infers the following closed programs for p and q:

$$\begin{aligned}
p([\,],[\,]) &\leftarrow true\\
q(H,L,[H|L]) &\leftarrow \neg odd(H)\\
q(H,L,L) &\leftarrow odd(H)
\end{aligned} \tag{4}$$

The final closed program is the union of the programs (3) and (4), as no predicate invention is deemed necessary. Sample syntheses with predicate invention are presented in Flener (1997) and Flener and Yılmaz (1999).

Example 5 The THESYS method (Summers, 1977) was one of the first methods for the inductive synthesis of functional (Lisp) programs. Although it has a rather restricted scope, it can be seen as the methodological foundation of many later methods for inducing functional programs. The non-interactive method is schema-biased, and the implementation has two schemas. Upon adaptation to functional programming, the template of the *linear recursion* schema is the instance of the generic template (2) obtained by having X as a sequence of exactly one induction parameter and Z as the empty sequence of auxiliary parameters, and by dividing X into $t = 1$ smaller value X_t, so that there is only $t = 1$ recursive call. The template of the *accumulate* schema extends this by having Z as a sequence of exactly one auxiliary parameter, playing the role of an accumulator. The evidence language (observation language) is sets of ground positive examples. The program language (hypothesis language) is recursive functional programs, with possibly multiple base cases, but only one recursive case. The only primitive functions are *nil*, *cons*, *head*, *tail*, and *empty*, because the implementation is limited to the list datatype, inductively defined by $list \equiv nil \mid cons(x, list)$, under the axioms $empty(nil) = true$, $head(cons(x,y)) = x$, and $tail(cons(x,y)) = y$. There is no function invention.

For instance, from the following examples of a list unpacking function:

$$\begin{aligned}
unpack(nil) &= nil\\
unpack((A)) &= ((A))\\
unpack((A\ B)) &= ((A)\ (B))\\
unpack((A\ B\ C)) &= ((A)\ (B)\ (C))
\end{aligned}$$

the abduced traces are:

$$\begin{aligned}
&empty(X) && \rightarrow nil\\
&empty(tail(X)) && \rightarrow cons(X, nil)\\
&empty(tail(tail(X))) && \rightarrow\\
&\qquad cons(cons(head(X), nil), cons(tail(X), nil))\\
&empty(tail(tail(tail(X)))) && \rightarrow\\
&\qquad cons(cons(head(X), nil),\\
&\qquad cons(cons(head(tail(X)), nil),\\
&\qquad cons(tail(tail(X)), nil)))
\end{aligned}$$

and the induced program is:

$$
\begin{array}{lll}
unpack(X) & \leftarrow & \\
empty(X) & \rightarrow & nil, \\
empty(tail(X)) & \rightarrow & cons(X, nil), \\
true & \rightarrow & cons(cons(head(X), nil), \\
 & & \quad unpack(tail(X)))
\end{array}
$$

A modern extension of THESYS is the IGOR method (Kitzelmann & Schmid, 2006). The underlying program template describes the set of all functional programs with the following restrictions: built-in functions can only be first-order, and no nested or mutual recursion is allowed. IGOR adopts the two-step approach of THESYS. Synthesis is still restricted to structural problems, where only the structure of the arguments matters, but not their contents, such as in list reversing. Nevertheless, the scope of synthesisable programs is considerably larger. For instance, tree-recursive functions and functions with hidden parameters can be induced. Most notably, programs consisting of a calling function and an arbitrary set of further recursive functions can be induced. The first step of synthesis (trace construction) is therefore expanded such that traces can contain nestings of conditions. The second step is expanded such that the synthesis of a function can rely on the invention and synthesis of other functions (that is, IGOR uses a technique of function invention in correspondence to the concept of predicate invention introduced above). An extension, IGOR2, relies on constructor-term rewriting techniques. The two synthesis steps are merged into one and make use of background knowledge. Therefore, the synthesis of programs for semantic problems, such as list sorting, becomes feasible.

Applications

In the framework of *software engineering*, inductive programming is defined as the inference of information that is pertinent to the construction of a generalized computational system for which the provided evidence is a representative sample (Flener & Partridge, 2001). In other words, inductive programming does *not* have to be a panacea for software development in-the-large and infer a complete software system in order to be useful: it suffices to induce, for instance, a self-contained system module while programming in-the-small, problem features and decision logic for specification acquisition and enhancement, or support for debugging and testing. Inductive programming is then not always limited to programs with repetitive or recursive control structures. There are opportunities for synergy with manual programming and deductive program synthesis, as there are sometimes system modules that no one knows how to specify in a complete way, or that are harder to specify or program in a complete way, and yet where incomplete information such as input-output examples is readily available. More examples and pointers to the literature are given in Flener (2002, Section 5) and Flener and Partridge (2001).

In the context of *end-user programming*, inductive programming methods can be used to enable non-expert users to take advantage of the more sophisticated functionalities offered by their software. This kind of application is in the focus of ▶programming by demonstration (PBD).

Finally, it is worth having an evidential synthesiser of recursive algorithms invoked by a more general-purpose machine learning method when necessary predicate invention is detected or conjectured, as such general methods require a lot of evidence to infer reliably a recursively defined hypothesis.

Future Directions

Inductive programming is still mainly a topic of basic research, exploring how the intellectual ability of humans to infer generalized recursive procedures from incomplete evidence can be captured in the form of synthesis methods. Already a variety of promising methods are available. A necessary step should be to compare and analyse the current methods. A first extensive comparison of different ILP methods for inductive programming was presented some years ago (Flener & Yılmaz, 1999). An up-to-date analysis should take into account not only ILP methods but also methods for the synthesis of functional programs, using classical (Kitzelmann & Schmid, 2006) as well as evolutionary

(Olsson, 1995) methods. The methods should be compared with respect to the required quantity of evidence, the kind and amount of background knowledge, the scope of programs that can be synthesized, and the efficiency of synthesis. Such an empirical comparison should result in the definition of characteristics that describe concisely the scope, usefulness, and efficiency of the existing methods in different problem domains.

Since only a few inductive programming methods can deal with semantic problems, it should be useful to investigate how inductive programming methods can be combined with other machine learning methods, such as kernel-based classification.

Finally, the existing methods should be adapted to a broad variety of application areas in the context of programming assistance, as well as in other domains where recursive data structures or recursive procedures are relevant.

Acknowledgment

Most of the work by Pierre Flener was done while on leave of absence in 2006/07 as a Visiting Faculty Member and Erasmus Exchange Teacher at Sabancı University.

Cross References

►Explanation-Based Learning
►Inductive Logic Programming
►Programming by Demonstration
►Trace-Based Programming

Websites

- Online Platform of the Inductive Programming Co- mmunity: http://www.inductiveprogramming.org/.
- Flener, P., & Partridge, D. (2001). Inductive programming. *Automated Software Engineering, 8*(2), 131–137. http://user.it.uu.se/~pierref/ase/.
- *Workshops on Approaches and Applications of Inductive Programming* (*AAIP 2005, AAIP 2007, and AAIP 2009*): http://www.cogsys.wiai.uni-bamberg.de/aaip/.
- *Journal of Machine Learning Research*, *Special Topic on Approaches and Applications on Inductive Programming*, February/March 2006: http://jmlr.csail.mit.edu/papers/topic/inductive_programming.html.
- *Tutorial on Automatic Inductive Programming* at ICML 2006: http://www.evannai.inf.uc3m.es/et/icml06/aiptutorial.htm.

Recommended Reading

Biermann, A. W. (1978). The inference of regular LISP programs from examples. *IEEE Transactions on Systems, Man, and Cybernetics, 8*(8), 585–600.

Flener, P. (1997). Inductive logic program synthesis with DIALOGS. In S. H. Muggleton, (Ed.), *Revised selected papers of the 6th international workshop on inductive logic programming (ILP 1996)*, volume 1314 of *lecture notes in artificial intelligence* (pp. 175–198). Berlin: Springer.

Flener, P. (2002). Achievements and prospects of program synthesis. In A. Kakas & F. Sadri (Eds.), *Computational logic: Logic programming and beyond; essays in honour of Robert A. Kowalski*, volume 2407 of *lecture notes in artificial intelligence* (pp. 310–346). Berlin: Springer.

Flener, P., & Partridge, D. (2001). Inductive programming. *Automated Software Engineering, 8*(2), 131–137.

Flener, P., & Yılmaz S. (1999). Inductive synthesis of recursive logic programs: achievements and prospects. *Journal of Logic Programming, 41*(2–3), 141–195.

Kitzelmann, E., & Schmid, U. (2006). Inductive synthesis of functional programs – An explanation based generalization approach. *Journal of Machine Learning Research, 7*, 429–454.

Olsson, J. R. (1995). Inductive functional programming using incremental program transformation. *Artificial Intelligence, 74*(1), 55–83.

Shapiro, E. Y. (1983). *Algorithmic program debugging*. Cambridge, MA: MIT Press.

Smith, D. R. (1984). The synthesis of LISP programs from examples: A survey. In A. W. Biermann, G. Guiho, & Y. Kodratoff (Eds.), *Automatic program construction techniques* (pp. 307–324). New York: Macmillan.

Smith, D. R. (1985). Top-down synthesis of divide-and-conquer algorithms. *Artificial Intelligence, 27*(1), 43–96.

Stahl, I. (1995). The appropriateness of predicate invention as bias shift operation in ILP. *Machine Learning, 20*(1–2), 95–117.

Summers, P. D. (1977). A methodology for LISP program construction from examples. *Journal of the ACM, 24*(1), 161–175.

Inductive Synthesis

►Inductive Programming

Inductive Transfer

Ricardo Vilalta[1], Christophe Giraud-Carrier[2], Pavel Brazdil[3], Carlos Soares[3]
[1]University of Houston, Houston TX, USA
[2]Brigham Young University, UT, USA
[3]University of Porto, Porto, Portugal

Synonyms

Transfer of knowledge across domains

Definition

Inductive transfer refers to the ability of a learning mechanism to improve performance on the current task after having learned a different but related concept or skill on a previous task. Transfer may additionally occur between two or more learning tasks that are being undertaken concurrently. Transfer may include background knowledge or a particular form of ▶search bias.

As an illustration, an application of inductive transfer arises in competitive games involving teams of robots (e.g., Robocup Soccer). In this scenario, transferring knowledge learned from one task into another task is crucial to acquire skills necessary to beat the opponent team. Specifically, imagine a situation where a team of robots has been taught to keep a soccer ball away from the opponent team. To achieve that goal, robots must learn to keep the ball, pass the ball to a close teammate, etc., always trying to remain at a safe distance from the opponents. Now let us assume that we wish to teach the same team of robots to play a different game where they must learn to score against a team of defending robots. Knowledge gained during the first activity can be transferred to the second one. Specifically, a robot can prefer to perform an action learned in the past over actions proposed during the current task because the past action has a significant higher merit value. For example, a robot under the second task may learn to recognize that it is preferable to shoot than to pass the ball because the goal is very close. This action can be learned from the first task by recognizing that the precision of a pass is contingent on the proximity of the teammate.

Structure of the System

The main idea behind a learning architecture using knowledge transfer is to produce a source model from which knowledge can be extracted and transferred to a target model. This allows for multiple scenarios (Brazdil, Giraud-Carrier, Soares, & Vilalta, 2009; Pratt & Thrun, 1997). For example, the target and source models can be trained at different times such that the transfer takes place after the source model has been trained; in this case there is an explicit form of knowledge transfer, also called *representational transfer*. In contrast, we use the term *functional transfer* to denote the case where two or more models are trained simultaneously; in this case the models share (part of) their internal structure during learning (see Neural Networks below). When the transfer of knowledge is explicit, we denote the case as *literal transfer* when the source model is left intact. In addition, we denote the case as *nonliteral transfer* when the source model is modified before knowledge is transferred to the target model; in this case some processing step takes place on the model before it is used to initialize the target model.

Neural Networks

A learning paradigm amenable to test the feasibility of knowledge transfer is that of neural networks (Caruana, 1993). A popular form of knowledge transfer is effected through multitask learning, where the output nodes in the multilayer network represent more than one task. In such a scenario, internal nodes are shared by different tasks dynamically during learning. As an illustration, consider the problem of learning to classify astronomical objects from images mapping the sky into multiple classes. One task may be in charge of classifying a star into several classes (e.g., main sequence, dwarf, red giant, neutron, pulsar, etc.). Another task can focus on galaxy classification (e.g., spiral, barred spiral, elliptical, irregular, etc.). Rather than separating the problem into different tasks where each task is in charge of identifying one type of luminous object, one can combine the tasks together into a single parallel multitask problem where the hidden layer of a neural network shares patterns that are common to all classification

tasks (see Fig. 1). The reason explaining why learning often improves in accuracy and speed in this context is that training with many tasks in parallel on a single neural network induces information that accumulates in the training signals; if there exists properties common to several tasks, internal nodes can serve to represent common subconcepts simultaneously.

Other Paradigms

Knowledge transfer can be performed using other learning and data-analysis paradigms such as ▶kernel methods, probabilistic methods (see ▶Bayesian Methods) and ▶clustering (Evgeniou, Micchelli, & Pontil, 2005; Raina, Ng, & Koller, 2006). For example, inductive transfer can take place in learning methods that assume a probabilistic distribution of the data by guaranteeing a form of relatedness among the distributions adopted across tasks (Raina et al.). As an illustration, if learning to classify both stars and galaxies assumes a mixture of normal densities to model the example-class distribution, one can force both distributions to have sets of parameters that are as similar as possible while preserving good generalization performance. In that case, shared knowledge can be interpreted as a set of assumptions about the data distribution for all tasks under analysis. The knowledge transfer concept is also related to the problem of introducing new intermediate concepts in the process of bottom–up induction of rules. In the inductive logic programming (ILP) setting, this is referred to as *predicate invention* (Stahl, 1995).

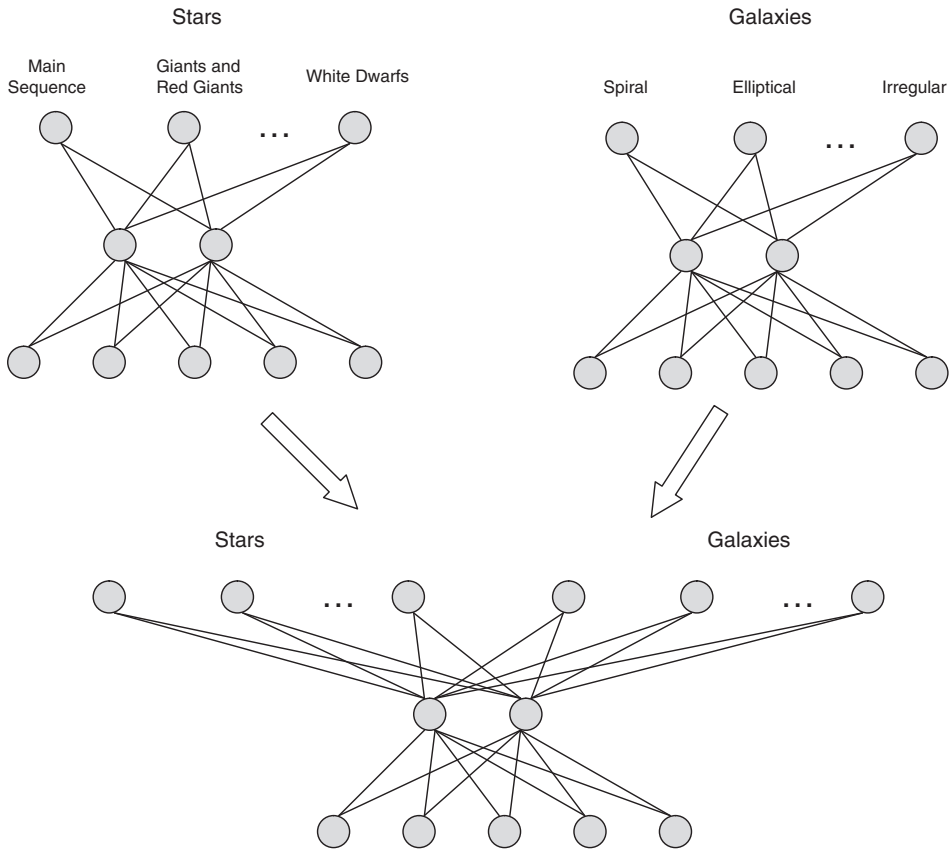
Metasearching for Problem Solvers

A different research direction in inductive transfer explores complex scenarios where the software architecture itself evolves with experience (Schmidhuber, 1997). The main idea is to divide a program into different components that can be reused during different stages of the learning process. As an illustration, one can work within the space of (self-delimiting binary) programs to propose an optimal ordered problem solver. The goal is to solve a sequence of problems, deriving one solution after the other, as optimally as possible; ideally, the system should be capable of exploiting previous solutions and incorporate them into the solution to the current problem. This can be done by allocating computing time to the search for previous solutions that, if useful, become transformed into building blocks. We assume that the current problem can be solved by copying or invoking previous pieces of code (i.e., building blocks or knowledge). In that case the mechanism will accept those solutions with substantial savings in computational time.

Theoretical Work

Several studies have provided a theoretical analysis of the case where a learner uses experience from previous tasks to learn a new task. This process is often referred to as metalearning. The aim is to understand the conditions under which a learning algorithm can provide good generalizations when embedded in an environment made of related tasks. Although the idea of knowledge transfer is normally made implicit in the analysis, it is clear that the metalearner extracts and exploits knowledge on every task to perform well on future tasks. Theoretical studies fall within a Bayesian model and within a Probably Approximately Correct (PAC) model. The idea is to find not only the right hypothesis in a hypothesis space (base learning), but in addition, to find the right hypothesis space in a family of hypothesis spaces (metalearning).

Let us review the main ideas behind these studies (Baxter, 2000). We begin by assuming that the learner is embedded in a set of related tasks that share certain commonalities. Going back to the problem where a learner is designed for the recognition of astronomical objects, the idea is to classify objects (e.g., stars, galaxies, nebulae, planets) extracted from images mapping certain region of the sky. One way to transfer learning experience from one astronomical center to another is by sharing a metalearner that carries a bias toward recognition of astronomical objects. In traditional learning, we assume a probability distribution **p** that indicates which examples are more likely to be seen in such task. Now we assume there is a more general distribution **P** over the space of all possible distributions. In essence, the metadistribution **P** indicates which tasks are more likely to be found within the sequence of tasks faced by the metalearner (just as an example distribution **p** indicates which examples are more likely to be seen in one task). In our example, the metadistribution **P** peaks over tasks corresponding to classification of astronomical objects. Given a family of hypothesis spaces {**H**}, the goal of the



Inductive Transfer. Figure 1. Example of multitask learning on astronomical images

metalearner is to find a hypothesis space $\mathbf{H}^*$ that minimizes a functional risk corresponding to the expected loss of the best possible hypothesis in each hypothesis space. In practice, since we ignore the form of $\mathbf{P}$, we need to draw samples $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_n$ to infer how tasks are distributed in our environment. To summarize, in the transfer learning scenario our input is made of samples $\mathbf{T} = \{\mathbf{T}_i\}$, where each sample $\mathbf{T}_i$ is composed of examples. The goal of the metalearner is to output a hypothesis space with an ►inductive bias that generates accurate models for a new task.

Future Directions

The research community faces several challenges on how to efficiently transfer knowledge across tasks. One challenge involves devising learning architectures with an explicit representation of knowledge about models and algorithms, i.e., metaknowledge. Most systems that integrate knowledge transfer mechanisms make an implicit assumption about the transfer process by modifying the bias embedded by the hypothesis space. For example, we may change bias by selecting a learning algorithm that draws linear boundaries over the input space instead of one that draws quadratic boundaries; here, no explicit knowledge is transferred specifying our preference for linear boundaries. Because of this limitation, transferring knowledge across domains becomes problematic.

Another challenge is to understand why a learning algorithm performs well or not on certain datasets and to use that (meta)knowledge to improve its performance. Recent work in metalearning has explored the idea that high-quality dataset characteristics or metafeatures provide enough information to differentiate the performance of a given set of learning algorithms. From a practical perspective, a proper characterization of datasets leads to an interesting goal: the construction of metalearning assistants. The main role of these assistants is to recommend a good predictive

model given a new dataset, or to attempt to modify the learning mechanism before it is invoked again in a dataset drawn from a similar distribution.

Cross References

►Metalearning

Recommended Reading

Baxter, J. (2000). A model of inductive learning bias. *Journal of Artificial Intelligence Research, 12*, 149–198.

Brazdil, P., Giraud-Carrier, C., Soares, C., & Vilalta, R. (2009). *Metalearning: Applications to data mining*. Springer-Verlag Berlin: Heidelberg.

Caruana, R. (1993). Multitask learning: A knowledge-based source of inductive bias. In P. E. Utgoff (Ed.), *Proceedings of the tenth international conference on machine learning* (pp. 41–48). San Mateo, Springer Netherlands: Morgan Kaufmann.

Dai, W., Yang, Q., Xue, G., & Yu, Y. (2007). Boosting for transfer learning. In *Proceedings of the 24th annual international conference on machine learning* (pp. 193–200). New York: ACM.

Evgeniou, T., Micchelli, C. A., & Pontil, M. (2005). Learning multiple tasks with kernel methods. *Journal of Machine Learning Research, 6*, 615–637.

Mihalkova, L., Huynh, T., & Mooney, R. J. (2007). Mapping and revising Markov logic networks for transfer learning. In *Proceedings of the 22nd AAAI conference on artificial intelligence* (pp. 608–614). Vancouver, BC: AAAI Press.

Oblinger, D., Reid, M., Brodie, M., & de Salvo Braz, R. (2002). Cross-training and its application to skill-mining. *IBM Systems Journal, 41*(3), 449–460.

Pratt, L., & Thrun, S. (1997). Second special issue on inductive transfer. *Machine Learning, 28*, No. 1, 5–130.

Raina, R., Ng, A. Y., & Koller, D. (2006). Constructing informative priors using transfer learning. In *Proceedings of the twenty-third international conference on machine learning* (pp. 713–720). Pittsburgh, PA: ACM.

Reid, M. (2004). Improving rule evaluation using multitask learning. In *Proceedings of the 14th international conference on ILP* (pp. 252–269). Springer-Verlag, Heidelberg.

Schmidhuber, J., Zhao, J., & Wiering M. A. (1997). Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning, 28*(1), 105–130.

Stahl, I. (1996). Predicate invention in inductive logic programming. In L. De Raedt (Ed.), *Advances in inductive logic programming*. (pp. 34–47). IOS Press.

Inequalities

►Generalization Bounds

Information Retrieval

Information retrieval (IR) is a set of techniques that extract from a collection of documents those that are relevant to a given query. Initially addressing the needs of librarians and specialists, the field has evolved dramatically with the advent of the World Wide Web. It is more general than *data retrieval*, whose purpose is to determine which documents contain occurrences of the keywords that make up a query. Whereas the syntax and semantics of data retrieval frameworks is strictly defined, with queries expressed in a totally formalized language, words from a natural language given no or limited structure are the medium of communication for information retrieval frameworks. A crucial task for an IR system is to index the collection of documents to make their contents efficiently accessible. The documents retrieved by the system are usually ranked by expected relevance, and the user who examines some of them might be able to provide feedback so that the query can be reformulated and the results improved.

Information Theory

►Minimum Description Length Principle
►Minimum Message Length

In-Sample Evaluation

Synonyms

Within-sample evaluation

Definition

In-sample evaluation is an approach to ►algorithm evaluation whereby the learned model is evaluated on the data from which it was learned. This provides a biased estimate of learning performance, in contrast to ►holdout evaluation.

Cross References

►Algorithm Evaluation

Instance

Synonyms
Case; Example; Item; Object

Definition
An *instance* is an individual object from the universe of discourse. Most ►learners create a ►model by analyzing a ►training set of instances. Most ►machine learning models take the form of a function from an ►instance space to an output space. In ►attribute-value learning, each instance is often represented as a vector of ►attribute values, each position in the vector corresponding to a unique attribute.

Instance Language

►Observation Language

Instance Space

Synonyms
Example space; Item space; Object space

Definition
An *instance space* is the space of all possible ►instances for some ►learning task. In ►attribute-value learning, the instance space is often depicted as a geometric space, one dimension corresponding to each attribute.

Instance-Based Learning

Eamonn Keogh
University of California, Riverside, CA, USA

Synonyms
Analogical reasoning; Case-based learning; Memory-based; Nearest neighbor methods; Non-parametric methods

Definition
Instance-based learning refers to a family of techniques for ►**classification** and ►**regression**, which produce a class label/predication based on the similarity of the query to its nearest neighbor(s) in the training set. In explicit contrast to other methods such as ►**decision trees** and ►**neural networks**, instance-based learning algorithms do not create an abstraction from specific instances. Rather, they simply store all the data, and at query time derive an answer from an examination of the query's ►**nearest neighbor**(s).

Somewhat more generally, instance-based learning can refer to a class of procedures for solving new problems based on the solutions of similar past problems.

Motivation and Background
Most instance-based learning algorithms can be specified by determining the following four items:

1. Distance measure: Since the notion of similarity is being used to produce class label/prediction, we must explicitly state what similarity/distance measure to use. For real-valued data, Euclidean distance is a popular choice and may be optimal under some assumptions.
2. Number of neighbors to consider: It is possible to consider any number from one to all neighbors. This number is typically denoted as k.
3. Weighting function: It is possible to give each neighbor equal weight, or to weight them based on their distance to the query.
4. Mapping from local points: Finally, some method must be specified to use the (possibly weighted) neighbors to produce an answer. For example, for regression the output can be the weighted mean of the k nearest neighbors, or for classification the output can be the majority vote of the k nearest neighbors (with some specified tie-breaking procedure).

Since instance-based learning algorithms defer all the work until a query is submitted, they are sometimes called lazy algorithms (in contrast to eager learning algorithms, such as decision trees). Beyond the setting of parameters/distance measures/mapping noted above, one of the main research issues with instance-based learning algorithms is mitigating their expensive classification time, since a naïve algorithm would require comparing the distance for the query to every point in the database. Two obvious solutions

are indexing the data to achieve a sublinear search, and numerosity reduction (data editing) (Wilson & Martinez, 2000).

Further Reading

The best distance measure to use with an instance-based learning algorithms is the subject of active research. For the special case of time series data alone, there are at least one hundred methods Ding, Trajcevski, Scheuermann, Wang, & Keogh (2008). Conferences such as ICML, SIGKDD, etc. typically have several papers each year which introduce new distance measures and/or efficient search techniques.

Recommended Reading

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*, 37–66.

Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., & Keogh, E. J. (2008). Querying and mining of time series data: Experimental comparison of representations and distance measures. *PVLDB, 1*(2), 1542–1552.

Wilson, D. R., & Martinez, T. R. (2000). Reduction techniques for exemplar-based learning algorithms. *Machine Learning, 38*(3), 257–286.

Instance-Based Reinforcement Learning

William D. Smart
Washington University in St. Louis,
St. Louis, MO, USA

Synonyms

Kernel-based reinforcement learning

Definition

Traditional reinforcement-learning (RL) algorithms operate on domains with discrete state spaces. They typically represent the value function in a table, indexed by states, or by state–action pairs. However, when applying RL to domains with continuous state, a tabular representation is no longer possible. In these cases, a common approach is to represent the value function by storing the values of a small set of states (or state–action pairs), and interpolating these values to other, unstored, states (or state–action pairs). This approach is known as instance-based reinforcement learning (IBRL). The instances are the explicitly stored values, and the interpolation is typically done using well-known instance-based supervised learning algorithms.

Motivation and Background

Instance-Based Reinforcement Learning (IBRL) is one of a set of value-function approximation techniques that allow standard RL algorithms to deal with problems that have continuous state spaces. Essentially, the tabular representation of the value function is replaced by an instance-based supervised learning algorithm and the rest of the RL algorithm remains unaltered. Instance-based methods are appealing because each stored instance can be viewed as analogous to one cell in the tabular representation. The interpolation method of the instance-based learning algorithm then blends the value between these instances.

IBRL allows generalization of value across the state (or state–action) space. Unlike tabular representations it is capable of returning a value approximation for states (or state–action pairs) that have never been directly experienced by the system. This means that, in theory, fewer experiences are needed to learn a good approximation to the value function and, hence, a good control policy. IBRL also provides a more compact representation of the value function than a table does. This is especially important in problems with multi-dimensional continuous state spaces. A straightforward discretization of such a space results in an exponential number of table cells. This, in turn, leads to an exponential increase in the amount of training experiences needed to obtain a good approximation of the value function.

An additional benefit of IBRL over other value-function approximation techniques, such as artificial neural networks, is the ability to bound the predicted value of the approximation. This is important, since it allow us to retain some of the theoretical non-divergence results for tabular representations.

Structure of Learning System

IBRL can be used to approximate both the state value function and the state–action value function. For problems with discrete actions, it is common to store a separate value function for each action. For continuous

actions, the (continuous) state and action vectors are often concatenated, and VFA is done over this combined domain. For clarity, we will discuss only the state value function here, although our comments apply equally well to the state–action value function.

The Basic Approach

IBRL uses an instance-based supervised learning algorithm to replace the tabular value function representation of common RL algorithms. It maintains a set of states, often called basis points, and their associated values, using them to provide a value-function approximation for the entire state space. These exemplar states can be obtained in a variety of ways, depending on the nature of the problem. The simplest approach is to sample, either regularly or randomly, from the state space. However, this approach can result in an unacceptably large number of instances, especially if the state space is large, or has high dimension. A better approach is to use states encountered by the learning agent as it follows trajectories in the state space. This allows the representational power of the approximation algorithm to be focused on areas of the space in which the learning agent is likely to be. This, too, can result in a large number of states, if the agent is long-lived. A final approach combines the previous two by sub-sampling from the observed states.

Each stored instance state has a value associated with it, and an instance-based supervised learning algorithm is used to calculate the value of all other states. While any instance-based algorithm can be used, kernel-based algorithms have proven to be popular. Algorithms such as locally weighted regression (Smart & Kaelbling, 2000), and radial basis function networks (Kretchmar & Anderson, 1997) are commonly seen in the literature. These algorithms make some implicit assumptions about the form of the value function and the underlying state space, which we discuss below. For a state s, the kernel-based value-function approximation $V(s)$ is

$$V(s) = \frac{1}{\eta} \sum_{i=1}^{n} \phi(s, s_i)\, V(s_i), \tag{1}$$

where the s_i values are the n stored basis points, η is a normalizer,

$$\eta = \sum_{i=1}^{n} \phi(s, s_i), \tag{2}$$

and ϕ is the kernel function. A common choice for ϕ is an exponential kernel,

$$\phi(s, t) = e^{\frac{(s-t)^2}{\sigma^2}}, \tag{3}$$

where σ is the kernel bandwidth. The use of kernel-based approximation algorithms is well motivated, since they respect Gordon's non-divergence conditions (Gordon, 1995), and also Szepesvári and Smart's convergence criteria (Szepesvári & Smart, 2004).

As the agent gathers experience, the value approximations at each of the stored states and, optionally, the location and bandwidth of the states must be updated. Several techniques, often based on the temporal difference error, have been proposed, but the problem remains open. An alternative to on-line updates is a batch approach, which relies on storing the experiences generated by the RL agent, composing these into a discrete MDP, solving this MDP exactly, and then using supervised learning techniques on the states and their associated values. This approach is known as fitted value iteration (Szepesvári & Munos, 2005).

Examples of IBRL Algorithms

Several IBRL algorithms have been reported in the literature. Kretchmar and Anderson (1997) presented one of the first IBRL algorithms. They used a radial basis function (RBF) network to approximate the state–action value function for the well-known mountain-car test domain. The temporal difference error of the value update is used to modify the weights, centers, and variances of the RBF units, although they noted that it was not particularly effective in producing good control policies.

Smart and Kaelbling (2000) used locally weighted learning algorithms and a set of heuristic rules to approximate the state–action value function. A set of states, sampled from those experienced by the learning agent, were stored along with their associated values. One approximation was stored for each discrete action. Interpolation between these exemplars was done by locally weighted averaging or locally weighted regression, supplemented with heuristics to avoid extrapolation and over-estimation. Learning was done on-line, with new instances being added as the learning agent explored the state space. The algorithm was shown to be effective in practice, but offered no theoretical guarantees.

Ormoneit and Sen (2002) presented an offline kernel-based reinforcement-learning algorithm that stores experiences (s_i, a_i, r_i, s'_i) as the instances, and uses these to approximate the state–action value function for problems with discrete actions. For a given state s and action a, the state–action value $Q(s,a)$ is approximated as

$$\hat{Q}(s,a) = \frac{1}{\eta_{s,a}} \sum_{i|a_i=a} \phi\left(\frac{d(s,s_i)}{\sigma}\right)\left[r_i + \gamma \max_{a'} \hat{Q}(s'_i, a')\right], \tag{4}$$

where ϕ is a kernel function, σ is the kernel bandwidth, γ is the RL discount factor, and $\eta_{s,a}$ is a normalizing term,

$$\eta_{s,a} = \sum_{i|a_i=a} \phi\left(\frac{d(s,s_i)}{\sigma}\right). \tag{5}$$

They showed that, with enough basis points, this approximation converges to the true value function, under some reasonable assumptions. However, they provide no bound on the number of basis points needed to provide a good approximation to the value function.

Assumptions

IBRL makes a number of assumptions about the form of the value function, and the underlying state space. The main assumptions are that state similarity is well measure by (weighted) Euclidean distance. This implicity assumes that the underlying state space be metric, and is a topological disk. Essentially, this means that stattes that are close to each other in the state space have similar value. This is clearly not true for states between which the agent cannot move, such as those on the opposite sides of a thin wall. In this case, there is a discontinuity in the state space, introduced by the wall, which is not well modeled by the instance-based algorithm.

Instance-based function approximation algorithms assume that the function they model is smooth and continuous between the basis points. Any discontinuities in the function tend to get "smoothed out" in the approximation. This assumption is especially problematic for value-function approximation, since it allows value on one side of the discontinuity to affect the approximation on the other. If the location of the discontinuity is known, and we are able to allocate an arbitrary number of basis points, we can overcome this problem. However, in practical applications of RL, neither of these is feasible, and the problem of approximating the value function at or near discontinuities remains an open one.

Problems and Drawbacks

Although IBRL has been shown to be effective on a number of problems, it does have a number of drawbacks that remain unaddressed. Instance-based approximation algorithms are often expensive in terms of storage, especially for long-lived agents. Although the literature contains many techniques for editing the basis set of instance-based approximators, these techniques are generally for a supervised learning setting, where the utility of a particular edit can be easily evaluated. In the RL setting, we lack the ground truth available to supervised learning, making the evaluation of edits considerably more difficult. Additionally, as the number of basis points increases, so does the time needed to perform an approximation. This limitation is significant in the RL setting, since many such value predictions are needed on every step of the accompanying RL algorithm.

The value of a particular state, s, is calculated by blending the values from other nearby states, s_i. This is problematic if it is not possible to move from state s to each of the states s_i. The value of s should only be influenced by the value of states reachable from s, but this condition is not enforced by standard instance-based approximation algorithms. This leads to problems when modeling discontinuities in the value function, as noted above, and in situations where the system dynamics constrain the agent's motion, as in the case of a "one-way door" in the state space.

IBRL also suffers badly from the curse of dimensionality; the number of points needed to adequately represent the value function is exponential in the dimensionality of the state space. However, by using only states actually experienced by the learning agent, we can lessen the impact of this problem. By using only observed states, we are explicitly modeling the manifold over which the system state moves. This manifold is embedded in the full state space and, for many real-world problems, has a lower dimensionality than the full space. The Euclidean distance metric used by many instance-based algorithms will not accurately measure distance along this manifold. In practice, the manifold over which the system state moves will be locally Euclidean for problems with smooth, continuous dynamics. As a result, the assumptions of

instance-based function approximators are valid locally and the approximations are of reasonable quality.

Cross References

►Curse of Dimensionality
►Instance-Based Learning
►Locally Weighted Learning
►Reinforcement Learning
►Value-Function Approximation

Recommended Reading

Gordon, G. J. (1995). Stable function approximation in dynamic programming. In *Proceedings of the twelfth international conference on machine learning* (pp. 261–268). Tahoe City, CA.

Kretchmar, R. M., & Anderson, C. W. (1997). Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. In *International conference on neural networks*, Houston, TX (Vol. 2, pp. 834–837).

Ormoneit, D., & Sen, Ś. (2002). Kernel-based reinforcement learning. *Machine Learning, 49*(2–3), 161–178.

Smart, W. D., & Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. In *Proceedings of the seventeenth international conference on machine learning (ICML 2000)* (pp. 903–910). Stanford, CA.

Szepesvári, C., & Munos, R. (2005). Finite time bounds for sampling based fitted value iteration. In *Proceedings of the twenty-second international conference on machine learning (ICML 2005)*, Bonn, Germany (pp. 880–887).

Szepesvári, C., & Smart, W. D. (2004). Interpolation-based Q-learning. In *Proceedings of the twenty-first international conference on machine learning (ICML 2004)*, Banff, Alberta, Canada (pp. 791–798).

Intelligent Backtracking

Synonyms

Dependency directed backtracking

Definition

Intelligent backtracking is a general class of techniques used to enhance search and constraint satisfaction algorithms. Backtracking is a general mechanism in search where a problem solver encounters an unsolvable search state and backtracks to a previous search state that might be solvable. Intelligent backtracking mechanisms provide various ways of selecting the backtracking point based on past experience in a way that is likely to be fruitful.

Intent Recognition

►Inverse Reinforcement Learning

Internal Model Control

Synonyms

Certainty equivalence principle; Model-based control

Definition

Many advanced controllers for nonlinear systems require knowledge of the model of the dynamics of the system to be controlled. The system dynamics is often called an "internal model," and the resulting controller is model-based. If the model is not known, it can be learned with function approximation techniques. The learned model is subsequently used as if it were correct in order to synthesize a controller – the control literature calls this assumption the "certainty equivalence principle."

Interval Scale

An **interval** measurement scale ranks the data, and the differences between units of measure can be calculated by arithmetic. However, *zero* in the interval level of measurement means neither "nil" nor "nothing" as *zero* in arithmetic means. See ►Measurement Scales.

Inverse Entailment

Definition

Inverse entailment is a ►generality relation in ►inductive logic programming. More specifically, when ►learning from entailment using a background theory B, a hypothesis H covers an example e, relative to the background theory B if and only if $B \wedge H \models e$, that is, the background theory B and the hypothesis H together entail the example (see ►entailment). For instance, consider the background theory B:

```
bird :- blackbird.
bird :- ostrich.
```

and the hypothesis H:

```
flies :- bird.
```

Together $B \wedge H$ entail the example e:

```
flies :- blackbird, normal.
```

This can be decided through deductive inference. Now when learning from entailment in inductive logic programming, one starts from the example e and the background theory B, and the aim is to induce a rule H that together with B entails the example. Inverting entailment is based on the observation that $B \wedge H \models e$ is logically equivalent to $B \wedge \neg e \models \neg H$, which in turn can be used to compute a hypothesis H that will cover the example relative to the background theory. Indeed, the negation of the example is $\neg e$:

```
blackbird.
normal.
:-flies.
```

and together with B this entails $\neg H$:

```
bird.
:-flies.
```

The principle of inverse entailment is typically employed to compute the ►bottom clause, which is the most specific clause covering the example under entailment. It can be computed by generating the set of all facts (true and false) that are entailed by $B \wedge \neg e$ and negating the resulting formula $\neg H$.

Cross References

►Bottom Clause
►Entailment
►Inductive Logic Programming
►Logic of Generality

Inverse Optimal Control

►Inverse Reinforcement Learning

Inverse Reinforcement Learning

Pieter Abbeel[1], Andrew Y. Ng[2]
[1]University of California, Berkeley, California, USA
[2]Stanford University, Stanford, California, USA

Synonyms

Intent recognition; Inverse optimal control; Plan recognition

Definition

Inverse reinforcement learning (inverse RL) considers the problem of extracting a reward function from observed (nearly) optimal behavior of an expert acting in an environment.

Motivation and Background

The motivation for inverse RL is two fold:

1. For many RL applications, it is difficult to write down an explicit reward function specifying how different desiderata should be traded off exactly. In fact, engineers often spend significant effort tweaking the reward function such that the optimal policy corresponds to performing the task they have in mind. For example, consider the task of driving a car well. Various desiderata have to be traded off, such as speed, following distance, lane preference, frequency of lane changes, distance from the curb, and so on. Specifying the reward function for the task of driving requires explicitly writing down the trade-off between these features.
   Inverse RL algorithms provide an efficient solution to this problem in the apprenticeship learning setting – when an expert is available to demonstrate the task. Inverse RL algorithms exploit the fact that an expert demonstration implicitly encodes the reward function of the task at hand.
2. Reinforcement learning and related frameworks are often used as computational models for animal and human learning (Schmajuk & Zanutto, 1997; Touretzky & Saksida, 1997; Watkins, 1989). Such models are supported both by behavioral studies and by neurophysiological evidence that reinforcement learning occurs in bee foraging (Montague, Dayan, Person, & Sejnowski, 1995) and

in songbird vocalization (Doya & Sejnowski, 1995). It seems clear that in examining animal and human behavior, we must consider the reward function as an unknown to be ascertained through empirical investigation, particularly when dealing with multiattribute reward functions. Consider, for example, that the bee might weigh nectar ingestion against flight distance, time, and risk from wind and predators. It is hard to see how one could determine the relative weights of these terms a priori. Similar considerations apply to human economic behavior, for example. Hence, inverse reinforcement learning is a fundamental problem of theoretical biology, econometrics, and other scientific disciplines that deal with reward-driven behavior.

Structure of the Learning System

Preliminaries and Notation

A Markov decision process (MDP) is a tuple $\langle S, A, T, \gamma, D, R\rangle$, where S is a finite set of states; A is a set of actions; $T = \{P_{sa}\}$ is a set of state-transition probabilities (here, P_{sa} is the state transition distribution upon taking action a in state s); $\gamma \in [0,1)$ is a discount factor; D is the distribution over states for time zero; and $R : S \mapsto \mathbb{R}$ is the reward function.

A policy π is a mapping from states to probability distributions over actions. Let Π denotes the set of all stationary policies. (We restrict attention to stationary policies, since it is well known that there exists a stationary policy that is optimal for infinite horizon MDPs.) The utility of a policy π is given by

$$U(\pi) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi\right].$$

The expectation is taken with respect to the random state sequence $s_0, s_1, s_2, \ldots$ drawn by starting from a state $s_0 \sim D$, and picking actions according to π.

Let $\mu_S(\pi)$ be the discounted distribution over states when acting according to the policy π. In particular, for a discrete state space we have that $[\mu_S(\pi)](s) = \sum_{t=0}^{\infty} \gamma^t \mathrm{Prob}(s_t = s|\pi)$. (In the case of a continuous state space, we replace $\mathrm{Prob}(s_t = s|\pi)$ by the appropriate probability density function.) Then, we have that

$$U(\pi) = R^\top \mu_S(\pi).$$

Thus, the utility of a policy π is linear in the reward function.

Often the reward function R can be represented more compactly. Let $\phi : S \to \mathbb{R}^n$ be a feature map. A typical assumption in inverse RL is to assume the reward function R is a linear combination of the features ϕ: $R(s) = w^\top \phi(s)$. Then, we have that the utility of a policy π is linear in the reward function weights w:

$$\begin{aligned} U(\pi) &= \mathrm{E}\left[\textstyle\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi\right] \\ &= \mathrm{E}\left[\textstyle\sum_{t=0}^{\infty} \gamma^t w^\top \phi(s_t)|\pi\right] \\ &= w^\top \mathrm{E}\left[\textstyle\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi\right] \\ &= w^\top \mu_\phi(\pi). \end{aligned} \tag{1}$$

Here, we used linearity of expectation to bring w outside of the expectation. The last equality defines the vector of *feature expectations* $\mu_\phi(\pi) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi\right]$.

We assume access to demonstrations by some expert. We denote the expert's policy by π^*. Specifically, we assume the ability to observe trajectories (state sequences) generated by the expert starting from $s_0 \sim D$ and taking actions according to π^*.

Characterization of the Inverse RL Solution Set

A reward function R is consistent with the policy π^* being optimal if and only if the utility obtained when acting according to the policy π^* is at least as high as the utility obtained when acting according to any other policy π, or equivalently,

$$U(\pi^*) \geq U(\pi) \quad \forall \pi \in \Pi. \tag{2}$$

Using the fact that $U(\pi) = R^\top \mu_S(\pi)$, we can equivalently write the conditions of Eq. (2) as a set of linear constraints on the reward function R:

$$R^\top \mu_S(\pi^*) \geq R^\top \mu_S(\pi) \quad \forall \pi \in \Pi. \tag{3}$$

The state distribution $\mu_S(\pi)$ does not depend on the reward function R. Thus, Eq. (3) is a set of linear constraints in the reward function and we can use a linear program (LP) solver to find a reward function consistent with the policy π^* being optimal. Strictly speaking, Eq. (3) solves the inverse RL problem. However, to apply inverse RL in practice, the following three issues need to be addressed:

1. **Reward Function Ambiguity.** Typically, a large set of reward functions satisfy all the constraints of Eq. (3). One such reward function that satisfies all the constraints for any MDP is the all-zeros reward function (it is consistent with any policy being optimal). Clearly, the all-zeros reward function is not a desirable answer to the inverse RL problem. More generally, this observation suggests not all reward functions satisfying Eq. (3) are of equal interest and raises the question of how to recover reward functions that are of interest to the inverse RL problem.
2. **Statistical Efficiency.** Often the state space is very large (or even infinite) and we do not have sufficiently many expert demonstrations available to accurately estimate $\mu(\cdot;\pi^*)$ from data.
3. **Computational Efficiency.** The number of constraints in Eq. (3) is equal to the number of stationary policies $|\Pi|$ and grows quickly with the number of states and actions of the MDP. For finite-state-action MDPs, we have $|A|^{|S|}$ constraints. So, even for small state and action spaces, feeding all the constraints of Eq. (3) into an LP solver becomes quickly impractical. For continuous state-action spaces, the formulation of Eq. (3) has an infinite number of constraints, and thus using a standard LP solver to find a feasible reward function R is impossible.

In the following sections, we address these three issues.

Reward Function Ambiguity As observed above, typically a large set of reward functions satisfy all the constraints of Eq. (3). To obtain a single reward function, it is natural to reformulate the inverse RL problem as an optimization problem. We describe one standard approach for disambiguation. Of course, many other formulations as an optimization problem are possible.

Similar to common practice in support vector machines research, one can maximize the (soft) margin by which the policy π^* outperforms all other policies. As is common in structured prediction tasks (see, e.g., Taskar, Guestrin, & Koller, 2003), one can require the margin by which the policy π^* outperforms another policy π to be larger when π differs more from π^*, as measured according to some function $h(\pi^*,\pi)$. The resulting formulation (Ratliff, Bagnell, & Zinkevich, 2006) is

$$\begin{aligned} &\min_{R,\xi} \quad \|R\|_2^2 + C\xi \\ &\text{s.t. } R^\top \mu_S(\pi^*) \geq R^\top \mu_S(\pi) + h(\pi^*,\pi) - \xi \quad \forall \pi \in \Pi. \end{aligned} \tag{4}$$

For the resulting optimal reward function to correspond to a desirable solution to the inverse RL problem, it is important that the objective and the margin scaling encode the proper prior knowledge. If a sparse reward function is suggested by prior knowledge, then a 1-norm might be more appropriate in the objective. An example of a margin scaling function for a discrete MDP is the number of states in which the action prescribed by the policy π differs from the action prescribed by the expert policy π^*. If the expert has only been observed in a small number of states, then one could restrict attention to these states when evaluating this margin scaling function.

Another way of encoding prior knowledge is by restricting the reward function to belong to a certain functional class, for example, the set of functions linear in a specified set of features. This approach is very common, and is also important for statistical efficiency. It will be explained in the next section.

Remark. When using inverse RL to help us specify a reward function for a given task based on an expert demonstration, it is not necessary to explicitly resolve the ambiguities in the reward function. In particular, one can provably perform as well as the expert without matching the expert's reward function. More details are given in Sect. 4.3.

Statistical Efficiency As formulated thus far, solving the inverse RL problem requires the knowledge (or accurate statistical estimates) of $\mu_S(\pi^*)$. For most practical problems, the number of states is large (or even infinite) and thus accurately estimating $\mu_S(\pi^*)$ requires a very large number of expert demonstrations. This (statistical) problem can be resolved by restricting the reward function to belong to a prespecified class of functions. The common approach is to assume the reward function R can be expressed as a linear combination of a known set of features. In particular, we have $R(s) = w^\top \phi(s)$. Using this assumption, we can use the expression for the utility of the policy π from Eq. (1).

Rewriting Eq. (4), we now have the following constraints in the reward weights w:

$$\min_{w,\xi} \quad \|w\|_2^2 + C\xi$$
$$\text{s.t. } w^\top \mu_\phi(\pi^*) \geq w^\top \mu_\phi(\pi) + h(\pi^*, \pi) - \xi \quad \forall \pi \in \Pi. \tag{5}$$

This new formulation requires only estimates of the expected feature counts $\mu_\phi(\pi^*)$, rather than estimates of the distribution over the state space $\mu_S(\pi^*)$. Assuming the number of features is smaller than the number of states, this significantly reduces the number of expert demonstrations required.

Computational Efficiency For concreteness, we will consider the formulation of Eq. (5). Although the number of variables is only equal to the number of features in the reward function, the number of constraints is very large (equal to the number of stationary policies). As a consequence, feeding the problem into a standard quadratic programming (QP) solver will not work.

Ratliff et al. (2006) suggested a formal computational approach to solving the inverse RL problem, using standard techniques from convex optimization, which provide convergence guarantees. More specifically, they used a subgradient method to optimize the following equivalent problem:

$$\min_{w,\xi} \|w\|_2^2 + C \max_{\pi \in \Pi} \left(w^\top \mu_\phi(\pi) + h(\pi^*, \pi) - w^\top \mu_\phi(\pi^*) \right). \tag{6}$$

In each iteration, to compute the subgradient, it is sufficient to find the optimal policy with respect to a reward function that is easily determined from the current reward weights w and the margin scaling function $h(\pi^*, \cdot)$. In more recent work, Ratliff, Bradley, Bagnell, and Chestnutt (2007) proposed a boosting algorithm to solve a formulation similar to Eq. (6), which also includes feature selection.

A Generative Approach to Inverse RL

Abbeel and Ng (2004) made the following observation, which resolves the ambiguity problem in a completely different way: if, for a policy π, we have that $\mu_\phi(\pi) = \mu_\phi(\pi^*)$, then the following holds:

$$U(\pi) = w^\top \mu_\phi(\pi) = w^\top \mu_\phi(\pi^*) = U(\pi^*),$$

no matter what the value of w is. Thus, to perform as well as the expert, it is sufficient to find a policy that attains the same expected feature counts μ_ϕ as the expert.

Abbeel and Ng provide an algorithm that finds a policy π satisfying $\mu_\phi(\pi) = \mu_\phi(\pi^*)$. The algorithm iterates over two steps: (1) generate a reward function by solving a QP; (2) solve the MDP for the current reward function.

In contrast to the previously described inverse RL methods, which focus on merely recovering a reward function that could explain the expert's behavior, this inverse RL algorithm is shown to find a policy that performs at least as well as the expert. The algorithm is shown to converge in a polynomial number of iterations.

Apprenticeship Learning: Inverse RL Versus Imitation Learning

Inverse RL alleviates the need to specify a reward function for a given task when expert demonstrations are available. Alternatively, one could directly estimate the policy of the expert using a standard machine-learning algorithm, since it is simply a mapping from state to action. The latter approach, often referred to as ▶imitation learning or ▶behavioral cloning, has been successfully tested on a variety of tasks, including learning to fly in a fixed-wing flight simulator (Sammut, Hurst, Kedzier, & Michie, 1992), and learning to drive a car (Pomerleau, 1989).

The behavioral cloning approach can be expected to be successful whenever the policy class to be considered can be learned efficiently from data. In contrast, the inverse RL approach relies on having a reward function that can be estimated efficiently from data.

Cross References

▶Apprenticeship Learning
▶Reinforcement Learning
▶Reward Shaping

Recommended Reading

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of ICML*, Banff, Alberta, Canada.

Doya, K., & Sejnowski, T. (1995). A novel reinforcement model of birdsong vocalization learning. In *Neural Information Processing Systems 7*. Cambridge, MA: MIT Press.

Montague, P. R., Dayan, P., Person, C., & Sejnowski, T. J. (1995). Bee foraging in uncertain environments using predictive hebbian learning. *Nature, 377*(6551), 725–728.
Pomerleau, D. (1989). ALVINN: An autonomous land vehicle in a neural network. In *NIPS 1*. San Francisco, CA: Morgan Kaufmann.
Ratliff, N., Bagnell, J., & Zinkevich, M. (2006). Maximum margin planning. In *Proceedings of ICML*, Pittsburgh, Pennsylvania.
Ratliff, N., Bradley, D., Bagnell, J., & Chestnutt, J. (2007). Boosting structured prediction for imitation learning. In *Neural Information Processing Systems 19*. Cambridge, MA: MIT Press.
Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. In *Proceedings of ICML*. Aberdeen, Scotland, UK.
Schmajuk, N. A., & Zanutto, B. S. (1997). Escape, avoidance, and imitation. *Adaptive Behavior, 6*, 63–129.
Taskar, B., Guestrin, C., & Koller, D. (2003). Max-margin Markov networks. In *Neural Information Processing Systems Conference (NIPS03)*, Vancouver, Canada.
Touretzky, D. S., & Saksida, L. M. (1997). Operant conditioning in skinnerbots. *Adaptive Behavior, 5*, 219–247.
Watkins, C. J. (1989). *Models of delayed reinforcement learning*. PhD thesis, Psychology Department, Cambridge University.

Inverse Resolution

Definition

Inverse resolution is, as the name indicates, a rule that inverts resolution. This follows the idea of induction as the inverse of deduction formulated in the ▶logic of generality. The resolution rule is the best-known deductive inference rule, used in many theorem provers and logic programming systems. ▶Resolution starts from two ▶clauses and derives the resolvent, a clause that is entailed by the two clauses. This can be graphically represented using the following schema (for propositional logic).

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n}.$$

Inverse resolution operators, such as *absorption* (17) and *identification* (17), invert this process. To this aim, they typically assume the resolvent is given together with *one* of the original clauses and then derive the missing clause. This leads to the following two operators, which start from the clauses below and induce the clause above the line.

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m},$$

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n \text{ and } h \leftarrow g, a_1, \ldots, a_n}.$$

The operators are shown here only for the propositional case, as the first order case is more involved as it requires one to deal with substitions as well as inverse substitutions.

As one example, consider the clauses

```
(1) flies :- bird, normal.
(2) bird :- blackbird.
(3) flies :- blackbird, normal.
```

Here, (3) is the resolvent of (1) and (2). Furthermore, starting from (3) and (2), the absorption operator would generate (1), and starting from (3) and (1), the identification operator would generate (2).

Cross References

▶First-Order Logic
▶Logic of Generality
▶Resolution

Is More General Than

▶Logic of Generality

Is More Specific Than

▶Logic of Generality

Item

▶Instance

Iterative Classification

▶Collective Classification

J

Junk Email Filtering

►Text Mining for Spam Filtering

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

K

k-Armed Bandit

Shie Mannor
Israel Institute of Technology, Haifa, Israel

Synonyms

Multi-armed bandit; Multi-armed bandit problem

Definition

In the classical *k*-armed bandit problem, there are *k* alternative arms, each with a stochastic reward whose probability distribution is initially unknown. A decision maker can try these arms in some order, which may depend on the rewards that have been observed so far. A common objective in this context is to find a policy for choosing the next arm to be tried, under which the sum of the expected rewards comes as close as possible to the ideal reward, that is, the expected reward that would be obtained if it were to try the "best" arm at all times. There are many variants of the *k*-armed bandit problem that are distinguished by the objective of the decision maker, the process governing the reward of each arm, and the information available to the decision maker at the end of every trial.

Motivation and Background

k-Armed bandit problems are a family of sequential decision problems that are among the most studied problems in statistics, control, decision theory, and machine learning. In spite of their simplicity, they encompass many of the basic problems of sequential decision making in uncertain environments such as the tradeoff between exploration and exploitation.

There are many variants of bandit problems including Bayesian, Markovian, adversarial, budgeted, and exploratory variants. Bandit formulations arise naturally in multiple fields and disciplines including communication networks, clinical trials, search theory, scheduling, supply chain automation, finance, control, information technology, etc. (Berry & Fristedt, 1985; Cesa-Bianchi & Lugosi, 2006; Gittins, 1989).

The term "multi-armed bandit" is borrowed from the slang term for a slot machine (the one-armed bandit), where a decision maker has to decide whether to insert a coin into the gambling machine and pull a lever possibly getting a significant reward, or to quit without spending any money.

Theory

We briefly review some of the most popular bandit variants.

The Stochastic *k*-Armed Bandit Problem

The classical stochastic bandit problem is described as follows. There are *k* arms (or machines or actions) and a single decision maker (or controller or agent). Each arm corresponds to a discrete time Markov process. At each timestep, the decision maker observes the current state of each arm's process and selects one of the arms. As a result, the decision maker obtains a reward from the process of the selected arm and the state of the corresponding process changes. Arms that are not selected are "frozen" and their processes remain in the same state. The objective of the decision maker is to maximize her (discounted) reward.

More formally, let the state of arm n's process at stage t be $x_n(t)$. Then, if the decision maker selects arm $m(t)$ at time t we have that:

$$x_n(t+1) = \begin{cases} x_n(t) & n \neq m(t) \\ f_n\left(x_n(t), \omega\right) & n = m(t) \end{cases},$$

where $f_n(x, \omega)$ is a function that describes the (possibly stochastic) transition probability of the n-th process and accepts the state of the n-th process and a random disturbance ω.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

The reward the decision maker receives at time t is a function of the current state and a random element: $r(x_{m(t)}(t), \omega)$. The objective of the decision maker is to maximize her cumulative discounted reward. That is, she wishes to maximize

$$V = \mathbf{E}^{\pi}\left[\sum_{t=1}^{\infty} \gamma^t r\left(x_{m(t)}(t), \omega_t\right)\right],$$

where $\mathbf{E}^{\pi}$ is the expectation obtained when following policy π and γ is a discount factor $(0 < \gamma < 1)$. A policy is a decision rule for selected arms as a function of the state of the processes.

This problem can be solved using ▶dynamic programming, but the state space of the joint Markov decision process is exponential in the number of arms. Moreover, the dynamic programming solution does not reveal the important structural properties of the solution.

Gittins and Jones (1972) showed that there exists an optimal index policy. That is, there is a function that maps the state of each arm to real number (the "index") such that the optimal policy is to choose the arm with the highest index at any given time. Therefore, the stochastic bandit problem reduces to the problem of computing the index, which can be easily done in many important cases.

Regret Minimization for the Stochastic *k*-Armed Bandit Problem

A different flavor of the bandit problem focuses on the notion of regret, or learning loss. In this formulation, there are k arms as before and when selecting arm m a reward that is independent and identically distributed is given (the reward depends only on the identity of the arm and not on some internal state or the results of previous trials). The decision maker's objective is to obtain high expected reward. Of course, if the decision maker had known the statistical properties of each arm, she would have always chosen the arm with the highest expected reward. However, the decision maker does not know the statistical properties of the arms in advance, in this setting.

More formally, if the reward when choosing arm m has expectation r_m, the regret is defined as:

$$r(t) = t \cdot \max_{1 \le m \le k} r_m - \mathbf{E}^{\pi}\left[\sum_{\tau=1}^{t} r(\tau)\right],$$

where $r(t)$ is sampled from the arm $m(t)$. This quantity represents the expected loss for not choosing the arm with the highest expected reward on every timestep.

This variant of the bandit problem highlights the tension between acquiring information (exploration) and using the available information (exploitation). The decision maker should carefully balance between the two since if she chooses to only try the arm with the highest estimated reward she might regret not exploring other arms whose reward is underestimated but is actually higher than the reward of the arm with highest estimated reward.

A basic question in this context is whether $R(t)$ can be made to grow sub-linearly. Robbins (1952) answered this question in the affirmative. It was later proved (Lai & Robbins, 1985) that it is possible in fact to obtain logarithmic regret (the growth of the regret is logarithmic in the number of timesteps). Matching lower bounds (and constants) were also derived.

The Non-stochastic *k*-Armed Bandit Problem

A third popular variant of the bandit problem is the non-stochastic one. In this problem, it is assumed that the sequence of rewards each arm produces is deterministic (possibly adversarial). The decision maker, as in the stochastic bandit problem, wants to minimize her regret, where the regret is measured with respect to the best fixed arm (this best arm might change with time, however). Letting the reward of arm m at time t be $r_m(t)$, we redefine the regret as:

$$r(t) = \max_{1 \le m \le k} \sum_{\tau=1}^{t} r_m(\tau) - \mathbf{E}^{\pi}\left[\sum_{\tau=1}^{t} r(\tau)\right],$$

where the expectation is now taken with respect to randomness in the arm selection. The basic question here is if the regret can be made to grow sub-linearly. The case where the reward of each arm is observed was addressed in the 1950s (see Cesa-Bianchi & Lugosi, 2006, for a discussion), where it was shown that there are algorithms that guarantee that the regret grows like $\sqrt{t}$. For the more difficult case, where only the reward of the selected arm is observed and that the rewards of the other arms may not be observed it was shown (Auer, Cesa-Bianchi, Freund, & Schapire, 2002) that the same conclusion still holds.

It should be noticed that the optimal policy of the decision maker in this adversarial setting is generally randomized. That is, the decision maker has to select an action at random by following some distribution. The reason is that if the action the decision maker takes is deterministic and can be predicted by Nature, then Nature can consistently "give" the decision maker a low reward for the selected arm while "giving" a high reward to all other arms, leading to a linear regret.

There are some interesting relationships between the non-stochastic bandit problem and prediction with expert advice, universal prediction, and learning in games (Cesa-Bianchi & Lugosi, 2006).

The Exploratory *k*-Armed Bandit Problem

This bandit variant emphasizes efficient exploration rather than on the exploration–exploitation tradeoff. As in the stochastic bandit problem, the decision maker is given access to k arms where each arm is associated with an independent and identically distributed random variable with unknown statistics. The decision maker's goal is to identify the "best" arm. That is, the decision maker wishes to find the arm with the highest expected reward as quickly as possible.

The exploratory bandit problem is a sequential hypothesis testing problem but with the added complication that the decision maker can choose where to sample next, making it among the simplest active learning problems. In the context of the probably approximate correct (PAC) setup, it was shown (Mannor & Tsitsiklis, 2004) that finding the ε-optimal arm (that is, an arm whose expected reward is lower than that of the best arm by at most ε) with probability of at least $1 - \delta$ requires

$$O\left(\frac{k}{\varepsilon^2}\log\left(\frac{1}{\delta}\right)\right)$$

samples on expectation. Moreover, this bound can be obtained (up to multiplicative constants) via an algorithm known as median elimination.

Bandit analyses such as these have played a key role in understanding the efficiency of ►reinforcement-learning algorithm as well.

Cross References

►Active Learning
►Associative Bandit Problems
►Dynamic Programming
►Machine Learning in Games
►Markov Processes
►PAC Learning
►Reinforcement Learning

Recommended Reading

Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. E. (2002). The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing, 32*(1), 48–77.

Berry, D., & Fristedt, B. (1985). *Bandit problems: Sequential allocation of experiments*. London/New York: Chapman and Hall.

Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games*. New York: Cambridge University Press.

Gittins, J. C. (1989). *Multi-armed bandit allocation indices*. New York: Wiley.

Gittins, J., & Jones, D. (1972). A dynamic allocation index for sequential design of experiments. In *Progress in statistics, European Meeting of Statisticians* (Vol. 1, pp. 241–266).

Lai, T. L., & Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics, 6*, 4–22.

Mannor, S., & Tsitsiklis, J. N. (2004). The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research, 5*, 623–648.

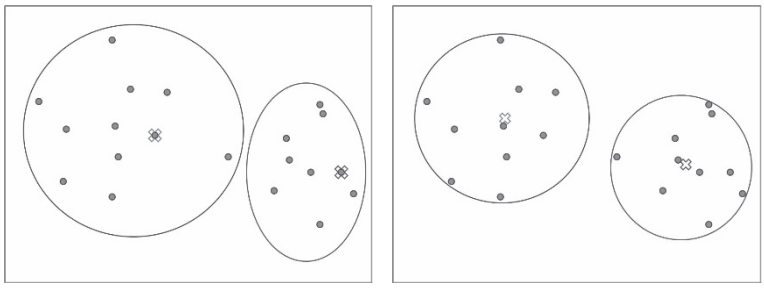
Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society, 55*, 527–535.

K-Means Clustering

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

K-means (Lloyd, 1957; MacQueen, 1967) is one of the most popular clustering methods. Algorithm 1 shows the procedure of *K*-means clustering. The basic idea is: Given an initial but not optimal clustering, relocate each point to its new nearest center, update the clustering centers by calculating the mean of the member points, and repeat the relocating-and-updating process until convergence criteria (such as predefined number of iterations, difference on the value of the distortion function) are satisfied.

The task of initialization is to form the initial K clusters. Many initializing techniques have been proposed, from simple methods, such as choosing the first K data points, Forgy initialization (randomly choosing K data points in the dataset) and Random partitions



***K*-Means Clustering. Figure 1.** ***K*-Means clustering example ($K = 2$). The center of each cluster is marked by "*x*"**

(dividing the data points randomly into K subsets), to more sophisticated methods, such as density-based initialization, Intelligent initialization, Furthest First initialization (FF for short, it works by picking the first center point randomly, then adding more center points which are furthest from existing ones), and subset furthest-first (SFF)initialization. For more details, refer to paper Steinley and Brusco (2007) which provides a survey and comparison of over 12 initialization methods.

Figure 1 shows an example of K-means clustering on a set of points, with $K = 2$. The clusters are initialized by randomly selecting two points as centers.

Complexity analysis. Let N be the number of points, D the number of dimensions, and K the number of centers. Suppose the algorithm runs I iterations to converge. The space complexity of K-means clustering algorithm is $O(N(D+K))$. Based on the number of distance calculations, the time complexity of K-means is $O(NKI)$.

Algorithm 1 K-means clustering algorithm

Require: K, number of clusters; D, a data set of N points
Ensure: A set of K clusters

1. Initialization.
2. **repeat**
3. **for** each point p in D **do**
4. find the nearest center and assign p to the corresponding cluster.
5. **end for**
6. update clusters by calculating new centers using mean of the members.
7. **until** stop-iteration criteria satisfied
8. **return** clustering result.

Recommended Reading

Lloyd, S. P. (1957). Least squares quantization in PCM. Technical Report RR-5497, Bell Lab, September 1957.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In L. M. Le Cam & J. Neyman (Eds.), *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281–297). California: University of California Press.

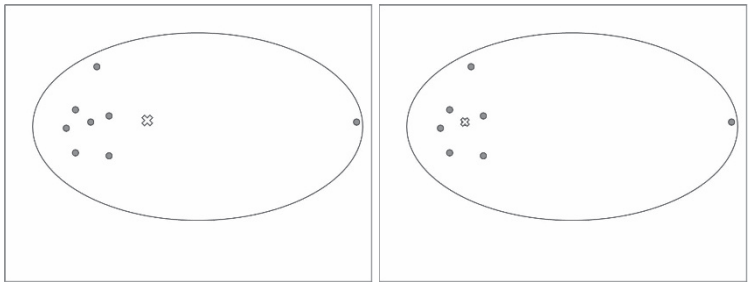
Steinley, D., & Brusco, M. J. (2007). Initializing k-means batch clustering: A critical evaluation of several techniques. *Journal of Classification, 24*(1), 99–121.

K-Medoids Clustering

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

The K-means clustering algorithm is sensitive to outliers, because a mean is easily influenced by extreme values. K-medoids clustering is a variant of K-means that is more robust to noises and outliers. Instead of using the mean point as the center of a cluster, K-medoids uses an actual point in the cluster to represent it. Medoid is the most centrally located object of the cluster, with minimum sum of distances to other points. Figure 1 shows the difference between mean and medoid in a 2-D example. The group of points in the right form a cluster, while the rightmost point is an outlier. Mean is greatly influenced by the outlier and thus cannot represent the correct cluster center, while medoid is robust to the outlier and correctly represents the cluster center.

Partitioning around medoids (PAM) (Kaufman & Rousseeuw, 2005) is a representative K-medoids clustering method. The basic idea is as follows: Select K representative points to form initial clusters, and then



(a) Mean (b) Medoid

***K*-Medoids Clustering. Figure 1. Mean vs. medoid in 2-D space. In both figures (a) and (b), the group of points in the right form a cluster and the rightmost point is an outlier. The red point represents the center found by mean or medoid**

repeatedly moves to better cluster representatives. All possible combinations of representative and nonrepresentative points are analyzed, and the quality of the resulting clustering is calculated for each pair. An original representative point is replaced with the new point which causes the greatest reduction in distortion function. At each iteration, the set of best points for each cluster form the new respective medoids.

The time complexity of the PAM algorithm is $O(K(N-K)^2 I)$. PAM is not scalable for large dataset, and some algorithms have been proposed to improve the efficiency, such as Clustering LARge Applications (CLARA) (Kaufman & Rousseeuw, 2005) and Clustering Large Applications based upon RANdomized Search (CLARANS) (Ng & Han, 2002).

Recommended Reading

Kaufman, L., & Rousseeuw, P. J. (2005). *Finding groups in data: An introduction to cluster analysis (Wiley series in probability and statistics)*. New York: Wiley-Interscience.

Ng, R. T., & Han, J. (2002). Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering, 14*(5), 1003–1016.

K-Way Spectral Clustering

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

In spectral clustering (Luxburg, 2007), the dataset is represented as a similarity graph $G=(V,E)$. The vertices represent the data points. Two vertices are connected if the similarity between the corresponding data points is larger than a certain threshold, and the edge is weighted by the similarity value. Clustering is achieved by choosing a suitable partition of the graph that each group corresponds to one cluster.

A good partition (i.e., a good clustering) is that the edges between different groups have overall low weights and the edges within a group have high weights, which indicates that the points in different clusters are dissimilar from each other and the points within the same cluster are similar to each other. One basic spectral clustering algorithm finds a good partition in the following way:

Given a set of data points P and the similarity matrix S, where S_{ij} measures the similarity between points $i,j \in P$, form a graph. Build a Laplacian matrix L of the graph,

$$L = I - D^{-1/2} S D^{-1/2}, \tag{1}$$

where D is the diagonal matrix

$$D_{ii} = \sum_j S_{ij}. \tag{2}$$

Find the eigenvalues and eigenvectors of the matrix L, map the vertices to corresponding components and form clusters based on the embedding space.

The methods to find K clusters include recursive bipartitioning and clustering multiple eigenvectors. The former technique is inefficient and unstable. The latter approach is more preferable because it is able to prevent instability due to information loss.

Recommended Reading

Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing, 17*(4), 395–416.

Kernel Density Estimation

►Density Estimation

Kernel Matrix

Synonyms

Gram matrix

Definition

Given a kernel function $k : X \times X \to$ and patterns $x_1, \ldots, x_m \in X$, the $m \times m$ matrix K with elements $K_{ij} := k(x_i, x_j)$ is called the kernel matrix of k with respect to $x_1, \ldots, x_m$.

Kernel Methods

Xinhua Zhang
Australian National University, Canberra, Australia
NICTA London Circuit, Canberra, Australia

Definition

Kernel methods refer to a class of techniques that employ positive definite kernels. At an algorithmic level, its basic idea is quite intuitive: implicitly map objects to high-dimensional feature spaces, and then directly specify the inner product there. As a more principled interpretation, it formulates learning and estimation problems in a reproducing kernel Hilbert space, which is advantageous in a number of ways:

- It induces a rich feature space and admits a large class of (nonlinear) functions.
- It can be flexibly applied to a wide range of domains including both Euclidean and non-Euclidean spaces.
- Searching in this infinite-dimensional space of functions can be performed efficiently, and one only needs to consider the finite subspace expanded by the data.
- Working in the linear spaces of function lends significant convenience to the construction and analysis of learning algorithms.

Motivation and Background

Over the past decade, kernel methods have gained much popularity in machine learning. Linear estimators have been popular due to their convenience in analysis and computation. However, nonlinear dependencies exist intrinsically in many real applications, and are indispensable for effective modeling. Kernel methods can sometimes offer the best of both aspects. The reproducing kernel Hilbert space provides a convenient way to model nonlinearity, while the estimation is kept linear. Kernels also offer significant flexibility in analyzing generic non-Euclidean objects such as graphs, sets, and dynamic systems. Moreover, kernels induce a rich function space where functional optimization can be performed efficiently. Furthermore, kernels have also been used to define statistical models via exponential families or Gaussian processes, and can be factorized by graphical models. Indeed, kernel methods have been widely used in almost all tasks in machine learning.

The reproducing kernel was first studied by Aronszajn (1950). Poggio and Girosi (1990) and Wahba (1990) used kernels for data analysis and Boser, Guyon, and Vapnik (1992) incorporated kernel function into the maximum margin models. Schölkopf, Smola, and Müller (1998) first used kernels for principal component analysis.

Theory

Positive semi-definite kernels are the most commonly used type of kernels, and its motivation is as follows. Given two objects x_1, x_2 from a space $\mathcal{X}$, which is not necessarily Euclidean, we map them to a high-dimensional feature space via $\phi(x_1)$ and $\phi(x_2)$, and then compute the inner products there by $k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$. In many algorithms, the set $\{x_i\}$ influences learning only via inner products between x_i and x_j, hence it is sufficient to specify $k(x_1, x_2)$ directly without explicitly defining ϕ. This leads to considerable savings in computation, when ϕ ranges in high-dimensional spaces or even infinite-dimensional spaces. Clearly, the function k must satisfy some conditions. For example, as a necessary condition, for any finite number of examples $x_1, \ldots, x_n$ from $\mathcal{X}$, the matrix

$$K := (k(x_i, x_j))_{i,j} = (\phi(x_1), \ldots, \phi(x_n))^\top (\phi(x_1), \ldots, \phi(x_n))$$

must be positive semi-definite. Surprisingly, this turns out to be a sufficient condition as well, and hence we define the positive semi-definite kernels.

Definition 1 (Positive semi-definite kernels) *Let $\mathcal{X}$ be a nonempty set. A function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called a positive semi-definite kernel if for any $n \in \mathbb{N}$ and $x_1, \ldots, x_n \in \mathcal{X}$, the Gram matrix $K := (k(x_i, x_j))_{i,j}$ is symmetric and positive semi-definite (psd).*

Reproducing Kernel Hilbert Space

Given a psd kernel k, we are able to construct a map ϕ from $\mathcal{X}$ to an inner product space $\mathcal{H}$, such that $\langle \phi(x_1), \phi(x_2) \rangle = k(x_1, x_2)$. The image of x under ϕ is just a function $\phi(x) := k(x, \cdot)$, where $k(x, \cdot)$ is a function of $\cdot$, assigning the value $k(x, x')$ for any $x' \in \mathcal{X}$. To define inner products between functions, we need to construct an inner product space $\mathcal{H}$ that contains $\{k(x, \cdot) : x \in \mathcal{X}\}$. First, $\mathcal{H}$ must contain the linear combinations $\{\sum_{i=1}^{n} \alpha_i k(x_i, \cdot) : n \in \mathbb{N}, x_i \in \mathcal{X}, \alpha_i \in \mathbb{R}\}$. Then, we endow it with an inner product as follows. For any $f, g \in \mathcal{H}$ and $f = \sum_{i=1}^{n} \alpha_i k(x_i, \cdot), g = \sum_{j=1}^{m} \beta_j k\left(x'_j, \cdot\right)$, define

$$\langle f, g \rangle := \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k\left(x_i, x'_j\right),$$

and it is easy to show that this is well defined (independent of the expansion of f and g). Using the induced norm, we can complete the space and thus get a Hilbert space $\mathcal{H}$, which is called reproducing kernel Hilbert space (RKHS). The term "reproducing" is because for any function $f \in \mathcal{H}$, $\langle f, k(x, \cdot) \rangle = f(x)$.

Properties of psd Kernels

Let $\mathcal{X}$ be a nonempty set and $k_1, k_2, \ldots$ be arbitrary psd kernels on $\mathcal{X} \times \mathcal{X}$. Then

- The set of psd kernels is a closed convex cone, that is, (a) if $\alpha_1, \alpha_2 \geq 0$, then $\alpha_1 k_1 + \alpha_2 k_2$ is psd; (b) if $k(x, x') := \lim_{n \to \infty} k_n(x, x')$ exists for all x, x', then k is psd.
- The pointwise product $k_1 k_2$ is psd.
- Assume for $i = 1, 2$, k_i is a psd kernel on $\mathcal{X}_i \times \mathcal{X}_i$, where $\mathcal{X}_i$ is a nonempty set. Then the tensor product $k_1 \otimes k_2$ and the direct sum $k_1 \oplus k_2$ are psd kernels on $(\mathcal{X}_1 \times \mathcal{X}_2) \times (\mathcal{X}_1 \times \mathcal{X}_2)$.

Example Kernels

One of the key advantage of kernels lies in its applicability to a wide range of objects.

Euclidean spaces: In $\mathbb{R}^n$, popular kernels include linear kernel $k(x_1, x_2) = \langle x_1, x_2 \rangle$, polynomial kernels $k(x_1, x_2) = (\langle x_1, x_2 \rangle + c)^d$ where $d \in \mathbb{N}$ and $c \geq 0$, Gaussian RBF kernels $k(x_1, x_2) = \exp(-\gamma \|x_1 - x_2\|^2)$ where $\gamma > 0$, and Laplacian RBF kernels $k(x_1, x_2) = \exp(-\gamma \|x_1 - x_2\|)$. Another useful type of kernels on Euclidean spaces is the spline kernels.

Convolution kernels: Haussler (1999) investigated how to define a kernel between composite objects by building on the similarity measures that assess their respective parts. It needs to enumerate all possible ways to decompose the objects, hence efficient algorithms like dynamic programming are needed.

Graph kernels: Graph kernels are available in two categories: between graphs and on a graph. The first type is similar to convolution kernels, which measures the similarity between two graphs. The second type defines a metric between the vertices, and is generally based on the graph Laplacian. By applying various transform functions to the eigenvalue of the graph Laplacian, various smoothing and regularization effects can be achieved.

Fisher kernels: Kernels can also be defined between probability densities $p(x|\theta)$. Let $U_\theta(x) = -\partial_\theta \log p(x|\theta)$ and $I = \mathbb{E}_x \left[U_\theta(x) U_\theta^\top(x)\right]$ be the Fisher score and Fisher information matrix respectively. Then the normalized and unnormalized Fisher kernels are defined by

$$k(x, x') = U_\theta^\top(x) I^{-1} U_\theta(x') \quad \text{and}$$
$$k(x, x') = U_\theta^\top(x) U_\theta(x'),$$

respectively. In theory, estimation using normalized Fisher kernels corresponds to regularization on the $L_2(p(\cdot|\theta))$ norm. And in the context of exponential families, the unnormalized Fisher kernels are identical to the inner product of sufficient statistics.

Kernel Function Classes

Many machine learning algorithms can be posed as functional minimization problems, and the RKHS is chosen as the candidate function set. The main advantage of optimizing over an RKHS originates from the representor theorem.

Theorem 2 (Representor theorem) *Denote by* $\Omega : [0,\infty) \mapsto \mathbb{R}$ *a strictly monotonic increasing function, by* $\mathcal{X}$ *a set, and by* $c : (\mathcal{X} \times \mathbb{R}^2)^n \mapsto \mathbb{R} \cup \{\infty\}$ *an arbitrary loss function. Then each minimizer* $f \in \mathcal{H}$ *of the regularized risk functional*

$$c((x_1,y_1,f(x_1)),\ldots,(x_n,y_n,f(x_n))) + \Omega\left(\|f\|_{\mathcal{H}}^2\right) \quad (1)$$

admits a representation of the form

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x).$$

The representer theorem is important in that although the optimization problem is in an infinite-dimensional space $\mathcal{H}$, the solution is guaranteed to lie in the span of n particular kernels centered on the training points.

The objective (1) is composed of two parts: the first part measures the loss on the training set $\{x_i, y_i\}_{i=1}^n$, which depends on f only via its value at x_i. The second part is the regularizer, which encourages small RKHS norm of f. Intuitively, this regularizer penalizes the complexity of f and prefers smooth f. When the kernel k is translation invariant, that is, $k(x_1, x_2) = h(x_1 - x_2)$, Smola, Schölkopf, and Müller (1998) showed that $\|f\|^2$ is related to the Fourier transform of h, with more penalty imposed on the high frequency components of f.

Applications

Kernels have been applied to almost all branches of machine learning.

Supervised Learning

One of the most well-known applications of kernel method is the SVM for binary classification. Its primal form can be written as

$$\begin{aligned} &\underset{w,b,\xi}{\text{minimize}}\ \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i, \\ &\text{s.t. } y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i, \text{ and } \xi_i \geq 0,\ \forall i. \end{aligned}$$

Its dual form can be written as

$$\begin{aligned} &\underset{\alpha_i}{\text{minimize}}\ \frac{1}{2\lambda}\sum_{i,j} y_i y_j \langle x_i, x_j\rangle \alpha_i \alpha_j - \sum_i \alpha_i, \\ &\text{s.t. } \sum_i y_i \alpha_i = 0,\ \alpha_i \in [0, n^{-1}],\ \forall i. \end{aligned}$$

Clearly, this can be extended to feature maps and kernels by setting $k(x_i, x_j) = \langle x_i, x_j\rangle$. The same trick can be applied to other algorithms like ν-SVM, regression, density estimation, etc. For multi-class classification and structured output classification where the possible label set $\mathcal{Y}$ can be large, kernel maximum margin machines can be formulated by introducing a joint kernel on pairs of (x_i, y) $(y \in \mathcal{Y})$, that is, the feature map takes the tuple (x_i, y). Letting $\Delta(y_i, y)$ be the discrepancy between the true label y_i and the candidate label y, the primal form is

$$\begin{aligned} &\underset{w,\xi_i}{\text{minimize}}\ \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i, \\ &\text{s.t. } \langle w, \phi(x_i, y_i) - \phi(x_i, y)\rangle \geq \Delta(y_i, y) - \xi_i,\ \forall\, i, y, \end{aligned}$$

and the dual form is

$$\begin{aligned} \underset{\alpha_{i,y}}{\text{minimize}}\ &\frac{1}{2\lambda}\sum_{(i,y),(i',y')} \alpha_{i,y}\alpha_{i',y'}\langle \phi(x_i, y_i) \\ &\quad - \phi(x_i, y), \phi(x_{i'}, y_{i'}) - \phi(x_{i'}, y')\rangle \\ &\quad - \sum_{i,y} \Delta(y_i, y)\alpha_{i,y} \\ \text{s.t.}\quad &\alpha_{i,y} \geq 0,\ \forall\, i, y; \quad \sum_y \alpha_{i,y} = \frac{1}{n},\ \forall i. \end{aligned}$$

Again all the inner products $\langle \phi(x_i, y), \phi(x_{i'}, y')\rangle$ can be replaced by the joint kernel $k((x_i, y), (x_{i'}, y'))$. Further factorization using graphical models are possible (see Taskar, Guestrin, & Koller, 2004). Notice when $\mathcal{Y} = \{1, -1\}$, setting $\phi(x_i, y) = y\phi(x_i)$ recovers the binary SVM formulation. Effective methods to optimize the dual objective include sequential minimal optimization, exponentiated gradient (Collins, Globerson, Koo, Carreras, & Bartlett, 2008), mirror descent, cutting plane, or bundle methods (Smola, Vishwanathan, & Le, 2007).

Unsupervised Learning

Data analysis can benefit from modeling the distribution of data in feature space. There we can still use the rather simple linear methods, which gives rise to nonlinear methods on the original data space. For example, the principal components analysis (PCA) can be extended to Hilbert spaces (Schölkopf et al., 1998),

which allows for image denoising, clustering, and nonlinear dimensionality reduction.

Given a set of data points $\{x_i\}_{i=1}^n$, PCA tries to find a direction d such that the projection of $\{x_i\}$ to d has the maximal variance. Mathematically, one solves:

$$\max_{d:\|d\|=1} \mathrm{Var}\{\langle x_i, d\rangle\} \iff \max_{d:\|d\|=1} d^\top \left(\frac{1}{n}\sum_i x_i x_i^\top - \frac{1}{n^2}\sum_{ij} x_i x_j^\top\right) d,$$

which can be solved by finding the maximum eigenvalue of the variance of $\{x_i\}$. Along the same line, we can map the examples to the RKHS and find the maximum variance projection direction again. Here we first center the data, that is, let the feature map be $\tilde{\phi}(x_i) = \phi(x_i) - \frac{1}{n}\sum_j \phi(x_j)$, and define a kernel $\tilde{k}$ based on the centered feature. So we have $\sum_{j=1}^n \tilde{K}_{ij} = 0$ for all i. Now the objective can be written as

$$\max_{f:\|f\|_{\mathcal{H}}=1} \mathrm{Var}\left\{\left\langle \tilde{\phi}(x_i), f\right\rangle_{\mathcal{H}}\right\} \iff \max_{f:\|f\|=1} \mathrm{Var}\{f(x_i)\} \iff \max_{f:\|f\|\le 1} \mathrm{Var}\{f(x_i)\}. \quad (2)$$

Treat the constraint $\|f\| \le 1$ as an indicator function $\Omega(\|f\|^2)$ where $\Omega(x) = 0$ if $x \le 1$ and ∞ otherwise. Then the representer theorem can be invoked to guarantee that the optimal solution is $f = \sum_i \alpha_i \tilde{k}(x_i, \cdot)$ for some $\alpha_i \in \mathbb{R}$. Plugging it into (2), the problem becomes $\max_{\alpha:\alpha^\top \tilde{K}\alpha=1} \alpha^\top \tilde{K}^2 \alpha$. To get necessary conditions for optimality, we write out the Lagrangian $L = \alpha \tilde{K}^2 \alpha - \lambda(\alpha \tilde{K} \alpha - 1)$. Setting to 0 the derivative over α, we get

$$\tilde{K}^2 \alpha = \lambda \tilde{K} \alpha. \quad (3)$$

Therefore $\alpha^\top \tilde{K}^2 \alpha = \lambda$. Although (3) does not guarantee that α is an eigenvector of $\tilde{K}$, one can show that for each λ satisfying (3) there exists an eigenvector α of $\tilde{K}$ such that $\tilde{K}\alpha = \lambda\alpha$. Hence, it is sufficient to study the eigensystem of $\tilde{K}$ just like in the vanilla PCA. Once the optimal α_i^* is obtained, any data point x can be projected to $\sum_i \alpha_i^* \tilde{k}(x_i, x)$.

More applications of kernels in unsupervised learning can be found in canonical correlation analysis, independent component analysis (Bach & Jordan, 2002), kernelized independence criteria via Hilbert space embeddings of distributions (Smola, Gretton, Song, & Schölkopf, 2007), etc.

Cross References

►Principal Component Analysis
►Support Vector Machine

Further Reading

A survey paper on kernel methods up to year 2007 is Hofmann, Schölkopf, and Smola (2008). For an introduction to SVMs and kernel methods, read Cristianini and Shawe-Taylor (2000). More comprehensive treatment can be found in Schölkopf and Smola (2002), Shawe-Taylor and Cristianini (2004), and Steinwart and Christmann (2008). As far as applications are concerned, see Lampert (2009) for computer vision and Schölkopf, Tsuda, and Vert (2004) for bioinformatics. Finally, Vapnik (1998) provides the details on statistical learning theory.

Recommended Reading

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, *68*, 337–404.

Bach, F. R., & Jordan, M. I. (2002). Kernel independent component analysis. *Journal of Machine Learning Research*, *3*, 1–48.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In D. Haussler, (Ed.), *Proceedings of the annual conference computational learning theory*, (pp. 144–152). Pittsburgh: ACM Press.

Collins, M., Globerson, A., Koo, T., Carreras, X., & Bartlett, P. (2008). Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks. *Journal of Machine Learning Research*, *9*, 1775–1822.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge: Cambridge University Press.

Haussler, D. (1999). *Convolution kernels on discrete structures* (Tech. Rep. UCS-CRL-99-10). University of California, Santa Cruz.

Hofmann, T., Schölkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning. *Annals of Statistics*, *36*(3), 1171–1220.

Lampert, C. H. (2009). Kernel methods in computer vision. *Foundations and Trends in Computer Graphics and Vision*, *4*(3), 193–285.

Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, *78*(9), 1481–1497.

Schölkopf, B., & Smola, A. (2002). *Learning with Kernels*. Cambridge: MIT Press.

Schölkopf, B., Smola, A. J., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*, 1299–1319.

Schölkopf, B., Tsuda, K., & Vert, J.-P. (2004). *Kernel methods in computational biology*. Cambridge: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.

Smola, A. J., Gretton, A., Song, L., & Schölkopf, B. (2007). A Hilbert space embedding for distributions. In *International conference on algorithmic learning theory. LNAI* (Vol. 4754, pp. 13–31). Springer, Berlin, Germany.

Smola, A. J., Schölkopf, B., & Müller, K.-R. (1998). The connection between regularization operators and support vector kernels. *Neural Networks*, *11*(5), 637–649.

Smola, A., Vishwanathan, S. V. N., & Le, Q. (2007). Bundle methods for machine learning. In D. Koller, & Y. Singer, (Eds.), *Advances in neural information processing systems* (Vol. 20). Cambridge: MIT Press.

Steinwart, I., & Christmann, A. (2008). *Support vector machines. Information Science and Statistics*. Springer, New York.

Taskar, B., Guestrin, C., & Koller, D. (2004). Max-margin Markov networks. In S. Thrun, L. Saul, & B. Schölkopf, (Eds.), *Advances in neural information processing systems* (Vol. 16, pp. 25–32). Cambridge: MIT Press.

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.

Wahba, G. (1990). *Spline models for observational data. CBMS-NSF regional conference series in applied mathematics* (Vol. 59). Philadelphia: SIAM.

Kernel Shaping

►Local Distance Metric Adaptation
►Locally Weighted Regression for Control

Kernel-Based Reinforcement Learning

►Instance-Based Reinforcement Learning

Kernels

►Gaussian Process

Kind

►Class

Knowledge Discovery

►Text Mining for Semantic Web

Kohonen Maps

►Self-Organizing Maps

L

L1-Distance

►Manhattan Distance

Label

A label is a target value that is associated with each ►object in ►training data. In ►classification learning, labels are ►classes. In ►regression, labels are numeric.

Labeled Data

Labeled data are ►data for which each ►object has an identified target value, the ►label. Labeled data are used in ►supervised learning. They stand in contrast to *unlabeled data* that are used in ►unsupervised learning.

Language Bias

Definition

A learner's language bias is the set of hypotheses that can be expressed using the hypothesis language employed by the learner.

This language bias can be implicit, or it can be defined explicitly, using a bias specification language (see ►Bias Specification Language).

Cross References

►Learning as Search

Laplace Estimate

►Rule Learning

Latent Class Model

►Mixture Model

Latent Factor Models and Matrix Factorizations

Definition

Latent Factor models are a state of the art methodology for model-based ►collaborative filtering. The basic assumption is that there exist an unknown low-dimensional representation of users and items where user-item affinity can be modeled accurately. For example, the rating that a user gives to a movie might be assumed to depend on few implicit factors such as the user's taste across various movie genres. Matrix factorization techniques are a class of widely successful Latent Factor models that attempt to find weighted low-rank approximations to the user-item matrix, where weights are used to hold out missing entries. There is a large family of matrix factorization models based on choice of loss function to measure approximation quality, regularization terms to avoid overfitting, and other domain-dependent formulations.

Lazy Learning

Geoffrey I. Webb
Monash University, Victoria, Australia

Definition

The computation undertaken by a learning system can be viewed as occurring at two distinct times, ►training time and ►consultation time. Consultation time is the time between when an ►object is presented to a system for an inference to be made and the time when the

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

inference is completed. Training time is the time prior to consultation time during which the system makes inferences from training data in preparation for consultation time. *Lazy learning* refers to any machine learning process that defers the majority of computation to consultation time. Two typical examples of lazy learning are ►instance-based learning and ►Lazy Bayesian Rules. Lazy learning stands in contrast to ►eager learning in which the majority of computation occurs at training time.

Discussion

Lazy learning can be computationally advantageous when predictions using a single ►training set will only be made for few objects. This is because only the immediate sections of the instance space that are occupied by objects to be classified need be modeled. In consequence, no computation is expended in the modeling areas of the instance space that are irrelevant to the predictions that need to be made. This can also be an advantage when a training set is frequently updated, as can be the case in ►online learning, as only the applicable portions of each model are created.

Lazy learning can help improve prediction ►accuracy by allowing a system to concentrate on deriving the best possible decision for the exact points of the instance space for which predictions are to be made. In contrast, eager learning can sometimes result in suboptimal predictions for some specific parts of the instance space as a result of trade-offs during the process of deriving a single model that seeks to minimize average error over the entire instance space.

Cross References

►Eager Learning
►Instance-Based Learning
►Locally Weighted Regression for Control
►Online Learning

Learning as Search

Claude Sammut
The University of New South Wales, Sydney NSW, Australia

Definition

Learning can be viewed as a search through the space of all sentences in a concept description language for a sentence that best describes the data. Alternatively, it can be viewed as a search through all hypotheses in a ►hypothesis space. In either case, a generality relation usually determines the structure of the search space.

Background

The input to a learning program consists of descriptions of objects from the universe (the ►training set) and, in the case of ►supervised learning, an output value associated with the example. A program is limited in the concepts that it can learn by the representational capabilities of both the ►observation language (i.e., the language used to describe the training examples) and ►hypothesis language (the language used to describe the concept). For example, if an attribute/value list is used to represent examples for an induction program, the measurement of certain attributes and not others places limits on the kinds of patterns that the learner can find. The learner is said to be *biased* by its observation language. The hypothesis language also places constraints on what may and may not be learned. For example, in the language of attributes and values, relationships between objects are difficult to represent. Whereas, a more expressive language, such as first-order logic, can easily be used to describe relationships. These biases are collectively referred to as *representation bias*.

Representational power comes at a price. Learning can be viewed as a search through the space of all sentences in a language for a sentence that best describes the data. The richer the language, the larger the search space. When the search space is small, it is possible to use "brute force" search methods. If the search space is very large, additional knowledge is required to reduce the search. Notions of generality and specificity are important for ordering the search (see ►Generalization and ►Specialization).

Representation

The representation of instances and concepts affects the way a learning system searches for concept representations.

The input to a learning program may take many forms, for example, records in a database, pages of text, images, audio, and other signals of continuous data. Very often, the raw data are transformed into *feature vectors* or *attribute/value lists*. The values of the attributes or features may be continuous or discrete.

These representation by attribute/value lists is the observation language.

The representation of the concept varies considerably, depending on the approach taken for learning. In ►instance-based learning, concepts are represented by a set of prototypical instances of the concept, so abstract representations are not constructed at all. This kind of representation is said to be *extensional*. Instance-based learning is also called ►lazy learning because the learner does little work at the time that training instances are presented. Rather, at classification time, the system must find the most similar instances to the new example. See Fig. 1.

When instances are represented as feature vectors, we can treat each feature or attribute as one dimension in a multi-dimensional space. The supervised learning problem can then be characterized as the problem of finding a surface that separates objects that belong to different classes into different regions. In the case of unsupervised learning, the problem becomes one of the finding clusters of instances in the multi-dimensional space.

Learning methods differ in the way they represent and create the discrimination surfaces. In *function approximation*, the learner searches for functions that describes the surface (Fig. 2). Function approximation methods can often produce accurate classifiers because they are capable of construction complex decision surfaces. However, the concept description is stored as a set of coefficients. Thus, the results of learning are not easily available for inspection by a human reader.

Rather than searching for discriminant functions, symbolic learning systems find expressions equivalent to sentences in some form of logic. For example, we may distinguish objects according to two attributes: size and color. We may say that an object belongs to class 3 if its color is red and its size is very small to medium. Following the notation of Michalski (1983), the classes in Fig. 3 may be written as:

$$class1 \leftarrow size = large \wedge color \in \{red, orange\}$$

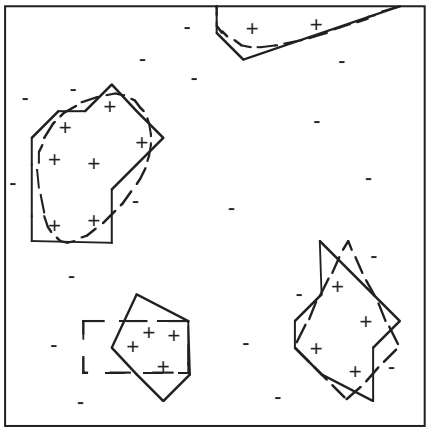
$$class2 \leftarrow size \in \{small, medium\} \wedge color \in \{oragne, yellow\}$$

$$class3 \leftarrow size \in \{v\_small \ldots medium\} \wedge color = blue$$

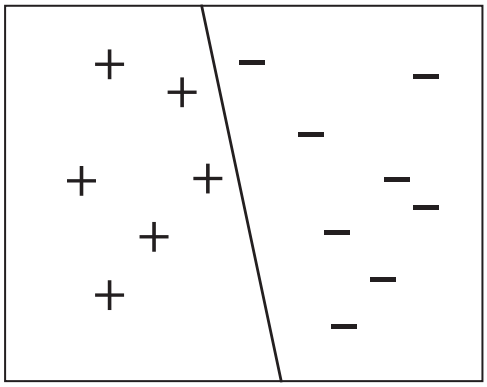
Note that this kind of description partitions the universe with axis-orthogonal surfaces, unlike the function approximation methods that find smooth surfaces to discriminate classes (Fig. 4).

Useful insights into induction can be gained by visualizing it as searching for a discrimination surface in a multi-dimensional space. However, there are limits to this geometric interpretation of learning. If we wish to learn concepts that describe complex objects and relationships between the objects, it is often useful to rely on reasoning about the concept description language itself.

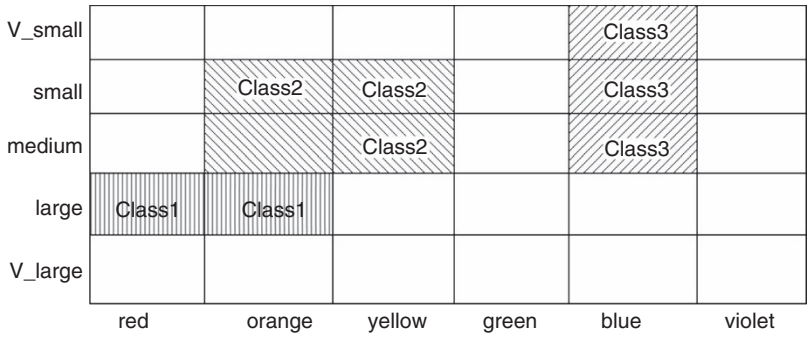
As we saw, the concepts in Fig. 3 can be expressed as clauses in propositional logic. We can establish a correspondence between sentences in the concept des-



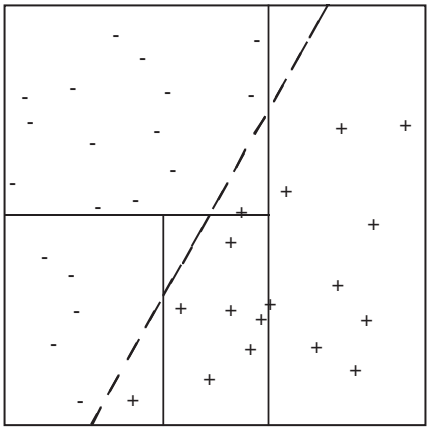
Learning as Search. Figure 1. The extension of an Instance-Based Learning concept is shown in *solid lines*. The *dashed lines* represent the target concept. A sample of positive and negative examples is shown Adapted from Aha, Kibler and Albert (1991)



Learning as Search. Figure 2. A linear discrimination between two classes



Learning as Search. Figure 3. Discrimination on attributes and values



Learning as Search. Figure 4. The dashed line shows the real division of objects in the universe. The solid lines show a decision tree approximation

cription language (the hypothesis language) and a diagrammatic representation of the concept. More importantly, we can create a correspondence between generalization and specialization operations on the sets of objects and generalization and specialization operations on the sentences of the language.

Once we have established the correspondence between sets of objects and their descriptions, it is often convenient to forget about the objects and only consider that we are working with expressions in a language. For example, the clause

$$class2 \leftarrow size = large \wedge color = red \qquad (1)$$

can be generalized to

$$class1 \leftarrow size = large \qquad (2)$$

by dropping one of the conditions. Thus, we can view learning as search through a generalization lattice that is created by applying different syntactic transformations on sentences in the hypothesis language.

Version Spaces and Subsumption

Mitchell (1977, 1982) defines the *version space* for a learning algorithm as the subset of hypotheses consistent with the training examples. That is, the hypothesis language is capable of describing a large, possibly infinite, number of concepts. When searching for the target concept, we are only interested in the subset of sentences in the hypothesis language that are consistent with the training examples, where consistent means that the examples are correctly classified. We can used the *generality* of concepts to help us limit our search to only those hypotheses in the version space.

In the above example, we stated that clause (2) is more general than clause (1). In doing so, we assumed that there is a general-to-specific ordering on the sentences in the hypothesis language. We can formalize the generality relation as follows. A hypothesis, h, is a predicate that maps an instance to *true* or *false*. That is, if $h(x)$ is true then x is hypothesized to belong to the concept being learned, the *target*. Hypothesis, h_1, is more general than or equal to h_2, if h_1 covers at least as many examples as h_2 (Mitchell, 1997). That is, $h_1 \geq h_2$ if and only if

$$(\forall x)[h_1(x) \rightarrow h_2(x)]$$

A hypothesis, h_1, is strictly more general than h_2, if $h_1 \geq h_2$ and $h_2 \not\leq h_1$.

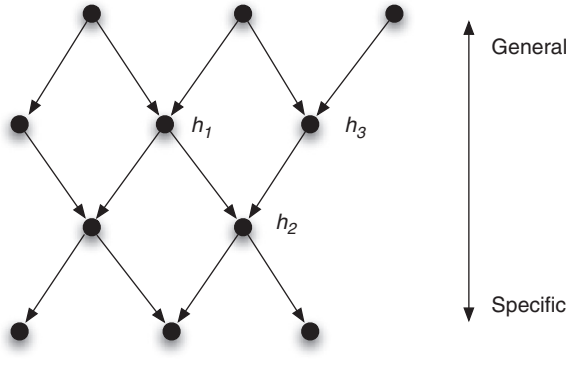
Note that the *more general than* ordering is strongly related to *subsumption* (see ▶subsumption and the

►Logic of Generality). Where the above definition of the generality relation is given in terms of the cover of a hypothesis, subsumption defines a generality ordering on expressions in the hypothesis language.

Learning algorithms can use the *more general than* relation to order their search for the best hypothesis. Because generalizations and specializations may not be unique, this relation forms a lattice over the sentences in the hypothesis language, as illustrated in Fig. 5. A search may start from the set of most specific hypotheses that fit the training data and perform a *specific-to-general* search or it may start from the set of most general hypotheses and perform a *general-to-specific* search. The search algorithm may also be bidirectional, combining both.

In Fig. 5, each node represents a hypothesis. The learning algorithm searches this lattice in an attempt to find the hypothesis that best fits the training data. Like searching in any domain, the algorithm may keep track of one node at a time, as in *depth first* or *best first* searches, or it may create a frontier of nodes as in *breadth first* or *beam searches*.

Suppose we have single-hypothesis search. A specific-to-general search may begin by randomly selecting a positive training example and creating a hypothesis that the target concept is exactly that example. Each time a new positive example is seen that is not covered by the hypothesis, the hypothesis must be *generalized*. That is, a new hypothesis is constructed that is general enough to cover all the examples covered by the previous hypothesis, as well as covering the new example. If the algorithm sees a negative example that is incorrectly covered by the current hypothesis, then the hypothesis must be *specialized*. That is, a new hypothesis is construct that is more specific than the current hypothesis such that all the positive examples that were previously covered are still covered by the new negative example is excluded.



Learning as Search. Figure 5. Generalization lattice

A similar method can be used for a general-to-specific search. In this case, the initial hypothesis is that the target concept covers every object in the universe. In both cases, the algorithm must choose how to construct either generalizations or specializations. That is, a method is needed to choose which nodes in the search to expand next. Here, the ►least general generalization (Plotkin, 1970) or the ►most general specialization are useful. These define the smallest steps that can be taken in expanding the search. For example, in Fig. 5, h_2 is the minimal specialization that can be made from h_1 or h_3 in a general-to-specific search that starts from the top of the lattice. Similarly, h_1 and h_3 are the least general generalizations of h_2. A search for the target concept can begin with an initial hypothesis and make minimal generalizations or specializations in expanding the next node in the search.

Rather than maintaining on a single current hypothesis, a search strategy may keep a set of candidate hypotheses. For example, a breadth first search generalizing from hypothesis h_2 will create a frontier for the search that is the set $\{h_1, h_3\}$. When there are many ways in which an hypothesis can be generalized or specialized, the size of the frontier set may be large. In algorithms such as Aq (Michalski, 1983) and CN2 (Clark and Niblett, 1989), a *beam search* is used. Rather than storing all possible hypotheses, the n best are kept are stored, where "best" can be defined in several ways. One metric for comparing hypotheses is given by

$$\frac{P_c + N_{\bar{c}}}{P + N}$$

where P and N are the number of positive and negative instances, respectively; P_c is the number of positive instances covered by the hypothesis; and $N_{\bar{c}}$ is the number of negative instances not covered.

Mitchell's (1997) candidate-elimination algorithm performs a bidirectional search in the hypothesis space. It maintains a set, S, of most specific hypotheses that are consistent with the training data and a set, G, of most general hypotheses consistent with the training data. These two sets form two boundaries on the version space. As new training examples are seen, the

Algorithm 1. The candidate-elimination algorithm, after Mitchell (1997)

Initialize *G* to the set of maximally general hypotheses in the hypothesis space
Initialize *S* to the maximally specific hypotheses in the hypothesis space
For each training example, *d*,
  if *d* is a positive example
    remove from *G* any hypothesis inconsistent with *d*
    For each hypothesis, *s*, in *S* that is not consistent with *d*
      remove *s* from *S*
        add all minimal generalizations, *h*, of *s* such that
          h is consistent with *d* and some member of *G* is more general than *h*
      remove from *S* any hypothesis that is more general than another hypothesis in *S*
  if *d* is a negative example
    remove from *S* any hypothesis inconsistent with *d*
    For each hypothesis, *g*, in *G* that is not consistent with *d*
      remove *g* from *G*
        add all minimal specializations, *h*, of *g* such that
          h is consistent with *d* and some member of *S* is more general than *h*
      remove from *G* any hypothesis that is less general than another hypothesis in *G*

boundaries are generalized or specialized to maintain consistency. If a new positive example is not covered by a hypothesis in *S*, then it must be generalized. If a new negative example is not rejected by an hypotheses in *G*, then it must be specialized. Any hypothesis in *G* not consistent with a positive example is removed and any hypothesis in *S* not consistent with a negative example is also removed. See Algorithm 1.

Noisy Data

Up to this point, we have assumed that the training data are free of noise. That is, all the examples are correctly classified and all the attribute values are correct. Once we relax this assumption, the algorithms described above must be modified to use approximate measures of consistency. The danger presented by noisy data is that the learning algorithm will *over fit* the training data by creating concept descriptions that try to cover the bad data as well as the good. For methods to handle noisy data see the entries in ►pruning.

Several standard texts give good introductions to search in learning, including Langley (1996), Mitchell (1997), Bratko (2000), Russell and Norvig (2009).

Cross References

►Decision Tree Learning
►Generalization
►Induction
►Instance-Based Learning
►Logic of Generality
►Rule Learning
►Subsumption

Recommended Reading

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*(1), 37–66.

Bratko, I. (2000). *Prolog programming for artificial intelligence* (3rd ed.). Boston, MA: Addison-Wesley.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*(4), 261–283.

Langley, P. (1996). *Elements of machine learning*. San Mateo: Morgan Kaufmann.

Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. Palo Alto: Tioga.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. S.

Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule-learning (pp. 305–310). In *Proceedings of the fifth international joint conference on artificial intelligence*, Cambridge.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence, 18*(2), 203–226.

Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.

Plotkin, G. D. (1970). A note on inductive generalization. In B. Meltzer & D. Michie (Eds.), *Machine intelligence* (Vol. 5, pp. 153–163). Edinburgh: Edinburgh University Press.

Russell, S., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Englewood cliffs, WJ: Prentice Hall.

Learning Bayesian Networks

►Learning Graphical Models

Learning Bias

►Inductive Bias

Learning By Demonstration

►Behavioral Cloning

Learning By Imitation

►Behavioral Cloning

Learning Classifier Systems

►Classifier Systems

Learning Control

Learning control refers to the process of acquiring a control strategy for a particular control system and a particular task by trial and error. Learning control is usually distinguished from adaptive control in that the learning system is permitted to fail during the process of learning. In contrast, adaptive control emphasizes single trial convergence without failure. Thus, learning control resembles the way that humans and animals acquire new movement strategies, while adaptive control is a special case of learning control that fulfills stringent performance constraints, e.g., as needed in life-critical systems like airplanes and industrial robots. In general, the control system can be any system that changes its state in response to a control signal, e.g., a web page with a hyperlink, a car, or a robot.

Learning Control Rules

►Behavioral Cloning

Learning Curves in Machine Learning

Claudia Perlich
IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

Synonyms

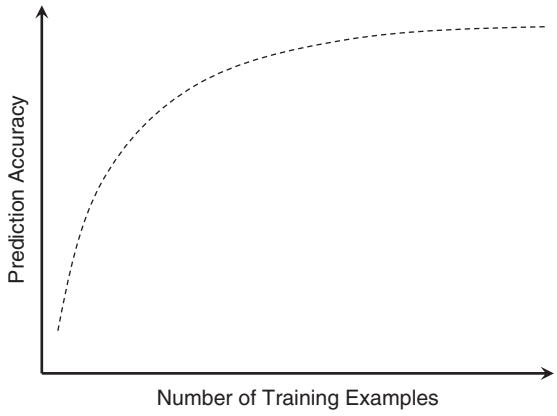
Error curve; Experience curve; Improvement curve; Training curve

Definition

A learning curve shows a measure of predictive performance on a given domain as a function of some measure of varying amounts of learning effort. The most common form of learning curves in the general field of machine learning shows predictive accuracy on the test examples as a function of the number of training examples as in Fig. 1.

Motivation and Background

Learning curves were initially introduced in educational and behavioral/cognitive psychology. The first person to describe the learning curve was Hermann Ebbinghaus in 1885 (Wozniak, 1999). He found that the time required to memorize a nonsense syllable increased sharply as the number of syllables increased. Wright (1936) described the effect of learning on labor



Learning Curves in Machine Learning. Figure 1. Stylized learning curve showing the model accuracy on test examples as a function of the number of training examples

productivity in the aircraft industry and proposed a mathematical model of the learning curve. Over time, the term has acquired related interpretation in many different fields including the above definition in machine learning and statistics.

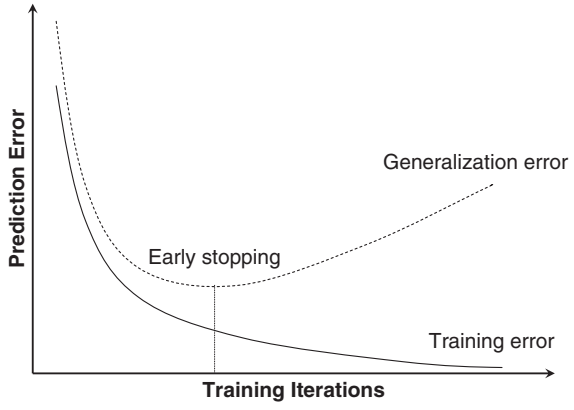
Use of Learning Curves in Machine Learning

In the area of machine learning, the term "learning curve" is used in two different contexts, the main difference being the variable on the *x*-axis of the curve.

- The ▶artificial neural network (ANN) literature has used the term to show the diverging behavior of in and out-of-sample performance as a function of the *number of training iterations* for a given number of training examples. Figure 2 shows this stylized effect.
- General machine learning uses learning curves to show the predictive ▶generalization performance as a function of the *number of training examples*. Both the graphs in Fig. 3 are examples of such learning curves.

Artificial Neural Networks

The origins of ANNs are heavily inspired by the social sciences and the goal of recreating the learning behavior of the brain. The original model of the "perceptron" mirrored closely the biological foundations of neural sciences. It is likely that the notion of learning curves was to some extent carried over from the social sciences of human learning into the field of ANNs. It shows the model error as a function of the training time measured in terms of the number of iterations. One iteration denotes in the context of neural network learning one single pass over the training data and the corresponding update of the network parameters (also called weights). The algorithm uses gradient descent minimizing the model error on the training data.



Learning Curves in Machine Learning. Figure 2. Learning curve for an artificial neural network

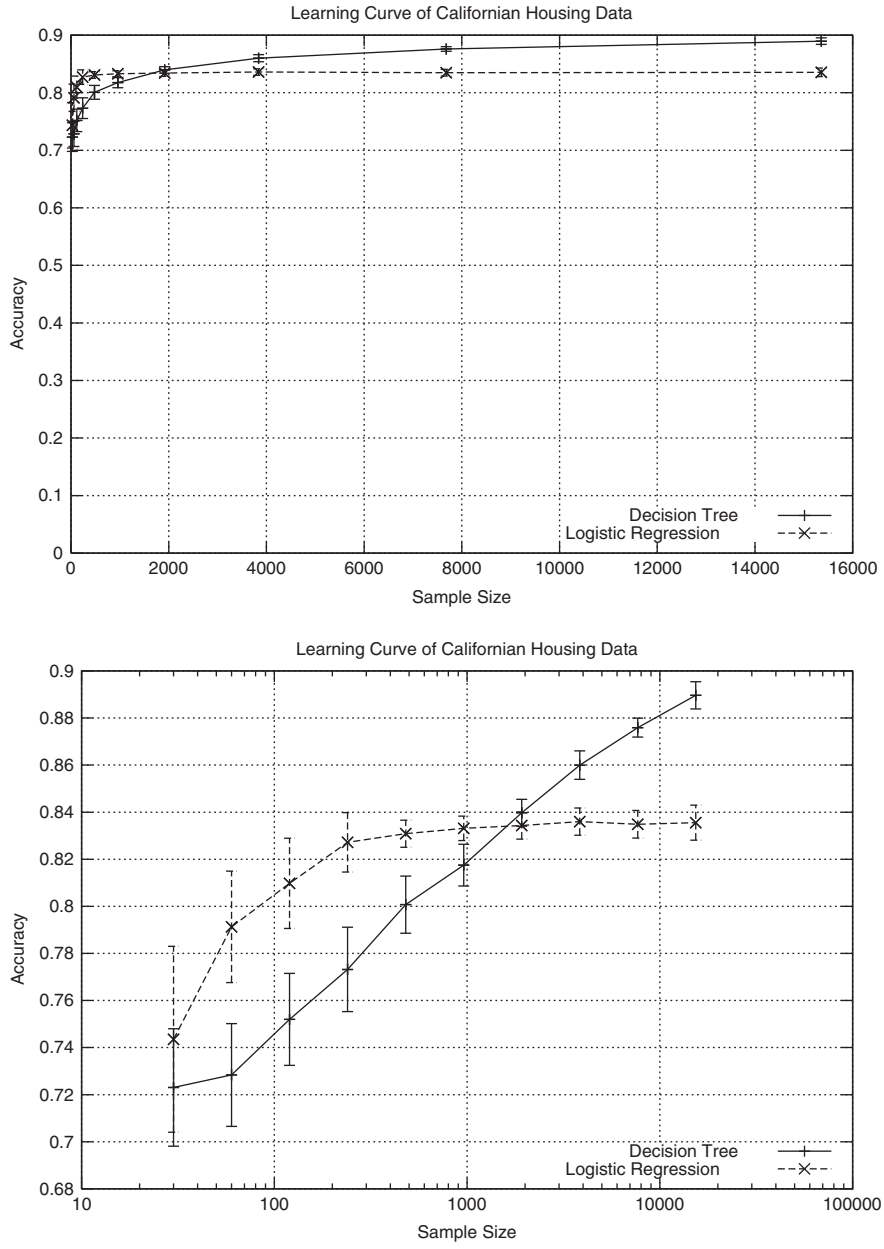
The learning curve in Fig. 2 shows the stylized effect of the relative training and generalization error on a test set as a function of the number of iterations. After initial decrease of both types of error, the generalization error reaches a minimum and starts to increase again while the training error continues to decrease.

This effect of increasing generalization error is closely related to the more general machine learning issue of ▶overfitting and variance error for models with high expressive power (or capacity). One of the initial solutions to this problem for neural networks was early stopping - some form of early regularization technique that picked the model at the minimum of the error curve on a validation subset of the data that was not used for training.

General Machine Learning

In the more general machine learning setting and statistics (Flury & Schmid, 1994), learning curves represent the generalization performance of the model as a function of the size of the training set.

Figure 3 was taken from Perlich, Provost, and Simonoff (2003) and shows two typical learning curves for two different modeling algorithms (▶decision tree and ▶logistic regression) on a fairly large domain. For smaller training-set sizes the curves are steep, but the increase in accuracy lessens for larger training-set sizes. Often for very large training-set sizes the standard representation in the upper graph obscures small, but non-trivial, gains. Therefore, to visualize the curves it is often useful to use a log scale on the horizontal axis and start the graph at the accuracy of the smallest training-set size (rather than at zero). In addition, one can include error bars that capture the estimated variance of the error over multiple experiments and provide some



Learning Curves in Machine Learning. Figure 3. Typical learning curves in original and log scale

impression of the relevance of the differences between two learning curves as shown in the graphs.

The figure also highlights a very important issue in comparative analysis of different modeling techniques: learning curves for the same domain and different models can cross. This implies an important pitfall as pointed out by Kibler and Langley (1998): "Typical empirical papers report results on training sets of fixed size, which tells one nothing about how the methods would fare given more or less data, rather than collecting learning curves …". A corollary on the above observation is the dangers of selecting an algorithm on a smaller subset of

the ultimately available training data either in the context of a proof of concept pre-study or some form of cross-validation.

Aside from its empirical relevance there has been significant theoretical work on learning curves - notably by Cortes, Jackel, Solla, Vapnik, and Denker (1994). They are addressing the question of predicting the expected generalization error from the training error of a model. Their analysis provides many additional insights about the generalization performance of different models as a function of not only training size but in addition the model capacity.

Cross References

►Artificial Neural Networks
►Computational Learning Theory
►Decision Tree
►Generalization Performance
►Logistic Regression
►Overfitting

Recommended Reading

Cortes, C., Jackel, L. D., Solla, S. A., Vapnik, V., & Denker, J. S. (1994). Learning curves: Asymptotic values and rate of convergence. *Advances in Neural Information Processing Systems, 6*, 327–334.

Flury, B. W., & Schmid, M. J. (1994). Error rates in quadratic discrimination with constraints on the covariance matrices. *Journal of Classification, 11*, 101–120.

Kibler, D., & Langley, P. (1988). Machine learning as an experimental science. In *Proceedings of the third European working session on learning*, Pittman, Glasgow (pp. 81–92). Hingham, MA: Kluwer Academic Publishers.

Perlich, C., Provost, F., & Simonoff, J. (2003). Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research, 4*, 211–255.

Shavlik, J. W., Mooney, R. J., & Towell, G. G. (1991). Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning, 6*, 111–143.

Wozniak, R. H. (1999). Introduction to memory: Hermann Ebbinghaus (1885/1913). In *Classics in the history of psychology*. Bristol, UK: Thoemmes Press.

Wright, T. P. (1936). Factors affecting the cost of airplanes. *Journal of Aeronautical Sciences, 3*(4), 122–128.

Learning from Complex Data

►Learning from Structured Data

Learning from Labeled and Unlabeled Data

►Semi-Supervised Learning

Learning from Nonpropositional Data

►Learning from Structured Data

Learning from Nonvectorial Data

►Learning from Structured Data

Learning from Preferences

►Preference Learning

Learning from Structured Data

Tamás Horváth, Stefan Wrobel
University of Bonn, Sankt Augustin, Germany

Synonyms

Learning from complex data; Learning from nonpropositional data; Learning from nonvectorial data

Definition

Learning from structured data refers to all those learning tasks where the objects to be considered as inputs and/or outputs can usefully be thought of as possessing internal structure and/or as being interrelated and dependent on each other, thus forming a structured space. Typical instances of data in structured learning tasks are sequences as they arise, e.g., in speech processing or bioinformatics, and trees or general graphs such as syntax trees in natural language processing and document analysis, molecule graphs in chemistry, relationship networks in social analysis, and link graphs in the World Wide Web. Learning from structured data presents special challenges,

since the commonly used feature vector representation and/or the i.i.d. (independently and identically distributed data) assumption are no longer applicable. Different flavors of learning from structured data are represented by (overlapping) areas such as ▶Inductive Logic Programming, ▶Statistical Relational Learning, probabilistic relational and logical learning, learning with structured outputs, sequence learning, learning with trees and graphs, ▶graph mining, and ▶collective classification.

Motivation and Background

For a long time, learning algorithms had almost exclusively considered data represented in rectangular tables defined by a fixed set of columns and a number of rows corresponding to the number of objects to be described. In this representation, each row independently and completely describes one object, each column containing the value of one particular property or feature of the object. Correspondingly, this representation is also known as feature vector representation, propositional representation, or vectorial data representation. Statistically, in such a representation, the values in each row (i.e., the objects) are assumed to be drawn i.i.d. from a fixed (but unknown) distribution.

However, when working with objects that are interrelated and/or have internal structure, this representation is no longer adequate. Consider representing chemical molecules with varying numbers of atoms and bonds in a table with a fixed number of columns. If we wanted each molecule to correspond to one row, we would have to fit the atoms and bonds into the columns, e.g., by reserving a certain number of columns for each one of them and their respective properties. To do that however, we would have to make the table wide enough to contain the largest possible molecule, resulting in many empty columns for smaller molecules, and by mapping the component atoms and bonds to columns, we would assign an order to them that would not be justified by the underlying problem and that would consequently mislead any feature vector learning algorithm.

The second issue with structured data arises from objects that are interrelated. Consider, e.g., the task of speech recognition, i.e., learning to map an acoustic unit into the corresponding lexical unit. Clearly, to solve this task, one must consider the sequence of such units, since both on the input and the output sides the probability of observing a particular unit will strongly depend on the preceding or subsequent units. The same is true, e.g., in classifying pages in the World Wide Web, where it is quite likely that the classification of the page will correlate with the classifications of neighboring pages. Therefore, any learning algorithm that would regard acoustic units or pages as independent and identically distributed objects is destined to fail, since for a successful solution the interdependencies must be modeled and exploited.

In machine learning, even though there has been interest in structured representation from the very beginning of the 1970s (cf. the systems Arch (Winston, 1975) or INDUCE (Michalski, 1983)), it was only in the 1990s, triggered by the popularity of logic programming and Horn clause representation, that learning from structured data was more intensively considered for logical representations in the subfield of Inductive Logic Programming. Outside of (what was then) machine learning, due to important applications such as speech processing, probabilistic models for sequence data such as ▶Hidden Markov Models have been considered much earlier. Toward the end of the 1990s, given an enormous surge of interest in applications in bioinformatics and the World Wide Web, and technical advances resulting from the integration of probabilistic and statistical approaches into machine learning (e.g., ▶Graphical Models and ▶kernel methods), work on learning from structured data has taken off and now represents a significant part of machine learning research in overlapping subareas such as Inductive Logic Programming, Statistical Relational Learning, probabilistic relational and logical learning, learning with structured outputs, sequence learning, learning with trees and graphs, graph mining, and collective inference.

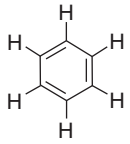
Main Tasks and Solution Approaches

A particular problem setting for learning from structured data is given by specifying, among others, (1) the language representing the input and output of the learning algorithms, (2) the type of the input and/or output data, and (3) the learning task.

1. Beyond attribute-value representation, the most intensively investigated representation languages

used in learning are ►First-Order Logic, in particular, the fragment of first-order Horn clauses, and labeled graphs. Although labeled graphs can be considered as special relational structures and thus form a special fragment of first-order logic, these two representation languages are handled separately in machine learning. As an example of first-order representation of labeled graphs, the molecular graph of a benzene ring can be represented as follows:

atom(a_1,carbon).,…,atom(a_6,carbon).,
atom(a_7,hydrogen).,…,atom(a_{12},hydrogen).,
edge(a_1,a_2,aromatic).,…,edge(a_6,a_1,aromatic).,
edge(a_1,a_7,single).,…,edge(a_6,a_{12},single).,
edge(X,Y) ← edge(Y,X).



The molecular graph of benzene rings (carbon atoms are unmarked)

Besides complexity reasons, the above two representation languages are motivated also by the difference in the matching operators typically used for these two representations. While in case of first-order logic, the matching operator is defined by logical implication or by relational homomorphism (often referred to as subsumption), which is a decidable, but thus, incomplete variant of logical implication, in case of labeled graphs it is defined by subgraph isomorphism (i.e., by injective homomorphism).

2. Another component defining a task for learning from structured data is the type of the input and/or output data (see ►Observation Language and ►Hypothesis Language). For the input, two main types can be distinguished: the instances are disjoint structures (structured instances) or substructures of some global structure (structured instance space). Molecular graphs formed by the atom-bond structure of chemical compounds are a common example of structured instances. For structured instance spaces, the web graph provides an example of a global structure; for this case, the set of instances corresponds to the set of vertices formed by the web sites. The primary goal of traditional discriminative learning is to approximate unknown target functions mapping the underlying instance space to some subset of the set of real numbers. In some of the applications, however, the elements of the range of the target function must also be structured. Such problems are referred to as learning in structured output spaces. As an example of structured output, we mention the protein secondary structure prediction problem, where the goal is to approximate the function mapping the primary structures of proteins to their secondary structures. Notice that primary and secondary structures can be represented by strings, which in turn can be considered as labeled directed paths.
3. Finally, the third component defining a problem setting is the learning task. Besides the classical learning tasks (e.g., supervised, semisupervised, unsupervised, transductive learning etc.), recent tasks include new problems such as, e.g., learning preferences (i.e., a directed graph, where an edge from vertex u to vertex v denotes that v is preferred to u), learning rankings (i.e., when the target preference relation must be a total order), etc.

Several classes of algorithms have been developed for the problem settings defined by the above components. ►Propositionalization techniques (e.g., as in LINUS (Lavrac et al., 1991)) first transform the structured data into a single table of fixed width by extracting a large number of propositional features and then use some propositional learner.

Non-propositionalization rule-based approaches follow mainly general-to-specific (top–down) or specific-to-general (bottom-up) search strategies. For top–down search (e.g., as in FOIL (Quinlan, 1990)), the crucial step of the algorithms is the definition of the refinement operators. While for graph structured data the specialization relation on the hypothesis space is usually defined by subgraph isomorphism and is therefore a partial order, for First-Order Logic it is typically defined by subsumption and is therefore only a preorder (i.e., antisymmetry does not hold), leading to undesirable algorithmic properties (e.g., incompleteness). For bottom–up search (e.g., as in GOLEM (Muggleton &

Feng, 1992)), which is less common for graph structured data, the generalization of hypotheses is usually defined by some variant of Plotkin's ►Least General Generalization operator for first-order clauses. While this generalization operator has nice algebraic properties, its application raises severe complexity issues, as the size of the hypotheses may exponentially grow in the number of examples.

Recent research in structural learning has been focusing very strongly on distance- and kernel-based approaches which in terms of accuracy have often turned out superior to rule-based approaches (e.g., in virtual screening of molecules). In such approaches, the basic algorithms carry over unchanged from the propositional case; instead, special distance (e.g., as in RIBL (Emde & Wettschereck, 1996)) or kernel functions for structural data are developed. Since even for graphs, computing any complete kernel (i.e., for which the underlying embedding function into the feature space is injective) is at least as hard as the graph isomorphism problem, most practical and efficient kernels are based on examining the structure for the occurrence of simpler parts (e.g., trees, walks, paths, and cycles) which are then counted and effectively used as feature vectors in an intersection kernel.

Finally, as a recent class of approaches, we also mention Statistical Relational Learning which extends probabilistic Graphical Models (e.g., Bayesian networks or Markov networks) with relational and logic elements (e.g., as in Alchemy (Domingos & Richardson, 2007), ICL (Poole, 2008), PRISM (Sato & Kameya, 2008)).

Applications

Virtual compound screening is a representative application example of learning from structured data. This computational problem in pharmaceutical research is concerned with the identification of chemical compounds that can be developed into drug candidates. Since current pharmaceutical compound repositories contain millions of molecules, the design of efficient algorithms for virtual compound screening has become an integral part of computer-aided drug design. One of the branches of the learning algorithms concerned with this prediction problem is based on using the compounds' 2D graph structures formed by their atoms and bonds. Depending on the representation of chemical graphs, this branch of algorithms can further be classified into logic and graph-based approaches. The first class of algorithms, developed mostly in Inductive Logic Programming, treats chemical graphs as relational structures addressing the problem to the context of learning in logic; the second class of algorithms regards them as labeled graphs addressing the problem to ►Graph Mining.

Cross References

►Hypothesis Language
►Inductive Logic Programming
►Observation Language
►Statistical Relational Learning
►Structured Induction

Recommended Reading

Cook, D., & Holder, L. (Eds.). (2007). *Mining graph data*. New York: Wiley.

De Raedt, L. (2008). *From inductive logic programming to multi-relational data mining*. Heidelberg: Springer.

Domingos, P., & Richardson, M. (2007). Markov logic: A unifying framework for statistical relational learning. In L. Getoor & B. Taskar (Eds.), *Introduction to statistical relational learning* (pp. 339–371). Cambridge, MA: MIT Press.

Emde, W., & Wettschereck, D. (1996). Relational instance based learning. In L. Saitta (Ed.), *Proceedings of the 13th international conference on machine learning* (pp. 122–130). San Francisco: Morgan Kaufmann.

Gärtner, T. (2003). A survey of kernels for structured data. *SIGKDD Explorations, 5*(1), 49–58.

Getoor, L., & Taskar, B. (Eds.). (2007). *Introduction to relational statistical learning*. Cambridge, MA: MIT Press.

Lavrac, N., Dzeroski, S., & Grobelnik, M. (1991). Learning nonrecursive definitions of relations with LINUS. In Y. Kodratoff (Ed.), *Proceedings of the 5th European working session on learning. Lecture notes in computer science* (Vol. 482, pp. 265–281). Berlin: Springer.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 83–134). San Francisco: Morgan Kaufmann.

Muggleton, S. H., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19,20*, 629–679.

Muggleton, S. H., & Feng, C. (1992). Efficient induction of logic programs. In S. Muggleton (Ed.), *Inductive logic programming* (pp. 291–298). London: Academic Press.

Poole, D. (2008). The independent choice logic and beyond. In L. De Raedt, P. Frasconi, K. Kersting, & S. Muggleton (Eds.), *Probabilistic inductive logic programming: Theory and application. Lecture notes in artificial intelligence* (Vol. 4911). Berlin: Springer.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*(3), 239–266.

Sato, T., & Kameya, Y. (2008). New advances in logic-based probabilistic modeling by PRISM. In L. De Raedt, P. Frasconi, K. Kersting, & S. Muggleton (Eds.), *Probabilistic inductive logic programming: Theory and application. Lecture notes in artificial intelligence* (Vol. 4911, pp. 118–155). Berlin: Springer.

Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision* (pp. 157–209). New York: McGraw-Hill.

Learning from Labeled and Unlabeled Data

►Semi-Supervised Text Processing

Learning Graphical Models

Kevin B. Korb
Monash University, Clayton, Victoria, Australia

Synonyms

Bayesian model averaging; Causal discovery; Dynamic bayesian network; Learning bayesian networks

Definition

Learning graphical models (see Graphical Models) means to learn a graphical representation of either a causal or probabilistic model containing the variables $X_j \in \{X_i\}$. Although graphical models include more than directed acyclic graphs (DAGs), the focus here shall be on learning DAGs, as that is where the majority of research and application is taking place.

Definition 1 (Directed acyclic graph (DAG)) *A directed acyclic graph (DAG) is a set of variables (nodes, vertices) $\{X_i\}$ and a set of directed arcs (edges) between them such that following the arcs in their given direction can never lead from a variable back to itself.*

DAGs parameterized to represent probability distributions are otherwise known as *Bayesian networks*. Some necessary concepts and notation for discussing the learning of graphical models is given in Table 1.

A key characteristic of multivariate probability distributions is the conditional independence structure they give rise to, that is, the complete list of statements of the form

$$X_A \perp\!\!\!\perp X_B | X_C$$

true of the distribution. A goal of learning DAGs is to learn a minimal DAG representation of the conditional independence structure for a distribution given the Markov condition:

Definition 2 (Markov condition) *A DAG satisfies the Markov condition relative to a probability distribution if and only if for all X_i and X_j in the DAG $X_i \perp\!\!\!\perp X_j | \pi_{X_i}$ so long as X_j is not a descendant of X_i (i.e., X_j is not in the transitive closure of the parent relation starting from X_i).*

DAGs which violate the Markov condition are not capable of fully representing the relevant probabability distribution. Upon discovering such a violation, the normal response is to fix the model by adding missing arcs. In the causal discovery literature, this condition is often referred to as the *causal Markov condition*, which simply means the arcs are being interpreted as representing causal relationships and not merely as probabilistic dependencies.

Definition 3 (Markov Blanket) *The Markov blanket (MB) of a node X_i is the minimal set X_{MB} such that for all other nodes X_j in the model $X_i \perp\!\!\!\perp X_j | X_{MB}$.*

The Markov blanket consists of a node's parents, children, and its children's other parents.

Motivation and Background

Bayesian networks have enjoyed substantial success in thousands of diverse modeling, prediction, and control applications, including medical diagnosis, epidemiology, software engineering, ecology and environmental management, agriculture, intelligence and security, finance and marketing (see, e.g., http://www.norsys.com for customers implementing such applications and more). Many of these networks have been built by the traditional process of "knowledge engineering," that is, by eliciting both structure and conditional probabilities from human domain experts. That process is limited by the availability of expertise and also by the time and cost of performing such elicitation and subsequent model validation. In domains where significant quantities of

Learning Graphical Models. Table 1 Notation

Notation	Description
X_i	a random variable
$\mathbf{X}$	a set of random variables
$\{X_i\}$	a set of random variables indexed by $i \in I$
$X = x_j$ (or, x_j)	a random variable taking the value x_j
$p(x)$	the probability that $X = x$
$X_A \perp\!\!\!\perp X_B$	X_A and X_B are *independent* (i.e., $p(X_A) = p(X_A \mid X_B)$)
$X_A \perp\!\!\!\perp X_B \mid X_C$	X_A and X_B are *conditionally independent* given X_C (i.e., $p(X_A \mid X_C) = p(X_A \mid X_B, X_C)$)
$X_A \not\perp\!\!\!\perp X_B$	X_A and X_B are *dependent* (i.e., $p(X_A) \neq p(X_A \mid X_B)$)
$X_A \not\perp\!\!\!\perp X_B \mid X_C$	X_A and X_B are *conditionally dependent* given X_C (i.e., $p(X_A \mid X_C) \neq p(X_A \mid X_B, X_C)$)
π_{X_i}	the set of parents of X_i in a DAG (i.e., nodes Y such that $Y \rightarrow X_i$)

data are available it is pertinent to consider whether automated learning of Bayesian networks might either replace or compliment knowledge engineering. A variety of techniques for doing so have been developed, and the causal discovery of Bayesian networks is by now an important subindustry of data mining.

Theory

Probability and Causality

The key to learning Bayesian networks from sample data is the relation between causal dependence and probabilistic dependence. This is most easily understood in reference to undirected chains of variables, as in Fig. 1.

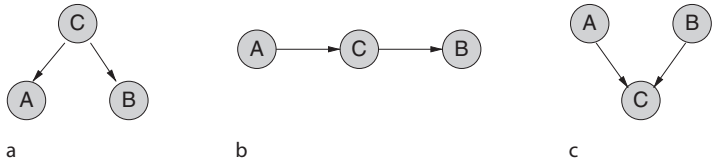
Where the arcs in Fig. 1 represent causal dependencies, then the probabilistic dependencies are as the caption describes. That is, in common causes and chains the end nodes A and B are rendered probabilistically independent of each other given knowledge of the state of C. Contrariwise, when A and B are parents of a common effect, and otherwise unrelated, they are probabilistically independent given no information (i.e., marginally independent), but *become* dependent given knowledge of C. This last relationship is often called "explaining away," because it corresponds to situations where, when already knowing the presence of, say, some disease C, the learning of the presence of a cause A reduces one's belief in some alternative explanation B of the disease.

These relationships between probabilistic dependence and causal dependence are the key for learning the causal structure of Bayesian networks because sample data allow one to estimate probabilistic dependencies directly, and the difference between conditional dependency structures in Fig. 1(a) and (b) versus its opposite in (c) allows automated learners to distinguish between these different underlying causal patterns. (This is related to *d-separation* in Graphical Models.) This distinction is explicitly made use of in constraint learners, but also implicitly used by metric learners.

In addition to structure learning, parameter learning is necessary, that is, learning the conditional probabilities of nodes given their parent values (conditional probability tables). Straightforward counting methods are frequently employed, although Expectation Maximization, Gibbs Sampling, and other techniques may come into play when the available data are noisy.

Statistical Equivalence

Two DAGs are said to be statistically equivalent (or, Markov equivalent) when they contain the same variables and each can be parameterized so as to represent any probability distribution that the other can



Learning Graphical Models. Figure 1. Causality and probabilistic dependence: (a) common cause with $A \perp\!\!\!\perp B|C$; (b) causal chain with $A \perp\!\!\!\perp B|C$; (c) common effect with $A \not\perp\!\!\!\perp B|C$

represent. Verma and Pearl (1990) proved that DAGs are statistically equivalent just in case they have the same undirected arc structures and the identical set of uncovered common effects, that is, common effects such as in Fig. 1(c) where the two parents are not themselves directly connected. They dubbed the set of statistically equivalent models *patterns*; these can be represented using partially directed acyclic graphs (PDAGs), that is, graphs with some arcs left undirected. Chickering (1995) showed that statistically equivalent models have identical maximum likelihoods relative to any given set of data. This latter result has suggested to many that causal learning programs can have no reasonable ambition to learn anything other than patterns, that is, any learner's discrimination between DAGs within a common pattern can only be based upon prior probability (e.g., prejudice). This is suggested, for example, by the fact that Bayesian learning combines (by multiplying) prior probabilities and likelihoods, so identical likelihoods will always lead to identical conclusions should the priors also be the same. One shall note some reason to doubt this supposed limit to causal discovery below.

Applications

Constraint Learners

The most direct application of the above ideas to learning Bayesian networks is exhibited in what may be called constraint learners. These programs assess conditional independencies between paired sets of variables given some other set of observed variables using statistical tests on the data, eliminating all DAGs that are incompatible with the independencies and dependencies asserted by the statistical test. (For this reason these programs are often called "conditional independence learners"; however, that tag is misleading, as is explained below.) The original such algorithm, the IC algorithm of Verma and Pearl (1990), can be described in simplified form as three rules for constructing a network from Yes or No answers to questions of the form "Is it the case that $X \perp\!\!\!\perp Y|\mathbf{W}$?"

Rule I: Put an undirected link between any two variables X and Y if and only if for every set of variables $\mathbf{W}$ s.t. $X, Y \notin \mathbf{W}$

$$X \not\perp\!\!\!\perp Y|\mathbf{W}$$

that is, X and Y are directly connected if and only if they are dependent under every conditioning set (including $\varnothing$).

Rule II: For every undirected structure $X - Y - Z$, orient the arcs $X \rightarrow Y \leftarrow Z$ if and only if

$$X \not\perp\!\!\!\perp Z|\mathbf{W}$$

for **every** $\mathbf{W}$ s.t. $X, Z \notin \mathbf{W}$ and $Y \in \mathbf{Z}$.
that is, Y is an uncovered common effect if and only if the end variables X and Z are dependent under every conditioning set that includes Y.

Rule I is justified by the need to express the probabilistic dependency between X and Y under all possible circumstances. Rule II is justified by the asymmetry in probabilistic dependencies illustrated in Fig. 1.

Application of these two rules is then followed by applying a Rule III, which just checks for any arc directions that are forced by further considerations, such as avoiding the introduction of cycles or any uncovered common effects not already identified in Rule II, and so not supported by the conditional independence tests.

This algorithm was first put into practice in the PC algorithm distributed as a part of the TETRAD program (Spirtes, Glymour, & Scheines, 1993). Aside from introducing some algorithmic efficiencies, PC adds orthodox statistical tests to answer the conditional independence questions. In the case of linear models, it uses a statistical signficance test for vanishing partial correlations,

accepting a dependence when and only when the test is statistically significant. For discrete networks a χ^2 test replaces the correlation test. Margaritis and Thrun further improve the algorithm's efficiency by limiting conditioning sets to the Markov blankets of the variable under test (Margaritis and Thrun, 2000). The PC algorithm has become the most widely used Bayesian network learner, available in `weka` and many Bayesian network modeling tools.

Metric Learners

Constraint learners attempt to build up a network using a sequence of independent statistical tests. One problem with them is that when one such test gives an incorrect result, subsequent tests will assume that result, with the potential for errors to cascade. Metric learners, by contrast, use some score applied to a network as a whole to assess it relative to the data. The earliest of this kind, by Cooper and Herskovits, turned the computation of a Bayesian measure into a counting problem. Under a number of fairly strong assumptions, such as that children variable states are always uniformly distributed given their parent states, they derived the measure

$$P(d,e) = P(d)\prod_{k=1}^{n}\prod_{j=1}^{s_{\pi(k)}}\frac{(s_k-1)!}{(S_{kj}+s_k-1)!}\prod_{l=1}^{s_k}\alpha_{kjl}!$$

where d is the DAG being scored, e the data, n the number of variables, s_k the number of values X_k may take, $s_{\pi(k)}$ the number of values the parents of X_k may take, S_{kj} the number of cases in the data where π_k takes its j-th value, and α_{kjl} is the number of cases where X_k takes its l-th value and π_k takes its j-th value. Cooper and Herskovits proved that this measure can be computed in polynomial time. Assuming the adequacy of this probability distribution, computation of the joint probability suffices for Bayesian learning, since by Bayes' Theorem maximizing $P(d,e)$ is equivalent to maximizing the posterior probability of d. Cooper and Herskovits applied this measure in the program K2, which required as inputs both the data and a total ordering of the variables. The latter input eliminates all problems about discovering arc orientations, which could be considered a cheat since, as the discussion of the IC algorithm showed, this is a part of the causal learning problem. Subsequently, Chow and Liu's (1968) maximum weighted spanning tree algorithm (MWST) has been used as a preprocessor to K2, doing a reasonable job of finding an ordering based upon the mutual information between pairs of variables.

A wide variety of alternative metrics for DAGs have been developed since K2. Heckerman, Geiger, and Chickering (1994) generalized the K2 metric to incorporate prior information, yielding BD (Bayesian metric with Dirichlet priors). Other alternatives include Minimum Description Length (MDL) scores (Bouckaert, 1993; Suzuki, 1996, 1999), Bayesian Information Criterion (BIC) (Cruz-Ramírez, Acosta-Mesa, Barrientos-Martínez, & Nava-Fernández , 2006) and Minimum Message Length (MML) (Korb & Nicholson, 2004; Wallace, Korb, & Dai, 1996). Although all of these measures score the DAG as a whole relative to some data set, they are just as (or more) sensitive to the individual dependencies and independencies between variables as are the constraint learners. The difference between the two types of learners is not whether they attend to the sets of conditional independencies expressed in the data, but whether they do so serially (which the constraint learners do) or collectively (as do the metric learners).

The question naturally arises whether constraint learners as a class are superior to metric learners or vice versa, or, indeed, which individual learner might be best. There is no settled answer to such questions, nor, in fact, is there any agreement about *how* such questions are best settled, even for fixed domains or data sets. Perhaps the issue is more general than that of learning Bayesian networks, since the fundamental theory of machine learning evaluation seems to be massively underdeveloped (see Algorithm Evaluation). In consequence, while nearly every new publication claims superiority in some sense for its preferred algorithm, the evidential basis for such claims remains suspect. It is clear, nonetheless, that many of the programs available are helpful with data analysis and are being so applied.

Search and Complexity

The space of DAGs is superexponential in the number of variables, making the learning process hard; it is NP-hard to be exact (Chickering, Heckerman, & Meek, 2004). In practice there are limits to the effectiveness of each algorithm, imposed by the number of variables (see Dimensionality), the number of

joint states the variables may take and the amount of data. The known limitations for different algorithms are scattered throughout the literature. The next section introduces some ideas for scaling up causal discovery.

Greedy search has frequently been used with both constraint-based and metric-based learning. The PC algorithm searching the space of patterns is an example, as it starts with a fully connected graph and searches greedily for arcs to remove. Chickering and Meek's Greedy Equivalence Search (GES) is another greedy algorithm operating in the pattern space (Chickering and Meek, 2002). Cooper and Herskovits' K2 is also a greedy searcher, adding arcs so long as single arc additions increases the probability score for the network. Bouckaert adopted this approach with his MDL score (Bouckaert, 1993). Greedy searches, of course, tend to get lost in local maxima, and Suzuki loosened the search method for his MDL scoring, using branch and bound (Suzuki, 1999).

Genetic algorithms (GAs) have been successfully applied to learning Bayesian networks. Larrañaga et al. used GAs over the space of total orderings to maximize the K2 score (Larrañaga, Kuijpers, Murga, & Yurramendi, 1996); Neil and Korb developed a GA searching the DAG space to maximize the MML score (Neil & Korb, 1999). A similar approach using MDL is found in Wong, Lam, and Leung (1999).

Markov Chain Monte Carlo (MCMC) searches perform stochastic sampling over the model space and have become a popular technique for Bayesian network learning. Gibbs sampling is used in Chickering and Heckerman (1997), where they compare a number of different metrics (and incorrectly conflate BIC and MDL scores; see Cruz-Ramírez, 2006) for learning a restricted class of Bayesian networks. Another MCMC approach, the Metropolis-Hastings algorithm, has been to estimate the posterior probability distribution over the space of total orderings, using the MML score (Korb & Nicholson, 2004, Chap. 8).

An alternative to model selection — searching for the single best model — is Bayesian model averaging, that is, searching for a set of models and weights for each of them (Chickering & Heckerman, 1997). And an alternative to *that* is to find a single Bayesian network that is equivalent to an averaged selection of networks (Dash & Cooper, 2004).

Markov Blanket Discovery

Recently, interest has grown in algorithms to learn, specifically, the Markov blankets around individual variables, which is a special kind of feature selection problem (see Feature Selection). One use for this is in prediction: since the MB renders all other variables conditionally independent of a target variable, finding the MB means having all the variables required for an optimal predictor. Tsamardinos et al. describe the max-min hill-climbing (MMHC) algorithm for MB discovery (Tsamardinos, Brown, & Aliferis, 2006). Nägele et al. apply this to learning in very high-dimensional spaces (Nägele, Dejori, & Stetter, 2007).

Given the MB for a target variable, one can then simply apply regression techniques (or any predictive classification technique) to the discovered variables. This works fine for standard prediction, but does not generalize to situations where some of the predictor variables are externally modified rather than observed. For an interesting collection of papers mostly applying some kind of Markov blanket discovery approach to prediction see the collection (Guyon et al., 2008).

A different use for local discovery is to avoid problems with computational complexity, whether due to the "curse of dimensionality" (too many variables) or the growing availability of very large data sets. Once the Markov blanket is found, one can employ causal discovery within the reduced set of variables, yielding local causal discovery. Iterating this will yield multiple causal subnetworks, when a global causal network might be stitched together from them (Aliferis, Statnikov, Tsamardinos, Mani, & Koutsoukos, 2010b), completing the whole causal discovery process while evading complexity problems. A current review of the issues and techniques can be found in two companion articles by Aliferis, Statnikov, Tsamardinos, Mani, and Koutsoukos (2010a, 2010b).

Knowledge Engineering with Bayesian Networks

Another approach to dealing with the complexity and tribulations of global causal discovery is to aid the discovery process with prior information. Bayesian inference is, after all, done by combining priors with likelihoods, and the priors need not always be perfectly flavorless, such as uniform priors over the DAG space. In almost all applications where data threaten to overwhelm automated discovery there is also at least some

expertise, if only the ability to say, for example, that the sex of a patient is determined before adult lifestyle practices are adopted. Such temporal information provided to a discovery algorithm can provide a huge boost to the discovery process.

This quite simple kind of prior information, the temporal tiers within which the variables may be allocated, has been available in many of the discovery programs for a long time. PC, for example, allows tiers to be specified. K2 more restrictively required a total ordering of the variables. The methods described by Heckerman, Geiger, and Chickering (1994) go beyond tiers. They provide for the specification of a network or subnetwork; the prior probability of any network in the search space can be computed according to its distance from the network provided. They also introduced the idea of equivalent sample size, that is, the weight to be given the prior information relative to the data, meaning that their priors are soft (probabilistic) rather than hard constraints. O'Donnell et al. (2006) adapted their MML score to allow soft priors for tiers, dependencies, direct and indirect causal relations, and networks or subnetworks, with variable degrees of confidence.

The flexible combination of prior information (expertise) with data in the causal discovery process allows for a full-fledged knowledge engineering process in the construction of Bayesian networks. Experts may be consulted for structural or parametric information, data may be gathered, and these different contributions may be weighted or reweighted according to the results of sensitivity analyses or other tests. The result can be a much faster and more useful approach to building and applying Bayesian networks.

Causal discovery with meaningful priors, by the way, shows that limiting discovery to patterns is insufficient: better priors, or better use of priors, can make a significant difference *within* patterns of DAGs.

Cross References

►Dimensionality
►Feature Selection
►Graphical Models
►Hidden Markov Models

Recommended Reading

The PC algorithm and variants were initially documented in Spirtes et al. (1993); their second edition (Spirtes, Glymour, & Scheines, 2000) covers more ground. Their TETRAD IV program is available from their web site http://www.phil.cmu.edu/projects/tetrad/. PC is contained within (and is available also with the weka machine learning platform at http://www.cs.waikato.ac.nz/ml/weka/).

A well-known tutorial by David Heckerman (1999) (reprinted without change in Heckerman, 2008) is well worth looking at for background in causal discovery and parameterizing Bayesian networks. A more current review of many of the topics introduced here is to be found in a forthcoming article (Daly, Shen, & Aitken, forthcoming). For other good treatments of parameterization see Cowell, Dawid, Lauritzen, and Spiegelhalter (1999) or Neapolitan (2003).

There are a number of useful anthologies in the area of learning graphical models. *Learning in Graphical Models* (Jordan, 1999) is one of the best, including Heckerman's tutorial and a variety of excellent reviews of causal discovery methods, such as Markov Chain Monte Carlo search techniques.

Textbooks treating the learning of Bayesian networks include Borgelt and Kruse (2002), Neapolitan (2003), Korb and Nicholson (2004), and Koller and Friedman (2009).

Aliferis, C. F., Statnikov, A., Tsamardinos, I., Mani, S., & Koutsoukos, X. D. (2010a). Local causal and Markov blanket induction for causal discovery and feature selection for classification. Part I: Algorithms and empirical evaluation. *Journal of Machine Learning Research, 11*, 171–234.

Aliferis, C. F., Statnikov, A., Tsamardinos, I., Mani, S., & Koutsoukos, X. D. (2010b). Local causal and Markov blanket induction for causal discovery and feature selection for classification. Part II: Analysis and extensions. *Journal of Machine Learning Research, 11*, 235–284.

Borgelt, C., & Kruse, R. (2002). *Graphical models: Methods for data analysis and mining*. New York: Wiley.

Bouckaert, R. (1993). Probabilistic network construction using the minimum description length principle. *Lecture Notes in Computer Science, 747*, 41–48.

Chickering, D. M. (1995). A tranformational characterization of equivalent Bayesian network structures. In P. Besnard & S. Hanks (Eds.), *Proceedings of the 11th conference on uncertainty in artificial intelligence*, San Francisco (pp. 87–98).

Chickering, D. M., & Heckerman, D. (1997). Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables. *Machine Learning, 29*, 181–212.

Chickering, D. M., Heckerman, D., & Meek, C. (2004). Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research, 5*, 1287–1330.

Chickering, D. M., & Meek, C. (2002). Finding optimal Bayesian networks. In *Proceedings of the 18th annual conference on uncertainty in AI*, San Francisco (pp. 94–102).

Chow, C., & Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, 14*, 462–467.

Cowell, R. G., Dawid, A. P., Lauritzen, St. L., & Spiegelhalter, D. J. (1999). *Probabilistic networks and expert systems*. New York: Springer.

Cruz-Ramírez, N., Acosta-Mesa, H. G., Barrientos-Martínez, R. E., & Nava-Fernández, L. A. (2006). How good are the Bayesian information criterion and the Minimum Description Length principle for model selection? A Bayesian network analysis. *Lecture Notes in Computer Science, 4293*, 494–504.

Daly, R., Shen, Q., & Aitken, S. (forthcoming). Learning Bayesian networks: Approaches and issues. *The Knowledge Engineering Review*.

Dash, D., & Cooper, G. F. (2004). Model averaging for prediction with discrete Bayesian networks. *Journal of Machine Learning Research, 5*, 1177–1203.

Guyon, I., Aliferis, C., Cooper, G., Elisseeff, A., Pellet, J.-P., Spirtes, P., et al. (Eds.) (2008). JMLR workshop and conference proceedings: Causation and prediction challenge (WCCI 2008), volume 3. *Journal of Machine Learning Research*.

Heckerman, D. (1999). A tutorial on learning with Bayesian networks. In M. Jordan, (Ed.), *Learning in graphical models* (pp. 301–354). Cambridge: MIT.

Heckerman, D. (2008). A tutorial on learning with Bayesian networks. In *Innovations in Bayesian networks* (pp. 33–82). Berlin: Springer Verlag.

Heckerman, D., Geiger, D., & Chickering, D. M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. In Lopes de Mantras & D. Poole (Eds.), *Proceedings of the tenth conference on uncertainty in artificial intelligence*, San Francisco (pp. 293–301).

Jordan, M. I. (1999). *Learning in graphical models*. Cambridge, MA: MIT.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: Principles and techniques*. Cambridge, MA: MIT.

Korb, K. B., & Nicholson, A. E. (2004). *Bayesian artificial intelligence*. Boca Raton, FL: CRC.

Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., & Yurramendi, Y. (1996). Learning Bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics, Part A, 26*, 487–493.

Margaritis, D., & Thrun, S. (2000). Bayesian network induction via local neighborhoods. In S. A. Solla, T. K. Leen, & K. R. Müller (Eds.), *Advances in neural information processing systems* (Vol. 12, pp. 505–511). Cambridge: MIT.

Nägele, A., Dejori, M., & Stetter, M. (2007). Bayesian substructure learning – Approximate learning of very large network structures. In *Proceedings of the 18th European conference on machine learning; Lecture notes in AI* (Vol. 4701, pp. 238–249). Warsaw, Poland.

Neapolitan, R. E. (2003). *Learning Bayesian networks*. Upper Saddle River, NJ: Prentice Hall.

Neil, J. R., & Korb, K. B. (1999). The evolution of causal models. In N. Zhong & L. Zhous (Eds.), *Third Pacific-Asia Conference on Knowledge Discovery and Datamining (PAKDD-99)*, Beijing, China (pp. 432–437). New York: Springer.

O'Donnell, R., Nicholson, A., Han, B., Korb, K., Alam, M., & Hope, L. (2006). Causal discovery with prior information. In *Australasian joint conference on artificial intelligence*, (pp. 1162–1167). New York: Springer.

Spirtes, P., Glymour, C., & Scheines, R. (1993). *Causation, prediction and search*. In *Lecture notes in statistics* (Vol. 81). New York: Springer.

Spirtes, P., Glymour, C., & Scheines, R. (2000). *Causation, prediction and search*. (2nd ed.). Cambridge: MIT.

Suzuki, J. (1996). Learning Bayesian belief networks based on the minimum description length principle. In L. Saitta (Ed.) *Proceedings of the 13th international conference on machine learning*, San Francisco (pp. 462–470). Morgan Kaufman.

Suzuki, J. (1999). Learning Bayesian belief networks based on the MDL principle: An efficient algorithm using the branch and bound technique. *IEEE Transactions on Information and Systems, 82*, 356–367.

Tsamardinos, I., Brown, L. E., & Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Machine learning, 65*(1), 31–78.

Verma, T. S., & Pearl, J. (1990). Equivalence and synthesis of causal models. In *Proceedings of the sixth conference on uncertainty in AI*, Boston (pp. 220–227). San Mateo, CA: Morgan Kaufmann.

Wallace, C. S., Korb, K. B., & Dai, H. (1996). Causal discovery via MML. In L. Saitta, (Ed.), *Proceedings of the 13th international conference on machine learning*, San Francisco (pp. 516–524). Morgan Kaufman.

Wong, M. L., Lam, W., & Leung, K. S. (1999). Using evolutionary programming and minimum description length principle for data mining of Bayesian networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 21*(2), 174–178.

Learning in Logic

►Inductive Logic Programming

Learning in Worlds with Objects

►Relational Reinforcement Learning

Learning Models of Biological Sequences

William Stafford Noble[1], Christina Leslie[2]
[1]University of Washington, Seattle, WA, USA
[2]Memorial Sloan-Kettering Cancer Center, New York, NY

Definition

Hereditary information is stored in the nucleus of every living cell as biopolymers of deoxyribonucleic acids (DNA). DNA thus encodes the blueprint for all known forms of life. A DNA sequence can be expressed as a finite string over an alphabet of {A, C, G, T}, corresponding to the four DNA bases. The human genome consists of approximately 3 billion bases, divided among 23 chromosomes.

During its life, each cell makes temporary copies of short segments of DNA. These shortlived copies are

comprised of ribonucleic acid (RNA). Each 3-mer of RNA can subsequently be translated, via the universal genetic code, into one of 20 amino acids. The resulting amino acid sequence is called a protein, and the DNA sequence that en codes the protein is called a gene.

Machine learning has been used to build models of many different types of biological sequences. These include models of short, functional elements within DNA or protein sequences, as well as models of genes, RNAs, and proteins.

Motivation and Background

Fundamentally, the motivation for building models of biological sequences is to understand the molecular mechanisms of the cell and the molecular basis for human disease. Each subheading below describes a different type of model, each of which attempts to capture a different facet of the underlying biology. All these models, ultimately, aim to uncover either evolutionary or functional relationships among sequences.

Although DNA and protein sequences were available in small numbers as early as the 1950s, a significant number of sequences were not available until the 1980s. Most of the advances in model development occurred in the 1990s, with the exception of phylogenetic models, which were already being developed in the 1970s.

Structure of Learning System

Motifs

In the context of biological sequences, a "motif" is a short (typically 6–20 letters) subsequence that is functionally significant. A motif may correspond to, e.g., the location along the DNA strand where a particular protein binds, or conversely, the location along the protein that binds to the DNA. The motif can arise either via convergent evolution (when two sequences evolve to look similar to one another) or via evolutionary conservation (if sequences that lack the motif are likely to be eliminated via natural selection).

Motif discovery is the problem of identifying a previously unknown motif within a given collection of sequences, by finding patterns that occur more often than one would expect by chance. The problem is challenging in part because two occurences of a given motif may not resemble each other exactly.

Work on motif discovery falls into two camps, based upon how the motifs themselves are represented. One camp uses position-specific scoring matrices (PSSMs), in which a motif of width w over an alphabet of size A is represented as a w-by-A probability matrix. In this matrix, each entry represents the probability that a given letter occurs at the given position. Early work in this area used expectation-maximization to identify protein motifs (Lawrence & Reilly, 1990). This effort was significantly extended in the MEME algorithm (Bailey & Elkan, 1994), which continues to be widely used today. A complementary approach uses Gibbs sampling (Lawrence, Altschul, Boguski, Liu, Neuwald, & Wootton, 1993), which offers several benefits, including avoiding local minima and the ability to sample multiple motifs simultaneously.

The other motif discovery camp uses a discrete motif representation, in which each motif is represented as a consensus sequence plus a specified maximum number of mismatches. In this formalism, enumerative methods can guarantee solving a given problem to optimality. For realistic problem sizes, this approach is most applicable to DNA, because of its much smaller alphabet size. Currently, perhaps the most popular such method is Weeder (Pavesi, Mereghetti, Mauri, & Pesole, 2004), which performed well in a recent comparison of motif discovery algorithms (Tompa, Li, Bailey, Church, Moor, Eskin, et al., 2005).

Proteins

A central problem in computational biology is the classification of proteins into functional and structural classes given their amino acid sequences. The 3D structure that a protein assumes after folding largely determines its function in the cell. However, directly obtaining a protein's 3D structure involves difficult experimental techniques such as X-ray crystallography or nuclear magnetic resonance, whereas it is relatively easy to determine a protein's sequence. Through evolution, structure is more conserved than sequence, so that detecting even very subtle sequence similarities, or remote homology, is important for predicting function.

Since the early 1980s, researchers have developed a battery of successively more powerful methods for detecting protein sequence similarities. This development can be broken into three main stages. Early methods focused on the *pairwise comparison* problem

and assessed the statistical significance of similarities between two proteins based on pairwise alignment. These methods are only capable of recognizing relatively close homologies. The BLAST algorithm (Altschul, Gish, Miller, Myers, & Lipman, 1990), based on heuristic alignment, and related tools are the most widely used methods for pairwise sequence comparison and database search today.

In the second stage, further accuracy was achieved by collecting aggregate statistics from a set of similar sequences and comparing the resulting statistics to a single, unlabeled protein of interest. One important example of *family-based models* are profile hidden Markov models (HMMs) (Krogh, Brown, Mian, Sjolander, & Haussler, 1994), probabilistic generative models estimated from a multiple alignment of sequences from a protein family. Profile HMMs generate variable length sequences by allowing insertions and deletions relative to the core residues of the alignment.

The third stage introduced *discriminative algorithms* based on classifiers like support vector machines for protein classification and remote homology detection. Such methods train both on positive sequences belonging to a protein family as well as negative examples consisting of sequences unrelated to the family. They require protein sequences to be represented using an explicit feature mapping or a kernel function in order to train the classifier. The first discriminative protein classification algorithm was the SVM-Fisher method (Jaakkola, Diekhans, & Haussler, 2000), which uses a profile HMM to extract a feature vector of Fisher scores for each input sequence x, defined by the gradient vector

$$\nabla_\theta \log P(x|\theta)|_{\theta=\theta_0},$$

where $\log P(x|\theta)$ is the log likelihood function of the sequence relative to the HMM and θ_0 is the maximum likelihood estimate for the model parameters. Another feature represention that has been used is the empirical kernel map

$$\Phi(x) = \langle s(x_1, x), \ldots, s(x_m, x)\rangle,$$

where $s(x, y)$ is a function depending on a pairwise similarity score between x and y and x_i, $i = 1 \ldots m$, are the training sequences Liano et al. (2002). In addition, it is possible to construct useful kernels directly without explicitly depending on generative models by using subsequence-based string kernels. For example, the mismatch kernel (Leslie, Eskin, Weston, & Noble, 2003) is defined by a histogram-like feature map. The feature space is indexed by all possible k-length subsequences $\alpha = a_1 a_2 \ldots a_k$, where each a_i is a character in the alphabet $\mathcal{A}$ of amino acids. The feature map is defined on k-gram α by $\Phi(\alpha) = (\phi_\beta(\alpha))_{\mathcal{A}^k}$ where $\phi_\beta(\alpha) = 1$ if α is within m mismatches of β, 0 otherwise, and is extended additively to longer sequences: $\Phi(x) = \sum_{k-\text{grams}\in x} \Phi(\alpha)$.

Genes

After a genome (or a portion of a genome) has been sequenced, a biologist's first question is usually, "Where are the genes?" In simple organisms, most of the genome is translated into proteins, and so the gene-finding problem reduces, essentially, to identifying the boundaries between genes. In more complex organisms, a large proportion of the genome consists of non protein coding DNA. The human genome, for example, is comprised of approximately 98% non-coding DNA. This non-coding DNA is interspersed between coding regions and even in the midst of a single coding region. The gene-finding problem, canonically, is to identify the regions of a given DNA sequence that encode proteins.

Initial methods for gene finding combined scores produced by different types of detectors. A *signal* detector attempts to recognize local, fixed-length features, such as characterize the boundaries between coding and non-coding regions within a single gene. A *content* detector attempts to recognize larger patterns on the basis of compositional statistics. Early gene finding algorithms combined these various scores in an *ad hoc* fashion to identify gene-like regions.

In the mid-1990s, several research groups began using HMMs for gene finding. HMMs provide a coherent, fully probabilistic method that is capable of capturing many of the complexities of real genes. Perhaps the most widely used such method is Genscan (Burge & Karlin, 1997), which uses fifth-order Markov statistics along with variable duration HMMs.

Gene finding is now a very mature field, but advances continue to be made using, e.g., conditional random field models (Bernal, Crammer, Hatzigeorgiou, & Pereira, 2007) and large-margin structured output techniques (Rätsch et al., 2007).

RNAs

Most RNA molecules are so-called messenger RNAs, which are used in the production of a corresponding protein molecule. Some RNAs, however, do not code for proteins but instead function on their own. These RNAs fall into functional categories, but they are not easily recognized by HMMs because (1) the RNAs themselves are often very short, and (2) functional RNA typically folds up in a deterministic fashion, and therefore exhibits nonlocal dependencies along the RNA sequence.

Useful RNA modeling is therefore accomplished using covariance models, which are a subclass of stochastic context-free grammars. The foundational work in this area was due to Eddy and Durbin (1994), who addressed both the structure inference problem and the inference of transition and emission probabilities given the structure. They applied these algorithms to transfer RNAs (tRNAs), and the approach was the basis for widely used tools such as Rfam.

Much effort in RNA covariance models has been devoted to improving the time and space efficiency of the algorithms associated with covariance models. For example, Eddy (2002) introduced a memory-efficient variant of the core dynamic programming algorithm used to align a covariance model to an RNA sequence. This improvement was practically important, since it reduced the $O(N^3)$ space requirement for a length N RNA sequence. Other work has focused on accelerating database search using the modeled families.

Recent efforts have focused on algorithms for genome-wide screens to discover functional noncoding RNAs as well as small regulatory RNAs like microRNAs. Various approaches to this problem have incorporated conservation as well as RNA structure prediction, both using covariance models and other methodologies. One such algorithm is RNAz (Washietl, Hofacker, & Stadler, 2005), which combines a measure for thermodynamic stability with a measure for structure conservation in an SVM approach to detect functional RNAs in multiple sequence alignments.

Phylogenetic Models

Phylogenetic models attempt to infer the series of evolutionary events (mutations, insertions, deletions, etc.) that gave rise to an observed collection of DNA or protein sequences. In most cases, these models ignore the possibility of copying DNA between individuals or species, and therefore represent the history as a phylogenetic tree, in which leaf nodes represent the observed sequences, and the internal nodes represent unobserved ancestral sequences. Of primary interest is inferring the topology and branch lengths of this tree.

Methods for phylogenetic tree inference can be divided into three classes: parsimony, distance, and likelihood methods, all described in detail in Felsenstein (2003).

Parsimony methods search for a tree that requires the smallest number of mutations, insertions or deletions along its branches. Because the search space of possible tree topologies is so large, this approach is feasible only for relatively small sets of sequences – tens rather than hundreds. Also, because parsimony models do not allow for so-called back-mutations – where a letter mutates to a different letter and then back again – and other similar events, parsimony models are provably suboptimal for distantly related sequences.

Distance methods replace parsimony with a generalized notion of distance, which may include back-mutation. A series of increasingly sophisticated distance metrics have been developed in this domain, starting with the one-parameter Jukes-Cantor model and the two-parameter Kimura model. Given an all-versus-all distance matrix, various tree inference algorithms can be used, including neighbor joining and agglomerative hierarchical clustering (called UPGMA in phylogenetics).

The third class of models use a fully probabilistic approach and attempt to infer the tree with maximum likelihood, given the observed sequences. This approach was first outlined by Felsenstein (1973), but was not computationally feasible for large sets of sequences until recently. Current methods employ Markov chain Monte Carlo methods to carry out the search.

Programs and Data

Following are some of the more popular web sites for performing biological sequence analysis:

- BLAST and PSI-BLAST (http://www.ncbi.nlm.nih.gov/BLAST) search a protein or DNA sequence database with a given, query sequence, and return a ranked list of homologs.

- MEME (http://meme.sdsc.edu) searches a given set of DNA or protein sequences for one or more recurrent motif patterns.
- HMMER (http://hmmer.janelia.org) is an HMM toolkit for training and searching with profile HMMs of proteins.
- Pfam (http://pfam.janelia.org) is a searchable library of profile HMMs corresponding to a curated collection of homologous protein domains.
- Rfam (http://rfam.janelia.org) is an analagous database of multiple sequence alignments and covariance models covering many common noncoding RNA families.
- PHYLIP (http://evolution.genetics.washington.edu/phylip.html) is a free software toolkit that includes many common phylogenetic inference algorithms.

Recommended Reading

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). A basic local alignment search tool. *Journal of Molecular Biology, 215*, 403–410.

Bailey, T. L., & Elkan, C. P. (1994). Fitting a mixture model by expectation-maximization to discover motifs in biopolymers. In R. Altman, D. Brutlag, P. Karp, R. Lathrop, & D. Searls (Eds.), *Proceedings of the second international conference on intelligent systems for molecular biology* (pp. 28–36). AAAI Press.

Bernal, A., Crammer, K., Hatzigeorgiou, A., & Pereira, F. (2007). Global discriminative learning for higher-accuracy computational gene prediction. *PLoS Computational Biology*, 3, c54.

Burge, C., & Karlin, S. (1997). Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, *268*(1), 78–94.

Eddy, S. R. (2002). A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an rna secondary structure. *BMC Bioinformatics*, *3*, 18.

Eddy, S. R., & Durbin, R. (1994). RNA sequence analysis using covariance models. *Nucleic Acids Research*, *22*, 2079–2088.

Felsenstein, J. (1973). Maximum-likelihood estimation of evolutionary trees from continuous characters. *American Journal of Human Genetics*, *25*, 471–492.

Felsenstein, J. (2003). *Inferring phylogenies*. Sunderland MA: Sinauer Associates, 2003.

Jaakkola, T., Diekhans, M., & Haussler, D. (2000). A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2), 95–114.

Krogh, A., Brown, M., Mian, I., Sjolander, K., & Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, *235*, 1501–1531.

Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., & Wootton, J. C. (1993). Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, *262*(5131), 208–214.

Lawrence, C. E., & Reilly, A. A. (1990). An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, *7*(1), 41–51.

Leslie, C., Eskin, E., Weston, J., & Noble, W. S. (2003). Mismatch string kernels for SVM protein classification. In S. Becker, Thrun, & Obermayer (Eds.) *Advances in neural information processing systems*, (pp. 1441–1448). Cambridge, MA: 2003. MIT Press.

Liao, Li and William Stafford Noble. "Combining pairwise sequence similarity and support vector machines for remote protein homology detection". In *Proceedings of the sixth annual international conferrence on research in computational molecular biology*, April 18-21, 2002. pp. 225–232.

Pavesi, G., Mereghetti, P., Mauri, G., & Pesole, G. (2004). Weeder web: Discovery of transcription factor binding sites in a set of sequences from co-regulated genes. *Nucleic Acids Research*, *32* (Web server issue), W199–203.

Rätsch, G., Sonnenburg, S., Srinivasan, J., Witte, H., Müller, K. R., Sommer, R., et al. (2007). Improving the *C. elegans* genome annotation using machine learning. *PLoS Computational Biology*, *3*(2), e20.

Tompa, M., Li, N., Bailey, T. L., Church, G. M., de Moor, B., Eskin, E., et al. (2005). Assessing Computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, *23*(1), 137–144.

Washietl, S., Hofacker, I. L., & Stadler, P. F. (2005). Fast and reliable prediction of noncoding rnas. *Proceedings of the National Academy of Sciences USA*, *102*(7), 2454–2459.

Learning Vector Quantization

Synonyms

LVQ

Definition

Learning vector quantization (LVQ) algorithms produce prototype-based classifiers. Given a set of labeled prototype vectors, each input vector is mapped to the closest prototype, and classified according to its label. The basic LVQ learning algorithm works by iteratively moving the closest prototype toward the current input if their labels are the same, and away from the input if not. Some variants of the algorithm have been shown to approximate Bayes optimal decision borders. The algorithm was introduced by Kohonen, and being prototype-based it bears close resemblance to ▶competitive learning and ▶Self-Organizing Maps. The differences are that LVQ is supervised and the prototypes are not ordered (i.e., there is no neighborhood function).

Learning with Different Classification Costs

►Cost-Sensitive Learning

Learning with Hidden Context

►Concept Drift

Learning Word Senses

►Word Sense Disambiguation

Least-Squares Reinforcement Learning Methods

Michail G. Lagoudakis
Technical University of Crete
Crete, Greece

Definition

Most algorithms for sequential decision making rely on computing or learning a value function that captures the expected long-term return of a decision at any given state. Value functions are in general complex, nonlinear functions that cannot be represented compactly as they are defined over the entire state or state-action space. Therefore, most practical algorithms rely on value function approximation methods and the most common choice for approximation architecture is a linear architecture. Exploiting the properties of linear architectures, a number of efficient learning algorithms based on least-squares techniques have been developed. These algorithms focus on different aspects of the approximation problem and deliver diverse solutions, nevertheless they share the tendency to process data collectively (batch mode) and, in general, achieve better results compared to their counterpart algorithms based on stochastic approximation.

Motivation and Background

Consider a ►Markov Decision ►Process (MDP) $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{D})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(s'|s,a)$ is a Markovian transition model, $\mathcal{R}(s,a)$ is a reward function, $\gamma \in (0,1]$ is the discount factor, and $\mathcal{D}$ is the initial state distribution. A linear approximation architecture approximates the value function $V^\pi(s)$ or $Q^\pi(s,a)$ of a stationary (stochastic) policy $\pi(a|s)$ as a linear weighted combination of linearly-independent basis functions or features ϕ:

$$\widehat{V}^\pi(s;w) = \sum_{j=1}^{k} \phi_j(s) w_j = \phi(s)^\top w$$

$$\widehat{Q}^\pi(s,a;w) = \sum_{j=1}^{m} \phi_j(s,a) w_j = \phi(s,a)^\top w.$$

The parameters or weights of the approximation are the coefficients w.

Let V^π and $\widehat{V}^\pi$ be the exact and the approximate, respectively, state value function of a policy π, both given as column vectors of size $|\mathcal{S}|$. Define $\mathbf{\Phi}_V$ as the $(|\mathcal{S}| \times k)$ matrix with elements $\phi_j(s)$, where $s \in \mathcal{S}$ span the rows and $j = 1, 2, ..., k$ span the columns. Then, $\widehat{V}^\pi$ can be expressed compactly as $\widehat{V}^\pi = \mathbf{\Phi}_V w^\pi$. Similarly, let Q^π and $\widehat{Q}^\pi$ be the exact and the approximate, respectively, state-action value function of a policy π, both given as column vectors of size $|\mathcal{S}||\mathcal{A}|$. Define $\mathbf{\Phi}_Q$ as the $(|\mathcal{S}||\mathcal{A}| \times m)$ matrix with elements $\phi_j(s,a)$, where $(s,a) \in (\mathcal{S} \times \mathcal{A})$ span the rows and $j = 1, 2, ..., m$ span the columns. Then, $\widehat{Q}^\pi$ can be expressed compactly as $\widehat{Q}^\pi = \mathbf{\Phi}_Q w^\pi$. In addition, let $\mathcal{R}$ be a vector of size $|\mathcal{S}||\mathcal{A}|$ with entries $\mathcal{R}(s,a)$ that contains the reward function, $\mathbf{P}$ be a stochastic matrix of size $(|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|)$ that contains the transition model $(\mathbf{P}((s,a),s') = \mathcal{P}(s'|s,a))$, and $\mathbf{\Pi}_\pi$ be a stochastic matrix of size $(|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|)$ that describes policy π $(\mathbf{\Pi}_\pi(s,(s,a)) = \pi(a|s))$. The state value function V^π and the state-action value function Q^π are the solutions of the linear Bellman equations

$$V^\pi = \mathbf{\Pi}_\pi(\mathcal{R} + \gamma \mathbf{P} V^\pi)$$

$$Q^\pi = \mathcal{R} + \gamma \mathbf{P} \mathbf{\Pi}_\pi Q^\pi$$

and also the fixed points of the corresponding linear Bellman operators

$$V^\pi = T_V^\pi(V^\pi), \quad \text{where } T_V^\pi(x) = \mathbf{\Pi}_\pi(\mathcal{R} + \gamma \mathbf{P} x)$$
$$Q^\pi = T_Q^\pi(Q^\pi), \quad \text{where } T_Q^\pi(x) = \mathcal{R} + \gamma \mathbf{P}\mathbf{\Pi}_\pi x.$$

If V^π and Q^π were known, they could be projected orthogonally onto the space spanned by the basis functions to obtain the optimal least-squares approximation. (For simplicity of presentation, we consider only uniform least-squares criteria in this text, but generalization to weighted least-squares criteria is possible in all cases). For the state value function we have:

$$\widehat{V}^\pi = \mathbf{\Phi}_V w^\pi = \mathbf{\Phi}_V \left(\mathbf{\Phi}_V^\top \mathbf{\Phi}_V\right)^{-1} \mathbf{\Phi}_V^\top V^\pi$$
$$w^\pi = \mathbf{\Phi}_V^{-1} \mathbf{\Phi}_V \left(\mathbf{\Phi}_V^\top \mathbf{\Phi}_V\right)^{-1} \mathbf{\Phi}_V^\top V^\pi,$$

whereas for the state-action value function we have:

$$\widehat{Q}^\pi = \mathbf{\Phi}_Q w^\pi = \mathbf{\Phi}_Q \left(\mathbf{\Phi}_Q^\top \mathbf{\Phi}_Q\right)^{-1} \mathbf{\Phi}_Q^\top Q^\pi$$
$$w^\pi = \mathbf{\Phi}_Q^{-1} \mathbf{\Phi}_Q \left(\mathbf{\Phi}_Q^\top \mathbf{\Phi}_Q\right)^{-1} \mathbf{\Phi}_Q^\top Q^\pi.$$

The learning algorithms described here strive to find a set of parameters w, such that the approximate value function is a good approximation to the true one. However, since the exact value functions are unknown, these algorithms have to rely on information contained in the Bellman equations and the Bellman operators to derive expressions that characterize a good choice for w. It has been shown that, in many cases, this kind of learning is equivalent to approximating the MDP using a linear (compressed) model and solving exactly the approximate model (Parr et al., 2008).

Bellman Residual Minimizing Approximation

An obvious approach to deriving a good approximation is to require that the approximate function satisfies the linear Bellman equation as closely as possible. Substituting the approximation $\widehat{V}^\pi$ into the Bellman equation for V^π yields an overconstrained linear system over the k parameters w^π:

$$\widehat{V}^\pi \approx \mathbf{\Pi}_\pi(\mathcal{R} + \gamma \mathbf{P}\widehat{V}^\pi)$$
$$\mathbf{\Phi}_V w^\pi \approx \mathbf{\Pi}_\pi(\mathcal{R} + \gamma \mathbf{P}\mathbf{\Phi}_V w^\pi)$$
$$(\mathbf{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P}\mathbf{\Phi}_V) w^\pi \approx \mathbf{\Pi}_\pi \mathcal{R}.$$

Solving this overconstrained system in the least-squares sense is a $(k \times k)$ system

$$(\mathbf{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P}\mathbf{\Phi}_V)^\top (\mathbf{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P}\mathbf{\Phi}_V) w^\pi = (\mathbf{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P}\mathbf{\Phi}_V)^\top \mathbf{\Pi}_\pi \mathcal{R} \quad (1)$$

whose solution is unique and minimizes $\left\| T_V^\pi(\widehat{V}^\pi) - \widehat{V}^\pi \right\|_2$. Similarly, substituting the approximation $\widehat{Q}^\pi$ into the Bellman equation for Q^π yields an overconstrained linear system over the m parameters w^π:

$$\widehat{Q}^\pi \approx \mathcal{R} + \gamma \mathbf{P}\mathbf{\Pi}_\pi \widehat{Q}^\pi$$
$$\mathbf{\Phi}_Q w^\pi \approx \mathcal{R} + \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}_Q w^\pi$$
$$(\mathbf{\Phi}_Q - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}_Q) w^\pi \approx \mathcal{R}.$$

Solving this overconstrained system in the least-squares sense is a $(m \times m)$ system

$$(\mathbf{\Phi}_Q - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}_Q)^\top (\mathbf{\Phi}_Q - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}_Q) w^\pi = (\mathbf{\Phi}_Q - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}_Q)^\top \mathcal{R} \quad (2)$$

whose solution is unique and minimizes $\left\| T_Q^\pi(\widehat{Q}^\pi) - \widehat{Q}^\pi \right\|_2$. In both cases, the solution minimizes the L_2 norm of the Bellman residual (the difference between the left-hand side and the right-hand side of the linear Bellman equation).

Least-Squares Fixed-Point Approximation

Recall that a value function is also the fixed point of the corresponding linear Bellman operator. Another way to go about finding a good approximation is to force the approximate value function to be a fixed point under the linear Bellman operator. For that to be possible, the fixed point has to lie in the space of approximate value functions, which is the space spanned by the basis functions. Even though the approximate function itself lies in that space by definition, the result of applying the linear Bellman operator to the approximation will in general be out of that space and must be projected back. Considering the orthogonal projection $\left(\mathbf{\Phi}(\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top\right)$ (which minimizes the L_2 norm) onto the column space of $\mathbf{\Phi}$, we seek an approximate value function that is invariant under one application of the linear Bellman operator followed by orthogonal projection onto the space spanned by the basis functions.

More specifically, for the state value function, we require that

$$\widehat{V}^{\pi} = \boldsymbol{\Phi}_V \left(\boldsymbol{\Phi}_V^{\top}\boldsymbol{\Phi}_V\right)^{-1} \boldsymbol{\Phi}_V^{\top} \left(T_V^{\pi}(\widehat{V}^{\pi})\right)$$
$$\boldsymbol{\Phi}_V w^{\pi} = \boldsymbol{\Phi}_V \left(\boldsymbol{\Phi}_V^{\top}\boldsymbol{\Phi}_V\right)^{-1} \boldsymbol{\Phi}_V^{\top} \left(\boldsymbol{\Pi}_{\pi}(\mathcal{R} + \gamma \mathbf{P}\boldsymbol{\Phi}_V w^{\pi})\right).$$

Note that the orthogonal projection to the column space of $\boldsymbol{\Phi}_V$ is well-defined, because the columns of $\boldsymbol{\Phi}_V$ (the basis functions) are linearly independent by definition. The expression above is equivalent to solving a $(k \times k)$ linear system

$$\boldsymbol{\Phi}_V^{\top}(\boldsymbol{\Phi}_V - \gamma \boldsymbol{\Pi}_{\pi}\mathbf{P}\boldsymbol{\Phi}_V)w^{\pi} = \boldsymbol{\Phi}_V^{\top}\boldsymbol{\Pi}_{\pi}\mathcal{R} \tag{3}$$

whose solution is guaranteed to exist for all, but finitely many, values of γ (Koller and Parr, 2000) and minimizes (in fact, zeros out) the projected Bellman residual. Since the orthogonal projection minimizes the L_2 norm, the solution w^{π} yields a value function $\widehat{V}^{\pi}$, which is the least-squares fixed-point approximation to the true state value function. Similarly, for the state-action value function, we require that

$$\widehat{Q}^{\pi} = \boldsymbol{\Phi}_Q \left(\boldsymbol{\Phi}_Q^{\top}\boldsymbol{\Phi}_Q\right)^{-1} \boldsymbol{\Phi}_Q^{\top} \left(T_Q^{\pi}(\widehat{Q}^{\pi})\right)$$
$$\boldsymbol{\Phi}_Q w^{\pi} = \boldsymbol{\Phi}_Q \left(\boldsymbol{\Phi}_Q^{\top}\boldsymbol{\Phi}_Q\right)^{-1} \boldsymbol{\Phi}_Q^{\top} \left(\mathcal{R} + \gamma \mathbf{P}\boldsymbol{\Pi}_{\pi}\boldsymbol{\Phi}_Q w^{\pi}\right).$$

This is equivalent to solving a $(m \times m)$ linear system

$$\boldsymbol{\Phi}_Q^{\top}(\boldsymbol{\Phi}_Q - \gamma \mathbf{P}\boldsymbol{\Pi}_{\pi}\boldsymbol{\Phi}_Q)w^{\pi} = \boldsymbol{\Phi}_Q^{\top}\mathcal{R} \tag{4}$$

whose solution is again guaranteed to exist for all, but finitely many, values of γ (Koller and Parr, 2000) and minimizes (in fact, zeros out) the projected Bellman residual. Since the orthogonal projection minimizes the L_2 norm, the solution w^{π} yields a value function $\widehat{Q}^{\pi}$, which is the least-squares fixed-point approximation to the true state-action value function.

Structure of Learning System

Least-Squares Temporal Difference Learning

The least-squares temporal difference (LSTD) learning algorithm (Bradtke and Barto, 1996) learns the least-squares fixed-point approximation to the state value function V^{π} of a fixed policy π. In essence, LSTD attempts to form and solve the linear system of Equation 3 using sampling. Each sample (s, r, s') indicates a minimal interaction with the unknown process, whereby in some state s, a decision was made using policy π, and reward r was observed, as well as a transition to state s'. LSTD processes a set of samples collectively to derive the weights of the approximate value function. LSTD is an on-policy method; it requires that all training samples are collected using the policy under evaluation. The LSTD algorithm is summarized in Algorithm 1.

LSTD improves upon the temporal difference (TD) learning algorithm for linear architectures by making efficient use of data and converging faster. The main advantage of LSTD over TD is the elimination of the slow stochastic approximation and the learning rate that needs careful adjustment. TD uses samples to make small modifications and then discards them. In contrast, with LSTD, the information gathered from a single sample remains present in the matrices of the linear system and is used in full every time the parameters are computed. In addition, as a consequence of the elimination of stochastic approximation, LSTD does not diverge.

LSTD(λ) (Boyan, 1999) is an extension to LSTD that spans the spectrum between LSTD ($\lambda = 0$) and ▶linear regression over Monte-Carlo returns ($\lambda = 1$) for $\lambda \in [0,1]$. LSTD(λ) for $\lambda > 0$ requires that samples come from complete episodes. RLSTD(λ) is a variant of LSTD(λ) that uses recursive least-squares techniques for efficient implementation (Xu et al., 2002).

Algorithm 1 Least-Squares Temporal Difference (LSTD)

$w = \textbf{LSTD}(D, k, \phi, \gamma)$

Input: samples D, integer k, basis functions ϕ, discount factor γ

Output: weights w of the learned state value function

$\mathbf{A} \leftarrow \mathbf{0}$ // $(k \times k)$ matrix
$b \leftarrow \mathbf{0}$ // $(k \times 1)$ vector
for each sample $(s, r, s') \in D$ **do**
  $\mathbf{A} \leftarrow \mathbf{A} + \phi(s)\left(\phi(s) - \gamma\phi(s')\right)^{\top}$
  $b \leftarrow b + \phi(s)r$
end for
$w \leftarrow \mathbf{A}^{-1}b$
return w

Bellman Residual Minimization Learning

The main idea behind LSTD can also be used to learn the Bellman residual minimization approximation to the state value function V^π of a fixed policy π. In this case, the goal is to form and solve the linear system of Equation 1 using sampling. However, the structure of the system, in this case, requires that samples are "paired," which means that two independent samples (s, r, s') and (s, r, s'') for the same state s must be drawn to perform one update. This is necessary to obtain unbiased estimates of the system matrices. Each sample (s, r, s') again indicates a minimal interaction with the unknown process, whereby in some state s, a decision was made using policy π, and reward r was observed, as well as a transition to state s'. Obtaining paired samples is trivial with a generative model (a simulator) of the process, but virtually impossible when samples are drawn directly from a physical process. This fact makes the Bellman residual minimization approximation somewhat impractical for learning, but otherwise a reasonable approach for computing approximate state value functions from the model of the process (Schweitzer and Seidmann, 1985). The learning algorithm for Bellman residual minimization is summarized in Algorithm 2.

Hybrid Least-Squares Learning

Value function learning algorithms, either in the Bellman residual minimization or in the fixed point sense, have been used within approximate policy iteration schemes for policy learning, but in practice they exhibit quite diverse performance. Fixed-point approximations tend to deliver better policies, whereas Bellman residual minimization approximations fluctuate less between different rounds of policy iteration. Motivated by a desire to combine the advantages of both approximations, some researchers have focused on learning hybrid approximations that lie somewhere between these two extremes. Johns et al. (2009) have proposed two different approaches to combine these two approximations. The first relies on a derivation that begins with the goal of minimizing a convex combination of the two objectives (Bellman residual and projected Bellman residual); the resulting learning algorithm is quite expensive as it requires the maintenance of three matrices and two vectors (as opposed to one matrix and one vector when learning a non-hybrid approximation), as well as the inversion of one of the three matrices at each update. The second approach focuses directly on a convex combination of the linear systems produced by the two extreme approximations (Equations 1 and 3); the resulting learning algorithm has the same complexity as non-hybrid algorithms and in many cases exhibits better performance than the original approximations. On the other hand, both hybrid learning algorithms still have to deal with the paired samples problem and additionally require tuning of the convex combination parameter.

Algorithm 2 Bellman Residual Minimization Learning

$w = \mathbf{BRML}(D, k, \phi, \gamma)$

Input: paired samples D, integer k, basis functions ϕ, discount factor γ

Output: weights w of the learned state value function

$\mathbf{A} \leftarrow \mathbf{0}$ // $(k \times k)$ matrix
$b \leftarrow \mathbf{0}$ // $(k \times 1)$ vector
for each pair of samples $[(s, r, s'), (s, r, s'')] \in D$ **do**
  $\mathbf{A} \leftarrow \mathbf{A} + (\phi(s) - \gamma\phi(s'))(\phi(s) - \gamma\phi(s''))^\top$
  $b \leftarrow b + (\phi(s) - \gamma\phi(s'))\, r$
end for
$w \leftarrow \mathbf{A}^{-1} b$
return w

Least-Squares Policy Evaluation

The least-squares policy evaluation (LSPE) learning algorithm (Nedić and Bertsekas, 2003), like LSTD, learns the least-squares fixed-point approximation to the state value function V^π of a fixed policy π. Both LSPE and LSTD strive to obtain the solution to the same linear system (Equation 3), but using different methods; LSPE uses an iterative method, whereas LSTD uses direct matrix inversion. Unlike LSTD, LSPE begins with some arbitrary approximation to the value function (given by a parameter vector w') and focuses on the one-step application of the Bellman operator within the lower dimensional space spanned by the basis functions aiming at producing an incremental improvement on the parameters. In that sense, LSPE can take advantage of a good initialization of the parameter vector. Given the current parameters w' and a set

Algorithm 3 Least-Squares Policy Evaluation (LSPE)

$w = \textbf{LSPE}(D, k, \phi, \gamma, w', \alpha)$

Input: samples D, integer k, basis functions ϕ, discount factor γ, weights w', stepsize α

Output: weights w of the learned state value function

$\mathbf{A} \leftarrow \mathbf{0}$ // $(k \times k)$ matrix
$b \leftarrow \mathbf{0}$ // $(k \times 1)$ vector
for each sample $(s, r, s') \in D$ **do**
  $\mathbf{A} \leftarrow \mathbf{A} + \phi(s)\phi(s)^\top$
  $b \leftarrow b + \phi(s)\left(r + \gamma\phi(s')^\top w'\right)$
end for
$\bar{w} \leftarrow \mathbf{A}^{-1} b$
$w \leftarrow \alpha w' + (1 - \alpha)\bar{w}$
return w

$\{(s_k, r_k, s'_k) : k = 0, \ldots, t\}$ of samples, LSPE first computes the solution $\bar{w}$ to the least-squares problem

$$\min_w \sum_{k=0}^{t} \left(\phi(s_k)^\top w - \left(r_k + \gamma\phi\left(s'_k\right)^\top w'\right)\right)^2$$

and then updates w' toward $\bar{w}$ using a stepsize $\alpha \in (0, 1]$. The LSPE algorithm is summarized in Algorithm 3.

The LSPE incremental update at the extreme can be performed whenever a new sample arrives or whenever a batch of samples becomes available to remedy computational costs. An efficiency improvement to LSPE is to use recursive least-squares computations, so that the least-squares problem can be solved without matrix inversion. LSPE(λ) for $\lambda \in [0, 1]$ is an extension of LSPE to multistep updates in the same spirit as LSTD(λ). LSPE(λ) for $\lambda > 0$ requires that samples come from complete episodes.

Least-Squares Policy Iteration

Least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003) is a model-free, reinforcement learning algorithm for policy learning based on the approximate policy iteration framework. LSPI learns in a batch manner by processing multiple times the same set of samples. LSPI is an off-policy method; samples can be collected arbitrarily from the process using any policy. Each sample (s, a, r, s') indicates that the learner observed the current state s, chose an action a, and observed the resulting next state s' and the reward received r. LSPI iteratively learns a (weighted) least-squares fixed-point approximation of the state-action value functions (Equation 4) of a sequence of improving (deterministic) policies π. At each iteration, the value function of the policy is approximated by solving a $(m \times m)$ linear system, formed using the single sample set and the policy from the previous iteration. LSPI offers a non-divergence guarantee and in most cases it converges in just a few iterations. LSPI exhibits excellent sample efficiency and has been used widely in many domains. Algorithm 4 summarizes the LSPI algorithm.

The default internal policy evaluation procedure in LSPI is the variation of LSTD for the state-action value function (LSTDQ). However, any other value function learning algorithm, such as BRML or LSPE, could be used instead; nevertheless, the λ extensions are not applicable in this case, because the samples in LSPI have been collected arbitrarily and not by the policy being evaluated each time. The variation of LSPI that internally learns the Bellman residual minimizing approximation (Equation 2) using BRML has produced inferior policies, in general, and suffers from the paired samples problem.

Algorithm 4 Least-Squares Policy Iteration (LSPI)

$w = \textbf{LSPI}(D, m, \phi, \gamma, \epsilon)$

Input: samples D, integer m, basis functions ϕ, discount factor γ, tolerance ϵ

Output: weights w of the learned value function of the best learned policy

$w \leftarrow \mathbf{0}$
repeat
  $\mathbf{A} \leftarrow \mathbf{0}$ // $(m \times m)$ matrix
  $b \leftarrow \mathbf{0}$ // $(m \times 1)$ vector
  $w' \leftarrow w$
  for each sample (s, a, r, s') in D **do**
    $a' = \arg\max_{a'' \in \mathcal{A}} \phi(s', a'')^\top w'$
    $\mathbf{A} \leftarrow \mathbf{A} + \phi(s, a)\left(\phi(s, a) - \gamma\phi(s', a')\right)^\top$
    $b \leftarrow b + \phi(s, a)r$
  end for
  $w \leftarrow \mathbf{A}^{-1} b$
until $(\|w - w'\| < \epsilon)$
return w

Least-Squares Fitted *Q*-Iteration

Fitted *Q*-iteration (FQI) (Ernst et al., 2005) is a batch reinforcement learning algorithm for policy learning based on the popular *Q*-Learning algorithm. FQI uses an iterative scheme to approximate the optimal value function, whereby an improved value function Q is learned at each iteration by fitting a function approximator to a set of training examples generated using a set of samples from the process and the *Q*-Learning update rule. Any function approximation architecture and the corresponding supervised learning algorithm could be used in the iteration. The simplest choice is to use least-squares regression along with a linear architecture to learn the least-squares fixed-point approximation of the state-action value function (Equation 4). This version of least-squares fitted *Q*-iteration is summarized in Algorithm 5. In a sense, this version of FQI combines ideas from LSPE and LSPI. Like LSPI, FQI is an off-policy method; samples can be collected arbitrarily from the process using any policy. In practice, FQI produces very good policies within a moderate number of iterations.

Algorithm 5 Least-Squares Fitted Q-Iteration

$w = \textbf{LS-FQI}(D, m, \phi, \gamma, N)$

Input: samples D, integer m, basis functions ϕ, discount factor γ, iterations N

Output: weights w of the learned value function of the best learned policy

$i \leftarrow 0$
$w \leftarrow \mathbf{0}$
while $(i < N)$ **do**
  $\mathbf{A} \leftarrow \mathbf{0}$ // $(m \times m)$ matrix
  $b \leftarrow \mathbf{0}$ // $(m \times 1)$ vector
  for each sample (s, a, r, s') in D **do**
    $\mathbf{A} \leftarrow \mathbf{A} + \phi(s,a)\phi(s,a)^\top$
    $b \leftarrow b + \phi(s,a)\left(r + \gamma \max_{a' \in \mathcal{A}} \{\phi(s',a')^\top w\}\right)$
  end for
  $w \leftarrow \mathbf{A}^{-1} b$
  $i \leftarrow i + 1$
end while
return w

Cross References

- ▶Curse of Dimensionality
- ▶Feature Selection
- ▶Radial Basis Functions
- ▶Reinforcement Learning
- ▶Temporal Difference Learning
- ▶Value Function Approximation

Recommended Reading

Boyan, J. A. (1999). Least-squares temporal difference learning. *Proceedings of the Sixteenth International Conference on Machine Learning*, Bled, Slovenia, pp. 49–56.

Bradtke, S. J., & Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning, 22*, 33–57.

Ernst, D., Geurts, P., & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research, 6*, 503–556.

Johns, J., Petrik, M., & Mahadevan, S. (2009). Hybrid least-squares algorithms for approximate policy evaluation. *Machine Learning, 76*(2–3), 243–256.

Koller, D., & Parr, R. (2000). Policy iteration for factored MDPs. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, Stanford, CA, USA, pp. 326–334.

Lagoudakis, M. G., Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research, 4*, 1107–1149.

Nedić, A., & Bertsekas, D. P. (2003). Least-squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems: Theory and Applications, 13*(1–2), 79–110.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., & Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning, Proceedings of the twenty-fifth international conference on machine learning, Helsinki, Finland, pp. 752–759.

Schweitzer, P. J., & Seidmann, A. (1985). Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications, 110*(6), 568–582.

Xu, X., He, H. G., & Hu, D. (2002). Efficient reinforcement learning using recursive least-squares methods. *Journal of Artificial Intelligence Research, 16*, 259–292.

Leave-One-Out Cross-Validation

Definition

Leave-one-out cross-validation is a special case of ▶cross-validation where the number of folds equals the number of ▶instances in the ▶data set. Thus, the learning algorithm is applied once for each instance, using all other instances as a ▶training set and using the selected instance as a single-item ▶test set. This process is closely

related to the statistical method of jack-knife estimation (Efron, 1982).

Cross References

►Algorithm Evaluation

Recommended Reading

Efron, B. (1982). The Jackknife, the Bootstrap and other resampling plans. In *CBMS-NSF regional conference series in applied mathematics 1982*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM).

Leave-One-Out Error

Synonyms

Hold-one-out error; LOO error

Definition

Leave-one-out error is an estimate of ►error obtained by ►leave-one-out cross-validation.

Lessons-Learned Systems

►Case-Based Reasoning

Lifelong Learning

►Cumulative Learning

Life-Long Learning

►Continual Learning

Lift

Lift is a measure of the relative utility of a ►classification rule. It is calculated by dividing the probability of the consequent of the rule, given its antecedent by the prior probability of the consequent:

$$\text{lift}(x \rightarrow y) = \text{P}(Y = y \mid X = x)/\text{P}(Y = y).$$

In practice, the probabilities are usually estimated from either ►training data or ►test data. In this case,

$$\text{lift}(x \rightarrow y) = \text{F}(Y = y \mid X = x)/\text{F}(Y = y)$$

where $\text{F}(Y = y \mid X = x)$ is the frequency with which the consequent occurs in the data in the context of the antecedent and $\text{F}(Y = y)$ is the frequency of the consequent in the data.

Linear Discriminant

Novi Quadrianto, Wray L. Buntine
RSISE, ANU and SML, NICTA, Canberra, Australia

Definition

A discriminant is a function that takes an input variable and outputs a class label for it. A linear discriminant is a discriminant that is linear in the input variables. This article focuses on one such linear discriminant function called Fisher's linear discriminant. Fisher's discriminant works by finding a projection of input variables to a lower dimensional space while maintaining a class separability property.

Motivation and Background

The curse of dimensionality (►Curse of Dimensionality) is an ongoing problem in applying statistical techniques to pattern recognition problems. Techniques that are computationally tractable in low-dimensional spaces can become completely impractical in high-dimensional spaces. Consequently, various methods have been proposed to reduce the dimensionality of the input or feature space in the hope of obtaining a more manageable problem. This relies on the fact that real data will often be confined to a region of the space having lower effective dimensionality, and in particular the directions over which important variations in the output variables occur may be so confined. For example, we can reduce a d-dimensional problem to one dimension

if we project the d-dimensional data onto a line. However, arbitrary projections will usually produce cluttered projected samples from all of the classes. Thus, the aim is to find a good projection so that the projected samples are well separated. This is exactly the goal of Fisher's linear discriminant analysis.

Fisher's Discriminant for Two-Category Problem

Given N observed training data points $\{(x_i, y_i)\}_{i=1}^{N}$ where $y_i \in \{1, \ldots, \Omega\}$ is the label for an input variable $x_i \in \mathbb{R}^d$, our task is to find the underlying discriminant function, $f : \mathbb{R}^d \to \{1, \ldots, \Omega\}$. The linear discriminant seeks a projection of d-dimensional input onto a line in the direction of $w \in \mathbb{R}^d$, such that

$$y = w^T x. \tag{1}$$

Subsequently, a class label assignment can be performed by thresholding the projected values, for example $y \geq C$ as class 1 and otherwise as class 2 for an appropriate choice of constant C. While the magnitude of w has no real significance (acts only as a scaling factor to y), the direction of w plays a crucial role. Inappropriate choice of w can result in an non-informative heavily cluttered line. However, by adjusting the components of weight w, we can find a projection that maximizes the class separability (Fig. 1). It is crucial to note that whenever the underlying data distributions are multimodal and highly overlapping, it might not be possible to find such a projection.

Consider a two-category problem, Ω_1 and Ω_2 with N_1 and N_2 number of data points, respectively. The d-dimensional sample mean is given by

$$\mu_1 = \frac{1}{N_1} \sum_{i \in \Omega_1} x_i \qquad \mu_2 = \frac{1}{N_2} \sum_{i \in \Omega_2} x_i. \tag{2}$$

The simplest class separability criterion is the separation of the projected class mean, that is we can find the weight w that maximizes

$$m_2 - m_1 = \frac{1}{N_2} \sum_{i \in \Omega_2} w^T x_i - \frac{1}{N_1} \sum_{i \in \Omega_1} w^T x_i = w^T (\mu_2 - \mu_1), \tag{3}$$

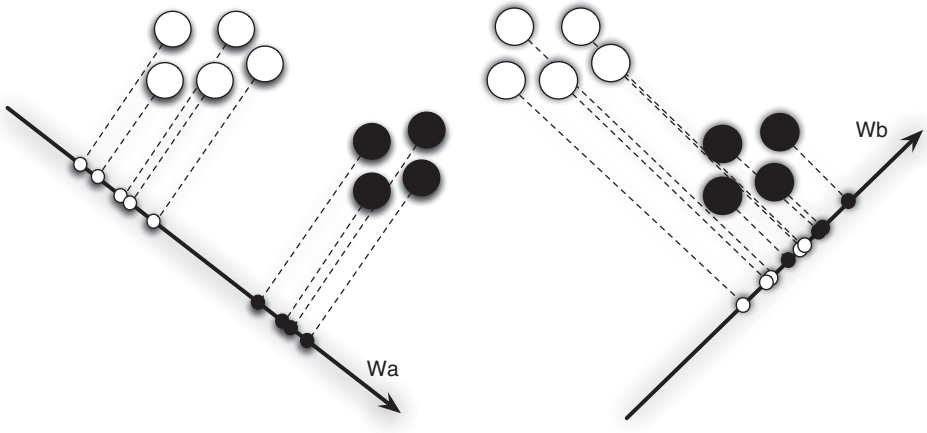
where m_1 and m_2 are the projected class means. An additional unit length constraint on w, i.e., $\sum_i w_i^2 = 1$ should be imposed to have a well-defined maximization problem. The above separability criterion produces a line that is parallel to the line joining the two means. However, this projection is sub-optimal whenever the data has distinct covariances depending on class (i.e., it is un-isotropic).

Fisher's criterion maximizes a large separation between the projected class means *while also* minimizing a variance within each class. This criterion can be expressed as

$$J(w) = \frac{w^T S_B w}{w^T S_W w}. \tag{4}$$

where the total within-class covariance matrix is

$$S_W = \sum_{i \in \Omega_1} (x_i - \mu_1)(x_i - \mu_1)^T + \sum_{i \in \Omega_2} (x_i - \mu_2)(x_i - \mu_2)^T, \tag{5}$$



Linear Discriminant. Figure 1. Colors encode class labels. Projection of samples onto two different lines. The plot on the left shows greater separation between the white and black projected points

and a between-class covariance matrix is

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T. \quad (6)$$

The maximizer of (4) can be found by setting its first derivative with respect to the weights vector to zero, that is

$$(w^T S_B w) S_W w = (w^T S_W w) S_B w. \quad (7)$$

It is clear from (6), that S_B is always in the direction of $(m_2 - m_1)$. As only the direction of w is important, we can drop the scaling factors in (7), those are $(w^T S_B w)$ and $(w^T S_W w)$. Multiplying both sides of (7) by S_W^{-1}, we can then obtain the solution of w that maximizes (4) as

$$w = S_W^{-1}(\mu_1 - \mu_2). \quad (8)$$

Fisher's Discriminant for Multi-category Problem

For the general Ω-class problem, we seek a projection from d-dimensional space to a $(\Omega - 1)$-dimensional space which is accomplished by $\Omega-1$ linear discriminant functions, that is

$$y_c = w_c^T x \qquad c = 1, \ldots, \Omega - 1. \quad (9)$$

In the matrix notation, $y = W^T x$ for $W \in \mathbb{R}^{d\times(\Omega-1)}$ and $y \in \mathbb{R}^{(\Omega-1)}$. The generalization of the within-class covariance matrix in (5) to the case of Ω classes is $S_W = \sum_{c=1}^{\Omega} S_c$ with $S_c = \sum_{i\in c}(x_i - \mu_c)(x_i - \mu_c)^T$. Following Duda and Hart (1973), the between-class covariance matrix is defined as the difference between the total covariance matrix, $\sum_{i=1}^{N}(x_i - \mu)(x_i - \mu)^T$, where μ denotes the total sample mean of the dataset, and the within-class covariance matrix. One of the criterion to be optimized is (Fukunaga, 1990)

$$J(w) = \text{Trace}((W^T S_W W)^{-1}(W^T S_B W)). \quad (10)$$

The maximizer of (10) is eigenvectors of $S_W^{-1}S_B$ associated with $\Omega - 1$ largest eigenvalues. It is important to note that the between-class covariance matrix S_B is the sum of Ω matrices of rank one or less, and because only $\Omega - 1$ of these matrices are independent, S_B has rank at most equal to $\Omega-1$ and so there are at most $\Omega-1$ nonzero eigenvalues. Therefore, we are unable to find more than $\Omega - 1$ discriminant functions (Fukunaga, 1990).

Cross References

►Regression
►Support Vector Machines

Recommended Reading

Most good statistical text books cover this.
Bellman, R. E. (1961). *Adaptive control processes*. Princeton: Princeton University Press.
Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. New York: Wiley.
Fukunaga, K. (1990). *Introduction to statistical pattern recognition* (2nd ed.). San Diego: Academic.

Linear Regression

Novi Quadrianto, Wray L. Buntine
RSISE, ANU and SML, NICTA, Canberra, Australia

Definition

Linear Regression is an instance of the ►Regression problem which is an approach to modelling a functional relationship between input variables x and an output/response variable y. In linear regression, a linear function of the input variables is used, and more generally a linear function of some vector function of the input variables $\phi(x)$ can also be used. The linear function estimates the mean of y (or more generally the median or a quantile).

Motivation and Background

Assume we are given a set of data points sampled from an underlying but unknown distribution, each of which includes input x and output y. The task of regression is to learn a hidden functional relationship between x and y from observed and possibly noisy data points, so y is to be approximated in some way by $f(x)$. This is the task covered in more detail in Regression. A general approach to the problem is to make the function $f()$ be linear. Depending on the optimization criteria used to fit between the linear function $f(x)$ and the output y, this can include many different regression techniques, but optimization is generally easier because of the linearity.

Theory/Solution

Formally, in a regression problem, we are interested in recovering a functional dependency $y_i = f(x_i) + \epsilon_i$ from N observed training data points $\{(x_i, y_i)\}_{i=1}^N$, where $y_i \in \mathbb{R}$ is the noisy observed output at input location $x_i \in \mathbb{R}^d$. For the linear parametric technique, we tackle this regression problem by parameterizing the latent regression function $f()$ by a parameter $w \in \mathbb{R}^H$, that is $f(x_i) := \langle \phi(x_i), w \rangle$ for H fixed basis functions $\{\phi_h(x_i)\}_{h=1}^H$. Note that the function is a linear function of the weight vector w. The simplest form of the linear parametric model is when $\phi(x_i) = x_i \in \mathbb{R}^d$, that is the model is also linear with respect to the input variables, $f(x_i) := w_0 + w_1 x_i^1 + \cdots + w_d x_i^d$. Here the weight w_0 allows for any constant offset in the data. With general basis functions such as polynomials, exponentials, sigmoids, or even more sophisticated Fourier or wavelets bases, we can obtain a regression function which is nonlinear with respect to the input variables although still linear with respect to the parameters.

In the subsequent section, the simplest and thus common linear parametric method for solving a regression problem is covered, the least squares method.

Least Squares Method Let $X \in \mathbb{R}^{N \times d}$ be a matrix of input variables and $y \in \mathbb{R}^N$ be a vector of output variables. The least squares method minimizes the following sum of squared error,

$$E(w) = (Xw - y)^T (Xw - y) \tag{1}$$

to infer the weight vector w. Note that the above error function is quadratic in the w, thus the minimization has a unique solution and leads to a closed-form expression for the estimated value of the unknown weight vector w. The minimizer of the error function in (1) can be found by setting its first derivative with respect to the weight vector to zero, that is

$$\partial_w E(w) = 2X^T(Xw - y) = 0 \tag{2}$$

$$w^* = (X^T X)^{-1} X^T y. \tag{3}$$

The term

$$(X^T X)^{-1} X^T := X^\dagger \tag{4}$$

is known as the Moore-Penrose pseudo-inverse (Golub & Van Loan, 1996) of the matrix X. This quantity can be regarded as a generalization of a matrix inverse to nonsquare matrices. Whenever X is square and invertible, $X^\dagger \equiv X^{-1}$. Having computed the optimal weight vector, we can then predict the output value at a novel input location x_{new} simply by taking an inner product: $y_{\text{new}} = \langle \phi(x_{\text{new}}), w^* \rangle$.

Under the assumption of an independent and normally distributed noise term, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, the above least squares approach can be shown to be equivalent to the maximum likelihood solution. With the Gaussian noise term, the log-likelihood model on an output vector y and an input matrix X is

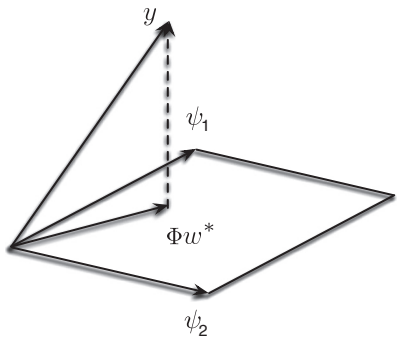
$$\ln p(y|X, w) = \ln \mathcal{N}(Xw, \sigma^2 I) \tag{5}$$

$$= -\frac{N}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y - Xw)^T(y - Xw). \tag{6}$$

Maximizing the above likelihood function with respect to w will give the optimal weight to be in the form of (3). We can also find the maximum likelihood estimate of the noise variance by setting the first derivative of (6) with respect to σ^2 to zero, that is

$$\sigma^2_{\text{ML}} = \frac{1}{N}(y - Xw)^T(y - Xw). \tag{7}$$

Geometrical Interpretation of Least Squares Method Let y be a vector in an N-dimensional space whose axes are given by $\{y_i\}_{i=1}^N$. Each of the H basis functions evaluated at N input locations can also be represented as a vector in the same N-dimensional space. For notational convenience, we denote this vector as ψ_h. The H vectors ψ_h will span a linear subspace of dimensionality H whenever the number of basis functions H is smaller than the number of input locations N (see Fig. 1). Denote $\Phi \in \mathbb{R}^{N \times H}$ as a matrix whose rows are the vectors $\{\phi_h(x_i)\}_{h=1}^H$. Our linear prediction model, Φw (in the simplest form Xw) will be an arbitrary linear combination of the vectors ψ_h. Thus, it can live anywhere in the H-dimensional space. The sum of squared error criterion in (1) then corresponds to the squared Euclidean distance between Φw and y. Therefore, the least squares solution of w corresponds to the orthogonal projection of y onto the linear subspace. This orthogonal projection is associated with the minimum of the squared Euclidean distance. As a side note, from Fig. 1, it is clear that the vector $y - \Phi w$ is normal (perpendicular) to the range of Φ thus $\Phi^T \Phi w = \Phi^T y$ is called the normal equation associated with the least squares problem.



Linear Regression. Figure 1. Geometrical interpretation of least squares. The optimal solution w^* with respect to the least squares criterion corresponds to the orthogonal projection of y onto the linear subspace which is formed by the vectors of the basis functions

Practical note: The computation of (3) requires an inversion of an H by H matrix $\Phi^T\Phi$ (or a d by d matrix X^TX). A direct inversion of this matrix might lead to numerical difficulties when two or more basis vectors ψ_h or input dimensions are (nearly) collinear. This problem can be addressed conveniently by using Singular Value Decomposition (SVD) (Press, Teukolsky, Vetterling, & Flannery, 1992). It is important to note that adding a regularization term (see also the later section on ridge regression) ensures the non-singularity of $\Phi^T\Phi$ matrix, even in the presence of degeneracies.

Sequential Learning of Least Squares Method Computation of the optimal weight vector in (3) involves the whole training set comprising N data points. This learning technique is known as a batch algorithm. Real datasets can however involve large numbers of data points which might make batch techniques computationally prohibitive. In contrast, sequential algorithms or online algorithms process one data point at a time, and can be more suited to handle large datasets.

We can use a sequential algorithm called stochastic gradient descent for learning the optimal weight vector. The objective function of (1) can be decomposed into $\sum_{i=1}^{N}(\langle x_i, w\rangle - y_i)^2$. This transformation suggests a simple stochastic gradient descent procedure: we traverse the data point i and update the weight vector using

$$w^{t+1} \leftarrow w^t - 2\eta(\langle x_i, w^t\rangle - y_i)x_i, \tag{8}$$

This algorithm is known as LMS (Least Mean Squares) algorithm. In the above equation, t denotes the iteration number and η denotes the learning rate. The value of η needs to be chosen carefully to ensure the convergence of the algorithm.

Regularized/Penalized Least Squares Method The issue of over-fitting as mentioned in Regression is usually addressed by introducing a regularization or penalty term to the objective function. The regularized objective function is now in the form of

$$E_{\text{reg}} = E(w) + \lambda R(w). \tag{9}$$

Here $E(w)$ measures the quality (such as least squares quality) of the solution on the observed data points, $R(w)$ penalizes complex solutions, and λ is called the regularization parameter which controls the relative importance between the two. This regularized formulation is sometimes called *coefficient shrinkage* as it shrinks coefficients/weights toward zero (c.f. *coefficient subset selection* formulation where the best k out of H basis functions are greedily selected). Two simple penalty terms $R(w)$ are given next, but more generally measures of curvature can also be used to penalize non-smooth functions.

Ridge regression The regularization term is in the form of

$$R(w) = \sum_{d=1}^{D} w_d^2. \tag{10}$$

Considering $E(w)$ to be in the form of (1), the regularized least squares quality function is now

$$(Xw - y)^T(Xw - y) + \lambda w^T w. \tag{11}$$

Since the additional term is a quadratic of w, the regularized objective function is still quadratic in w, thus the optimal solution is unique and can be found in closed form. As before, setting the first derivative of (11) with respect to w to zero, the optimal weight vector is in the form of

$$\partial_w E_{\text{reg}}(w) = 2X^T(Xw - y) + 2\lambda w = 0 \tag{12}$$

$$w^* = (X^TX + \lambda I)^{-1}X^Ty. \tag{13}$$

The effect of the regularization term is to put a small weight for those basis functions which are useful only in a minor way as the penalty for small weights is very small.

Lasso regression The regularization term is in the form of

$$R(w) = \sum_{d=1}^{D} |w_d|. \quad (14)$$

In contrast to ridge regression, lasso regression (Tibshirani, 1996) has no closed-form solution. In fact, the non-differentiability of the regularization term has produced many approaches. Most of the methods involve quadratic programming and recently coordinate-wise descent algorithms for large lasso problems (Friedman et al., 2007). Lasso regression leads to sparsity in w, that is, only a subset of w is nonzero, so irrelevant basis functions will be ignored.

Cross References

►Correlation Matrix
►Gaussian Processes
►Regression

Recommended Reading

Statistical textbooks and machine learning textbooks, such as Bishop (2006) among others, introduce different linear regression models. For a large variety of built-in linear regression techniques, refer to R (http://www.r-project.org/).

Bishop, C. (2006). *Pattern recognition and machine learning*. New York: Springer.

Friedman, J., Hastie, T., Hölfling, H., & Tibshirani, R. (2007). Pathwise coordinate optimization. *Annals of statistics, 1*(2): 302–332.

Golub, G.H., & Van Loan, C.F. (1996). *Matrix computations* (3rd ed.). Baltimore: John Hopkins University Press.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., & Flannery, B.P. (1992). *Numerical recipes in C: The art of scientific computing* (2nd ed.). Cambridge: Cambridge University Press. ISBN 0-521-43108-5.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, **58**, 267–288.

Linear Regression Trees

►Model Trees

Linear Separability

Two classes are linearly separable if there exists a hyperplane that separates the data for each of the classes.

Cross References

►Perceptrons
►Support Vector Machines

Link Analysis

►Link Mining and Link Discovery

Link Mining and Link Discovery

Lise Getoor
University of Maryland, College Park, MD, USA

Synonyms

Link analysis; Network analysis

Definition

Many domains of interest today are best described as a linked collection of interrelated objects. Datasets describing these domains may describe homogeneous networks, in which there is a single-object type and link type, or richer, heterogeneous networks, in which there may be multiple object and link types (and possibly other semantic information). Examples of homogeneous networks include social networks, such as people connected by friendship links, or the WWW, a collection of linked web pages. Examples of heterogeneous networks include those in medical domains describing patients, diseases, treatments and contacts, or bibliographic domains describing publications, authors, and venues. *Link mining* refers to data mining techniques that explicitly consider these links when building predictive or descriptive models of the linked data. Commonly addressed link mining tasks include collective classification, object ranking, group detection, link prediction, and subgraph discovery. Additional important

components include entity resolution, and other data cleaning and data mapping operations.

Motivation and Background

"Links," or more generically "relationships," among data instances are ubiquitous. These links often exhibit patterns that can indicate properties of the data instances such as the importance, rank, or category of the instances. In some cases, not all links will be observed; therefore, we may be interested in predicting the existence of links between instances. Or, we may be interested in identifying unusual or anomalous links. In other domains, where the links are evolving over time, our goal may be to predict whether a link will exist in the future, given the previously observed links. By taking links into account, more complex patterns may be discernable as well. This observation leads to other challenges focused on discovering substructures, such as communities, groups, or common subgraphs. In addition, links can also help in the process of ►entity resolution, or figuring out when two instance references refer to the same underlying entity.

Link mining is a newly emerging research area that is at the intersection of the work in link analysis (Feldman, 2002; Jensen & Goldberg, 1998) hypertext and web mining (Chakrabarti, 2002), ►relational learning and ►inductive logic programming (Raedt, 2008), and ►graph mining (Cook & Holder, 2000). We use the term link mining to put a special emphasis on the links – moving them up to first-class citizens in the data analysis endeavor.

Theory/Solution

Traditional data mining algorithms such as ►association rule mining, market basket analysis, and cluster analysis commonly attempt to find patterns in a dataset characterized by a collection of independent instances of a single relation. This is consistent with the classical statistical inference problem of trying to identify a model given an independent, identically distributed (IID) sample. One can think of this process as learning a model for the node attributes of a homogeneous graph while ignoring the links between the nodes.

A key emerging challenge for data mining is tackling the problem of mining richly structured, heterogeneous datasets. These kinds of datasets are commonly described as networks or graphs. The domains often consist of a variety of object types; the objects can be linked in a variety of ways. Thus, the graph may have different node and edge (or hyperedge) types. Naively applying traditional statistical inference procedures, which assume that instances are independent, can lead to inappropriate conclusions about the data (Jensen, 1999). Care must be taken that potential correlations due to links are handled appropriately. In fact, object linkage is knowledge that should be exploited. This information can be used to improve the predictive accuracy of the learned models: attributes of linked objects are often correlated, and links are more likely to exist between objects that have some commonality. In addition, the graph structure itself may be an important element to include in the model. Structural properties such as degree and connectivity can be important indicators.

Data Representation

While data representation and feature selection are significant issues for traditional machine learning algorithms, data representation for linked data is even more complex. Consider a simple example from Singh et al. (2005) of a social network describing actors and their participation in events. Such social networks are commonly called *affiliation networks* (Wasserman & Faust, 1994), and are easily represented by three tables representing the actors, the events, and the participation relationships. Even this simple structure can be represented as several distinct graphs. The most natural representation is a bipartite graph, with a set of actor nodes, a set of event nodes, and edges that represent an actor's participation in an event. Other representations may enable different insights and analysis. For example, we may construct a network in which the actors are nodes and edges correspond to actors who have participated in an event together. This representation allows us to perform a more actor-centric analysis. Alternatively, we may represent these relations as a graph in which the events are nodes, and events are linked if they have an actor in common. This representation may allow us to more easily see connections between events.

This flexibility in the representation of a graph arises from a basic graph representation duality. This duality is illustrated by the following simple example: Consider

a data set represented as a simple $G = (\mathbf{O}, \mathbf{L})$, where $\mathbf{O}$ is the set of objects (i.e., the nodes or vertices) and $\mathbf{L}$ is the set of links (i.e., the edges or hyperedges). The graph $G(\mathbf{O}, \mathbf{L})$ can be transformed into a new graph $G'(\mathbf{O}', \mathbf{L}')$, in which the links l_i, l_j in G are objects in G' and there exists an link between $o_i, o_j \in \mathbf{O}'$ if and only if l_i and l_j share an object in G. This basic graph duality illustrates one kind of simple data representation transformation. For graphs with multiple node and edge types, the number of possible transformations becomes immense. Typically, these reformulations are not considered as part of the link mining process. However, the representation chosen can have a significant impact on the quality of the statistical inferences that can be made. Therefore, the choice of an appropriate representation is actually an important issue in effective link mining, and is often more complex than in the case where we have IID data instances.

Link Mining Tasks

Link mining puts a new twist on some classic data mining tasks, and also poses new problems. One way to understand the different types of learning and inference problems is to categorize them in terms of the components of the data that are being targeted. Table 1 gives a simple characterization. Note that for the object-related tasks, even though we are concerned with classifying, clustering, consolidating, or ranking the objects, we will be exploiting the links. Similarly for link-related tasks, we can use information about the objects that participate in the links, and their links to other objects and so on.

Link Mining and Link Discovery. Table 1 A simple categorization of different link mining tasks

1. Object-related tasks
a. Object classification (collective classification)
b. Object clustering (group detection)
c. Object consolidation (entity resolution)
d. Object ranking
2. Link-related tasks
a. Link labeling/classification
b. Link prediction
c. Link ranking
3. Graph-related tasks
a. Subgraph discovery
b. Graph classification

In addition, because of the underlying link structure, link mining affords the opportunity for inferences and predictions to be *collective* or dependent on one another. The simplest example of this is in collective classification, where the inferred label of one node can depend on the inferred label of its neighbors. There are a variety of ways of modeling and exploiting this dependence. Methods include performing joint inference in the appropriate probabilistic model, use of information diffusion models, constructing and optimizing the appropriate structured prediction using a max margin approach, and others.

Additional information on different link mining subtasks is provided in separate entries on *collective classification*, *entity resolution*, *group detection*, and *link prediction*. Related problems and techniques can be found in the entries on *relational learning*, *graph mining*, and *inductive logic programming*.

Cross References

- ▶Collective Classification
- ▶Entity Resolution
- ▶Graph Clustering
- ▶Graph Mining
- ▶Group Detection
- ▶Inductive Logic Programming
- ▶Link Prediction
- ▶Relational Learning

Recommended Reading

Chakrabarti, S. (2002). *Mining the web*. San Francisco, CA: Morgan Kaufman.

Cook, D. J., & Holder, L. B. (2000). Graph-based data mining. *IEEE Intelligent Systems, 15*(2), 32–41. ISSN 1094-7167. doi: http://dx.doi.org/10.1109/5254.850825.

Feldman, R. (2002). Link analysis: Current state of the art. In *Proceedings of the KDD '02*, Edmonton, Alberta, Canada.

Jensen, D. (1999). Statistical challenges to inductive inference in linked data. In *Seventh international workshop on artificial intelligence and statistics*, Fort Lauderdale, FL. San Francisco, CA: Morgan Kaufmann.

Jensen, D., & Goldberg, H. (1998). *AAAI fall symposium on AI and link analysis*, Orlando, FL. Menlo Park, CA: AAAI Press.

Raedt, L. D., (Ed.). (2008). *Logical and relational learning*. Berlin: Springer.
Singh, L., Getoor, L., & Licamele, L. (2005). Pruning social networks using structural properties and descriptive attributes. In *International conference on data mining, 2005*. Houston, TX: IEEE Computer Society.
Wasserman, S., & Faust, K. (1994). *Social network analysis: Methods and applications*. Cambridge: Cambridge University Press.

Link Prediction

Galileo Namata, Lise Getoor
University of Maryland, College Park, Maryland, USA

Synonyms

Edge prediction; Relationship extraction

Definition

Many datasets can naturally be represented as graph where nodes represent instances and links represent relationships between those instances. A fundamental problem with these types of data is that the link information in the graph maybe of dubious quality; links may incorrectly exist between unrelated nodes and links may be missing between two related nodes. The goal of link prediction is to predict the existence of incorrect or missing links between the nodes of the graph.

Theory/Solution

Inferring the existences of edges between nodes in a graph has traditionally been referred to as *link prediction* (Liben-Nowell & Kleinberg, 2003; Taskar, Wong, Abbeel, & Koller, 2003). Link prediction is a challenging problem that has been studied in various guises in different domains. For example, in social network analysis, there is work on predicting friendship links (Zheleva, Getoor, Golbeck, & Kuter, 2008), event participation links (i.e., coauthorship (O'Madadhain, Hutchins, & Smyth, 2005)), communication links (i.e., email (O'Madadhain et al., 2005)), and links representing semantic relationships (i.e., advisor-of (Taskar et al., 2003), subordinate-manager (Diehl, Namata, & Getoor, 2007)). In bioinformatics, there is interest in predicting the existence of edges representing physical protein–protein interactions (Yu, Paccanaro, Trifonov, & Gerstein, 2006; Szilagyi et al., 2005), domain–domain interactions (Deng, Mehta, Sun, & Chen, 2002), and regulatory interactions (Albert et al., 2007). Similarly, in computer network systems there is work in inferring unobserved connections between routers, as well as inferring relationships between autonomous systems and service providers (Spring, Wetherall, & Anderson, 2004). There is also work on using link prediction to improve recommender systems (Farrell, Campbell, & Myagmar, 2005), Web site navigation (Zhu, 2003), surveillance (Huang & Lin, 2008), and automatic document cross referencing (Milne & Witten, 2008).

We begin with some basic definitions and notation. We refer to the set of possible edges in a graph as *potential edges*. The set of potential edges depends on the graph type, and how the edges for the graph are defined. For example, in a directed graph, the set of potential edges consists of all edges $e = (v_1, v_2)$ where v_1 and v_2 are any two nodes V in the graph (i.e., the number of potential edges is $|V| \times |V|$). In an undirected bipartite graph with two subsets of nodes $(V_1, V_2 \in V)$, while the edges still consist of a pair of nodes, $e = (v_1, v_2)$, there is an added condition such that one node must be from V_1 and the other node must be from V_2; this results in $|V_1| \times |V_2|$ potential edges. Next, we refer to set of "true" edges in a graph as *positive edges*, and we refer to the "true" non-edges in a graph (i.e., pairs of nodes without edges between them) as *negative edges*. For a given graph, typically we only have information about a subset of the edges; we refer to this set as the *observed* edges. The observed edges can include both positive and negative edges, though in many formulations there is an assumption of positive-only information. We can view link prediction as a probabilistic inference problem, where the evidence includes the observed edges, the attribute values of the nodes involved in the potential edge, and possibly other information about the network, and for any unobserved, potential edge, we want to compute the probability of it existing. This can be reframed as a binary classification problem by choosing some probability threshold, and concluding that potential edges with existence probability above the threshold are true edges, and those below the threshold are considered false edges (more complex schemes are possible as well). For noisy and incomplete networks, we use terminology from the machine learning literature and refer to an edge that is inferred to exists

and is a true edge in the graph as a *true positive edge*, an edge that should exist but is not inferred as a *false negative edge*, an edge that should not exist and is not inferred as a *true negative edge*, and an edge that should not exist but is incorrectly inferred to exist as a *false positive edge*.

One of the early and simple formulations of the link prediction problem was proposed by Liben-Nowell and Kleinberg (2003). They proposed a temporal prediction problem defined over a dynamic network where given a graph $G_t(V_t, E_t)$ at time t, the problem is to infer the set of edges at the next time step $t + 1$. More formally, the objective is to infer a set of edges E_{new} where $E_{t+1} = E_t \bigcup E_{new}$. We use a more general definition of link prediction proposed by Taskar et al. (2003) where given a graph G and the set of potential edges in G, denoted $P(G)$, the problem of link prediction is to predict for all $p \in P(G)$ whether p exists or does not exists, remaining agnostic on whether G is a noisy graph with missing edges or a snapshot of a dynamic graph at a particular time point.

Approaches

In this section, we discuss the two general categories of the current link prediction models: topology-based approaches and node attribute-based approaches. Topology-based approaches are methods that rely solely on the topology of the network to infer edges. Node attribute-based approaches make predictions based on the attribute values of the nodes incident to the edges. In addition, there are models that make use of both structure and attribute values.

Topology-Based Approaches

A number of link prediction models have been proposed, which rely solely on the topology of the network. These models typically rely on some notion of structural proximity, where nodes that are close are likely to share an edge (e.g., sharing common neighbors, nodes with a small shortest path distance between). The earliest topological approach for link prediction was proposed by Liben-Nowell and Kleinberg (2003). In this work, Liben-Nowell and Kleinberg proposed various structure based similarity scores and applied them over the unobserved edges of an undirected graph. They then use a threshold k, and only predict edges with the top k scores as existing. A variety of similarity scores were proposed, given two nodes v_1 and v_2, including graph distance (the length of the shortest path between v_1 and v_2), common neighbors (the size of the intersection of the sets of neighbors of v_1 and v_2), and more complex measures such as the Katz measure, (the sum of the lengths of the paths between v_1 and v_2 exponentially damped by length to count short paths more heavily). Evaluating over a coauthorship network, the best performing proximity score measure was the Katz measure, however the simple measures, which rely only on the intersection of the set of nodes adjacent to both nodes, performed surprisingly well. A related approach was proposed by Yu et al. (2006), which applies the link prediction problem to predicting missing protein–protein interactions (PPI) from PPI networks generated by high throughput methods. This work assumes that interacting proteins tend to form a clique. Thus, missing edges can be predicted by predicting the existence of edges that will create cliques in the network. More recent work by Clauset, Moore, and Newman (2008) has tried to go beyond predicting edges between neighboring nodes. In their problem domain of food webs, for example, pairs of predators often prey on a shared prey species but rarely prey on each other. Thus, in these networks, predicting "predator–prey" edges need to go beyond proximity. For this, they propose a "hierarchical random graph" approach, which fits a hierarchical model to all possible dendrograms of a given network. The model is then used to calculate the likelihood of an edge existing in the network.

Node Attribute-Based Approaches

Although topology is useful in link prediction, topology-based approaches ignore an important source of information in networks, the attributes of nodes. Often there are correlations in the attributes of nodes that share an edge with each other. One approach that exploits this correlation was proposed by Taskar et al. (2003). In their approach, Taskar et al. (2003) applied the relational Markov network (RMN) framework to link prediction to predicting the existence and class of edges between Web sites. They exploit the fact that certain links can only exist between nodes of the appropriate type. For example, an "advisor" edge can only exist between student and faculty.

Another approach that uses node attributes was proposed by Popescul and Ungar (2003). In that approach, they used a structured ▶logistic regression model over learned relational features to predict citation edges in a citation network. Their relational features are built over attributes such as the words used in the paper nodes. O'Madadhain et al. (2005) also approached an attribute based approach, constructing local conditional probability models based on the attributes such as node attribute similarity, topic distribution, and geographical location in predicting "co-participation" edges in an email communication network. More recently, there is work on exploiting other node attributes like the group membership of the nodes. Zheleva et al. (2008) showed that membership in family groups are very useful in predicting friendship links in social networks. Similarly, Sprinzak, Altuvia, & Margalit (2006) showed that using protein complex information can be useful in predicting protein–protein interactions. Finally, we note that in link prediction, as in classification, the quality of predictions can be improved by making the predictions collectively. Aside from the relational Markov network approach by Taskar et al. (2003) mentioned earlier, Markov logic networks (Richardson & Domingos, 2006) and probabilistic relational models (Getoor, Friedman, Koller, & Taskar, 2003) have also been proposed for link prediction and are capable of performing joint inference.

Issues

There are a number of challenges that make link prediction very difficult. The most difficult challenge is the large class skew between the number of edges that exist and the number of edges that do not. To illustrate, consider directed graph denoted by $G(V, E)$. While the number of edges $|E|$ is often $O(|V|)$, the number of edges that do not exist is often $O(|V|^2)$. Consequently, the prior probability edge existence is very small. This causes many supervised models, which naively optimize for accuracy, to learn a trivial model, which always predicts that a link does not exist. A related problem in link prediction is the large number of edges whose existence must be considered. The number of potential edges is $O(|V|^2)$ and this limits the size of the data sets that can be considered.

In practice, there are general approaches to addressing these issues either prior to or during the link prediction. With both large class skew and number of edges to contend with, the general approach is to make assumptions that reduce the number of edges to consider. One common way to do this is to partition the set of nodes where we only consider potential edges between nodes of the same partition; edges between partitions are not explicitly modeled, but are assumed not to exist. This is useful in many domains where there is some sort of natural partition among the nodes available (e.g., geography in social networks, location of proteins in a cell), which make edges across partitions unlikely. Another way is to define some simple, computationally inexpensive distance measure such that only edges whose nodes are within some distance are considered.

Another practical issue in link prediction is that while real-world data often indicates which edges exist (positive examples), the edges which do not exist (negative examples) are rarely annotated for use by link prediction models. In bioinformatics, for example, the protein–protein interaction network of yeast, the most and annotated studied organism, is annotated with thousands of observed edges (physical interactions) between the nodes (proteins) gathered from numerous experiments. There are currently, however, no major datasets available that indicate which proteins definitely do not physically interact. This is an issue not only in creating and learning models for link prediction, but is also an issue evaluating them. Often, it is unclear whether a predicted edge which is not in our ground truth data is an incorrectly predicted edge or an edge resulting from incomplete data.

Related Problems

In addition to the definition of link prediction discussed above, it is also important to mention three closely related problems: *link completion*, *leak detection*, and *anomalous link discovery*, whose objectives are different but very similar to link prediction. Link completion (Chaiwanarom & Lursinsap, 2008; Goldenberg, Kubica, Komarek, Moore, & Schneider, 2003) and leak detection (Balasubramanyan, Carvalho, & Cohen, 2009; Carvalho & Cohen, 2007), are a variation of link prediction over hypergraphs. A hypergraph is a graph where the edges (known as hyperedges) can connect any number

of nodes. For example, in a hypergraph representing an email communication network, a hyperedge may connect nodes representing email addresses that are recipients of a particular email communication. In link completion, given the set of nodes that participate in a particular hyperedge, the objective is to infer nodes that are missing. For the email communication network example, link completion may involve inferring which email addresses need to be added to the recipients list of an email communication. Conversely, in leak detection, given the set of nodes participating in a particular hyperedge, the objective is to infer which nodes should not be part of that hyperedge. For example, in email communications, leak detection will attempt to infer which email address nodes are incorrectly part of the hyperedge representing the recipient list of the email communication.

The last problem, anomalous link discovery (Huang & Zeng, 2006; Rattigan & Jensen, 2005), has been proposed as an alternate task to link prediction. As with link completion, the existence of the edges are assumed to be observed, and the objective is to infer which of the observed links are anomalous or unusual. Specifically, anomalous link discovery identifies which links are statistically improbable with the idea that these may be of interest for those analyzing the network. Rattigan and Jensen (2005) show that some methods that perform poorly for link prediction can still perform well for anomalous link discovery.

Cross References

►Graph Mining
►Statistical Relational Learning

Recommended Reading

Albert, R., DasGupta, B., Dondi, R., Kachalo, S., Sontag, E., Zelikovsky, A., et al. (2007). A novel method for signal transduction network inference from indirect experimental evidence. *Journal of Computational Biology, 14*, 407–419.

Balasubramanyan, R., Carvalho, V. R., & Cohen, W. (2009). Cutonce recipient recommendation and leak detection in action. In *Workshop on enhanced messaging*.

Carvalho, V. R., & Cohen, W. W. (2007). Preventing information leaks in email. In *SIAM conference on data mining*.

Chaiwanarom, P., & Lursinsap, C. (2008). Link completion using prediction by partial matching. In *International symposium on communications and information technologies*.

Clauset, A., Moore, C., & Newman, M. E. J. (2008). Hierarchical structure and the prediction of missing links in networks. *Nature, 453*, 98.

Deng, M., Mehta, S., Sun, F., & Chen, T. (2002). Inferring domain-domain interactions from protein-protein interactions. *Genome Research, 12*(10), 1540–1548.

Diehl, C., Namata, G. M., & Getoor, L. (2007). Relationship identification for social network discovery. In *Proceedings of the 22nd national conference on artificial intelligence*.

Farrell, S., Campbell, C., & Myagmar, S. (2005). Relescope: An experiment in accelerating relationships. In *Extended abstracts on human factors in computing systems*.

Getoor, L., Friedman, N., Koller, D., & Taskar, B. (2003). Learning probabilistic models of link structure. *Machine Learning, 3*, 679–707.

Goldenberg, A., Kubica, J., Komarek, P., Moore, A., & Schneider, J. (2003). A comparison of statistical and machine learning algorithms on the task of link completion. In *Conference on knowledge discovery and data mining, Workshop on link analysis for detecting complex behavior*.

Huang, Z., & Lin, D. K. J. (2008). The time-series link prediction problem with applications in communication surveillance. *Informs Journal on Computing, 21*, 286–303.

Huang, Z., & Zeng, D. D. (2006). A link prediction approach to anomalous email detection. In *IEEE International conference on systems, man, and cybernetics*, Taipei, Taiwan.

Liben-Nowell, D., & Kleinberg, J. (2003). The link prediction problem for social networks. In *International conference on information and knowledge management*.

Milne, D., & Witten, I. H. (2008). Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on information and knowledge management*.

O'Madadhain, J., Hutchins, J., & Smyth, P. (2005). Prediction and ranking algorithms for event-based network data. *SIGKDD Explorations Newsletter, 7*(2), 23–30.

Popescul, A., & Ungar, L. H. (2003). Statistical relational learning for link prediction. In *International joint conferences on artificial intelligence workshop on learning statistical models from relational data*.

Rattigan, M. J., & Jensen, D. (2005). The case for anomalous link discovery. *SIGKDD Explorations Newsletter, 7*, 41–47.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning, 62*, 107–136.

Spring, N., Wetherall, D., & Anderson, T. (2004). Reverse engineering the internet. *SIGCOMM Computer Communication Review, 34*(1), 3–8.

Sprinzak, E., Altuvia, Y., & Margalit, H. (2006). Characterization and prediction of protein-protein interactions within and between complexes. *Proceedings of the National Academy of Sciences, 103*(40), 14718–14723.

Szilagyi, A., Grimm, V., Arakaki, A. K., & Skolnick, J. (2005). Prediction of physical protein-protein interactions. *Physical Biology, 2*(2), S1–S16.

Taskar, B., Wong, M.-F., Abbeel, P., & Koller, D. (2003). Link prediction in relational data. In *Advances in neural information processing systems*.

Yu, H., Paccanaro, A., Trifonov, V., & Gerstein, M. (2006). Predicting interactions in protein networks by completing defective cliques. *Bioinformatics, 22*(7), 823–829.

Zheleva, E., Getoor, L., Golbeck, J., & Kuter, U. (2008). Using friendship ties and family circles for link prediction. In *2nd ACM SIGKDD workshop on social network mining and analysis*.

Zhu, J. (2003). *Mining web site link structure for adaptive web site navigation and search*. Ph.D. thesis, University of Ulster at Jordanstown, UK.

Link-Based Classification

►Collective Classification

Liquid State Machine

►Reservoir Computing

Local Distance Metric Adaptation

Synonyms

Supersmoothing; Nonstationary kernels; Kernel shaping

Definition

In learning systems with kernels, the shape and size of a kernel plays a critical role for accuracy and generalization. Most kernels have a distance metric parameter, which determines the size and shape of the kernel in the sense of a Mahalanobis distance. Advanced kernel learning tune every kernel's distance metric individually, instead of turning one global distance metric for all kernels.

Cross References

►Locally Weighted Regression for Control

Local Feature Selection

►Projective Clustering

Locality Sensitive Hashing Based Clustering

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

The basic idea of the LSH (Gionis, Indyk, & Motwani, 1999) technique is using multiple hash functions to hash the data points and guarantee that there is a high probability of collision for points which are close to each other and low collision probability for dissimilar points. LSH schemes exist for many distance measures, such as Hamming norm, L_p norms, cosine distance, earth movers distance (EMD), and Jaccard coefficient.

In LSH, define a family $H = \{h : S \to U\}$ as locality-sensitive, if for any a, the function $p(t) = Pr_H[h(a) = h(b) : ||a - b|| = x]$ is decreasing in x. Based on this definition, the probability of collision of points a and b is decreasing with their distance.

Although LSH was originally proposed for approximate nearest neighbor search in high dimensions, it can be used for clustering as well (Das, Datar, Garg, & Rajaram, 2007; Haveliwala, Gionis, & Indyk, 2000). The buckets could be used as the bases for clustering. Seeding the hash functions several times can help getting better quality clustering.

Recommended Reading

Das, A. S., Datar, M., Garg, A., & Rajaram, S. (2007). Google news personalization: Scalable online collaborative filtering. In *WWW '07: Proceedings of the 16th international conference on World Wide Web* (pp. 271–280). New York: ACM.

Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing. In *VLDB '99: Proceedings of the 25th international conference on very large data bases* (pp. 518–529). San Francisco: Morgan Kaufmann Publishers.

Haveliwala, T. H., Gionis, A., & Indyk, P. (2000). Scalable techniques for clustering the web (extended abstract). In *Proceedings of the third international workshop on the web and databases* (pp. 129–134). Stanford, CA: Stanford University.

Locally Weighted Learning

►Locally Weighted Regression for Control

Locally Weighted Regression for Control

Jo-Anne Ting[1], Sethu Vijayakumar[1,2], Stefan Schaal[2,3]
[1]University of Edinburgh
[2]University of Southern California
[3]ATR Computational Neuroscience Labs

Synonyms

Kernel shaping; Lazy learning; Local distance metric adaptation; Locally weighted learning; LWPR; LWR; Nonstationary kernels supersmoothing

Definition

This article addresses two topics: ►learning control and locally weighted regression.

►Learning control refers to the process of acquiring a control strategy for a particular control system and a particular task by trial and error. It is usually distinguished from adaptive control (Aström & Wittenmark, 1989) in that the learning system is permitted to fail during the process of learning, resembling how humans and animals acquire new movement strategies. In contrast, adaptive control emphasizes single trial convergence without failure, fulfilling stringent performance constraints, e.g., as needed in life-critical systems like airplanes and industrial robots.

Locally weighted regression refers to ►supervised learning of continuous functions (otherwise known as function approximation or ►regression) by means of spatially localized algorithms, which are often discussed in the context of ►kernel regression, ►nearest neighbor methods, or ►lazy learning (Atkeson, Moore, & Schaal, 1997). Most regression algorithms are global learning systems. For instance, many algorithms can be understood in terms of minimizing a global ►loss function such as the expected sum squared error:

$$J_{\text{global}} = E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \mathbf{y}_i\right)^2\right] = E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \phi\left(\mathbf{x}_i\right)^T\boldsymbol{\beta}\right)^2\right] \tag{1}$$

where $E[\cdot]$ denotes the expectation operator, $\mathbf{t}_i$ the noise-corrupted target value for an input $\mathbf{x}_i$, which is expanded by basis functions into a basis function vector $\phi(\mathbf{x}_i)$, and $\boldsymbol{\beta}$ the vector of (usually linear) regression coefficients. Classical feedforward ►neural networks, ►radial basis function networks, ►mixture models, or ►Gaussian Process regression are all global function approximators in the spirit of Eq. (1).

In contrast, local learning systems split up conceptually the cost function into multiple independent local function approximation problems, using a cost function such as the one below:

$$\begin{aligned} J_{\text{global}} &= E\left[\frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{N} w_{k,i}\left(\mathbf{t}_i - \mathbf{x}_i^T\boldsymbol{\beta}_k\right)^2\right] \\ &= \frac{1}{2}\sum_{k=1}^{K} E\left[\sum_{i=1}^{N} w_{k,i}\left(\mathbf{t}_i - \mathbf{x}_i^T\boldsymbol{\beta}_k\right)^2\right] \end{aligned} \tag{2}$$

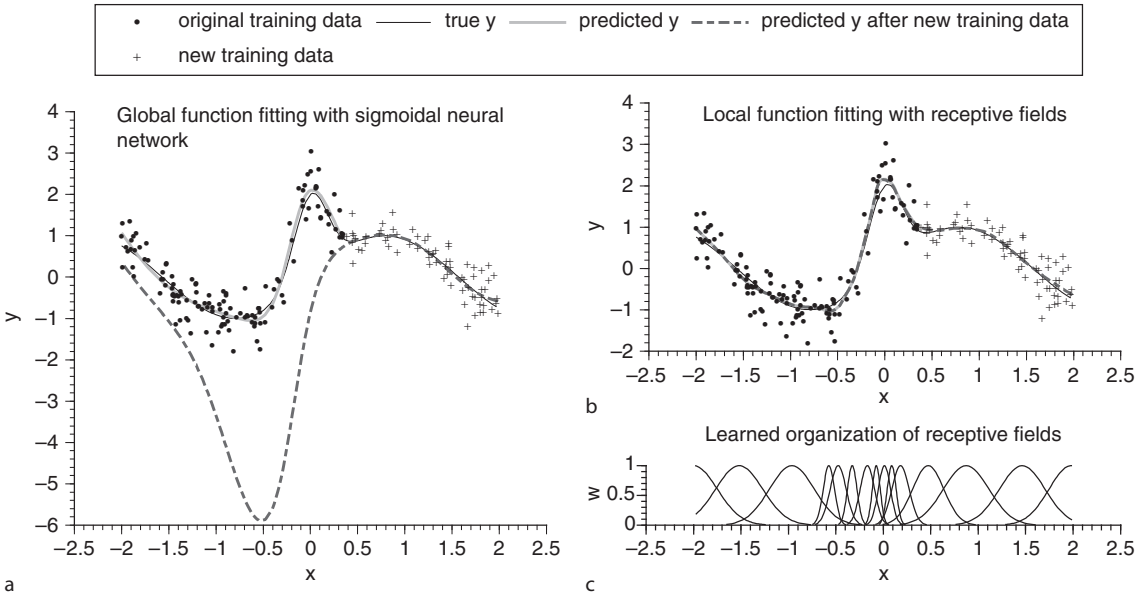
Motivation and Background

Figure 1 illustrates why locally weighted regression methods are often favored over global methods when it comes to learning from incrementally arriving data, especially when dealing with nonstationary input distributions. The figure shows the division of the training data into two sets: the "original training data" and the "new training data" (in dots and crosses, respectively).

Initially, a sigmoidal ►neural network and a locally weighted regression algorithm are trained on the "original training data," using 20% of the data as a cross-validation set to assess convergence of the learning. In a second phase, both learning systems are trained solely on the "new training data" (again with a similar cross-validation procedure), but without using any data from the "original training data." While both algorithms generalize well on the "new training data," the global learner incurred catastrophic interference, unlearning what was learned initially, as seen in Fig. 1a, b shows that the locally weighted regression algorithm does not have this problem since learning (along with ►generalization) is restricted to a local area.

Appealing properties of locally weighted regression include the following:

- Function approximation can be performed incrementally with nonstationary input and output distributions and without significant danger of interference. Locally weighted regression can provide ►posterior probability distributions, offer confidence assessments, and deal with heteroscedastic data.
- Locally weighted learning algorithms are computationally inexpensive to compute. It is well suited for online computations (e.g., for ►online and ►incremental learning) in the fast control loop of a robot – typically on the order of 100–1000 Hz.
- Locally weighted regression methods can implement continual learning and learning from large amounts of data without running into severe computational problems on modern computing hardware.
- Locally weighted regression is a nonparametric method (i.e., it does not require that the user determine *a priori* the number of local models in the learning system), and the learning systems grows with the complexity of the data it tries to model.
- Locally weighted regression can include ►feature selection, ►dimensionality reduction, and ►Bayesian inference – all which are required for robust statistical inference.



Locally Weighted Regression for Control. Figure 1. Function approximation results for the function $y = \sin(2x) + 2\exp(-16x^2) + N(0, 0.16)$ with (a) a sigmoidal neural network; (b) a locally weighted regression algorithm (note that the data traces "true y," "predicted y," and "predicted y after new training data" largely coincide); and (c) the organization of the (Gaussian) kernels of (b) after training. See Schaal and Atkeson (1998) for more details

- Locally weighted regression works favorably with locally linear models (Hastie & Loader, 1993), and local linearizations are of ubiquitous use in control applications.

Background

Returning to Eqs. (1) and (2), the main differences between both equations are listed below:

(i) A weight $w_{i,k}$ is introduced that focuses the function approximation on only a small neighborhood around a point of interest $\mathbf{c}_k$ in input space (see Eq. 3 below).
(ii) The cost function is split into K independent optimization problems.
(iii) Due to the restricted scope of the function approximation problem, we do not need a nonlinear basis function expansion and can, instead, work with simple local functions or local polynomials (Hastie & Loader, 1993).

The weights $w_{k,i}$ in Eq. (2) are typically computed from some ▶kernel function (Atkeson, Moore, & Schaal, 1997) such as a squared exponential kernel

$$w_{k,i} = \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{c}_k)^T \mathbf{D}_k (\mathbf{x}_i - \mathbf{c}_k)\right) \quad (3)$$

with $\mathbf{D}_k$ denoting a positive semidefinite distance metric and $\mathbf{c}_k$ the center of the kernel. The number of kernels K is not finite. In many local learning algorithms, the kernels are never maintained in memory. Instead, for every query point $\mathbf{x}_q$, a new kernel is centered at $\mathbf{c}_k = \mathbf{x}_q$, and the localized function approximation is solved with weighted ▶regression techniques (Atkeson et al., 1997).

Locally weighted regression should not be confused with mixture of experts models (Jordan & Jacobs, 1994). ▶Mixture models are *global* learning systems since the experts compete globally to cover training data. Mixture models address the ▶bias-variance dilemma (Intuitively, the ▶bias-variance dilemma addresses how many parameters to use for a function approximation problem to find an optimal balance between ▶overfitting and oversmoothing of the training data) by finding the right number of local experts. Locally weighted regression addresses the ▶bias-variance dilemma in a local way by finding the

optimal distance metric for computing the weights in the locally weighted regression (Schaal & Atkeson, 1998). We describe some algorithms to find $\mathbf{D}_k$ next.

Structure of Learning System

For a locally linear model centered at the query point $\mathbf{x}_q$, the regression coefficients would be

$$\boldsymbol{\beta}_q = \left(\mathbf{X}^T\mathbf{W}_q\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}_q\mathbf{t} \tag{4}$$

where $\mathbf{X}$ is a matrix that has all training input data points in its rows (with a column of 1s added in the last column for the offset parameter in ►linear regression). $\mathbf{W}_q$ is a diagonal matrix with the corresponding weights for all data points, computed from Eq. (3) with $\mathbf{c}_k = \mathbf{x}_q$, and $\mathbf{t}$ is the vector of regression targets for all training points. Such a "compute-the-prediction-on-the-fly" approach is often called lazy learning (The approach is "lazy" because the computational of a prediction is deferred until the last moment, i.e., when a prediction is needed) and is a memory-based learning system where all training data is kept in memory for making predictions.

Alternatively, kernels can be created as needed to cover the input space, and the sufficient statistics of the weighted regression are updated incrementally with recursive least squares (Schaal & Atkeson, 1998). This approach does not require storage of data points in memory. Predictions of neighboring local models can be blended, improving function fitting results in the spirit of committee machines.

Memory-Based Locally Weighted Regression (LWR)

The original locally weighted regression algorithm was introduced by Cleveland (1979) and popularized in the machine learning and learning control community by Atkeson (1989). The algorithm is largely summarized by Eq. (4) (for algorithmic pseudo-code, see (Schaal, Atkeson, & Vijayakumar, 2002)):

- All training data is collected in the matrix $\mathbf{X}$ and the vector $\mathbf{t}$ (For simplicity, only functions with a scalar output are addressed. Vector-valued outputs can be learned either by fitting a separate learning system for each output or by modifying the algorithms to fit multiple outputs (similar to multi-output linear regression)).
- For every query point $\mathbf{x}_q$, the weighting kernel is centered at the query point.
- The weights are computed with Eq. (3).
- The local regression coefficients are computed according to Eq. (4).
- A prediction is formed with $y_q = \left[\mathbf{x}_q^T\ 1\right]\boldsymbol{\beta}_q$.

As in all kernel methods, it is important to optimize the kernel parameters in order to get optimal function fitting quality. For LWR, the critical parameter determining the ►bias-variance tradeoff is the distance metric $\mathbf{D}_q$. If the kernel is too narrow, it starts fitting noise. If it is too broad, oversmoothing will occur. $\mathbf{D}_q$ can be optimized with leave-one-out cross-validation to obtain a *globally* optimal value, i.e., the same $\mathbf{D}_q = \mathbf{D}$ is used throughout the entire input space of the data. Alternatively, $\mathbf{D}_q$ can be *locally* optimized as a function of the query point, i.e., obtain a $\mathbf{D}_q$ as a function of the query point (as already indicated by the subscript "q"). In the recent machine learning literature (in particular, work related to kernel methods), such input dependent kernels are referred to as nonstationary kernels.

Locally Weighted Projection Regression (LWPR)

Schaal and Atkeson (1998) suggested a memoryless version of LWR in order to avoid the expensive ►nearest neighbor computations – particularly for large training data sets – of LWR and to have fast real-time (In most robotic systems, "real-time" means on the order of maximally 1–10 ms computation time, corresponding to a 1000–100 Hz control loop) prediction performance. The main ideas of the RFWR algorithm (Schaal & Atkeson, 1998) are listed below:

- Create new kernels only if no existing kernel in memory covers a training point with some minimal activation weight.
- Keep all created kernels in memory and update the weighted regression with weighted recursive least squares for new training points $\{\mathbf{x}, t\}$:

$$\begin{aligned}\boldsymbol{\beta}_k^{n+1} &= \boldsymbol{\beta}_k^n + w\mathbf{P}^{n+1}\tilde{\mathbf{x}}\left(t - \tilde{\mathbf{x}}^T\boldsymbol{\beta}_k^n\right)\\ \text{where } \mathbf{P}_k^{n+1} &= \frac{1}{\lambda}\left(\mathbf{P}_k^n - \frac{\mathbf{P}_k^n\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\mathbf{P}_k^n}{\frac{\lambda}{w} + \tilde{\mathbf{x}}^T\mathbf{P}_k^n\tilde{\mathbf{x}}}\right)\\ \text{and } \tilde{\mathbf{x}} &= \left[\mathbf{x}^T\ 1\right]^T.\end{aligned} \tag{5}$$

- Adjust the distance metric $\mathbf{D}_q$ for each kernel with a gradient descent technique using leave-one-out cross-validation.
- Make a prediction for a query point taking a weighted average of predictions from all local models:

$$\mathbf{y}_q = \frac{\sum_{k=1}^{K} w_{q,k} \hat{\mathbf{y}}_{q,k}}{\sum_{k=1}^{K} w_{q,k}} \tag{6}$$

Adjusting the distance metric $\mathbf{D}_q$ with leave-one-out cross-validation *without* keeping all training data in memory is possible due to the PRESS residual. The PRESS residual allows the leave-one-out cross-validation error to be computed in closed form without needing to actually exclude a data point from the training data.

Another deficiency of LWR is its inability to scale well to high-dimensional input spaces since the ►covariance matrix inversion in Eq. (4) becomes severely ill-conditioned. Additionally, LWR becomes expensive to evaluate as the number of local models to be maintained increases. Vijayakumar, D'Souza and Schaal (2005) suggested local ►dimensionality reduction techniques to handle this problem. Partial least squares (PLS) regression is a useful ►dimensionality reduction method that is used in the LWPR algorithm (Vijayakumar et al., 2005). In contrast to PCA methods, PLS performs ►dimensionality reduction for ►regression, i.e., it eliminates subspaces of the input space that minimally correlate with the outputs, not just parts of the input space that have low variance.

LWPR is currently one of the best developed locally weighted regression algorithms for control (Klanke, Vijayakumar, & Schaal, 2008) and has been applied to learning control problems with over 100 input dimensions.

A Full Bayesian Treatment of Locally Weighted Regression

Ting, Kalakrishnan, Vijayakumar, and Schaal (2008) proposed a fully probabilistic treatment of LWR in an attempt to avoid cross-validation procedures and minimize any manual parameter tuning (e.g., gradient descent rates, kernel initialization, and forgetting rates). The resulting Bayesian algorithm learns the distance metric of local linear model (For simplicity, a local linear model is assumed, although local polynomials can be used as well) probabilistically, can cope with high input dimensions, and rejects data outliers automatically. The main ideas of Bayesian LWR are listed below (please see Ting (2009) for details):

- Introduce hidden variables $\mathbf{z}$ to the local linear model (as in Variational Bayesian least squares (Ting et al., 2005)) to decompose the statistical estimation problem into d individual estimation problems (where d is the number of input dimensions). The result is an iterative Expectation-Maximization (EM) algorithm that is of linear ►computational complexity in d and the number of training data samples N, i.e., $O(Nd)$.
- Associate a scalar weight w_i with each training data sample $\{\mathbf{x}_i, t_i\}$, placing a Bernoulli ►prior probability distribution over a weight *for each input dimension* so that the weights are positive and between 0 and 1:

$$w_i = \prod_{m=1}^{d} w_{im} \text{ where}$$
$$w_{im} \sim \text{Bernoulli}\,(q_{im}) \text{ for } i = 1, .., N; m = 1, .., d \tag{7}$$

where the weight w_i is decomposed into independent components in each input dimension w_{im} and q_{im} is the parameter of the Bernoulli ►probability distribution. The weight w_i indicates a training sample's contribution to the local model. An outlier will have a weight of 0 and will, thus, be automatically rejected. The formulation of q_{im} determines the shape of the weighting function applied to the local model. The weighting function q_{im} used in Bayesian LWR is listed below:

$$q_{im} = \frac{1}{1 + \left(x_{im} - x_{qm}\right)^2 h_m} \text{ for } i = 1, .., N; m = 1, .., d \tag{8}$$

where $\mathbf{x}_q \in \Re^{d \times 1}$ is the query input point and h_m is the bandwidth parameter/distance metric of the local model in the m-th input dimension (The distance metric/bandwidth is assumed to be a diagonal matrix, i.e., bandwidths in each input dimension are independent. That is to say, $\mathbf{D} = \mathbf{H}$, where $\mathbf{h}$ is the diagonal vector and h_m are the coefficients of $\mathbf{h}$).

- Place a Gamma ▶prior probability distribution over the distance metric h_m:

$$h_m \sim \text{Gamma}\,(a_{hm0}, b_{hm0}) \tag{9}$$

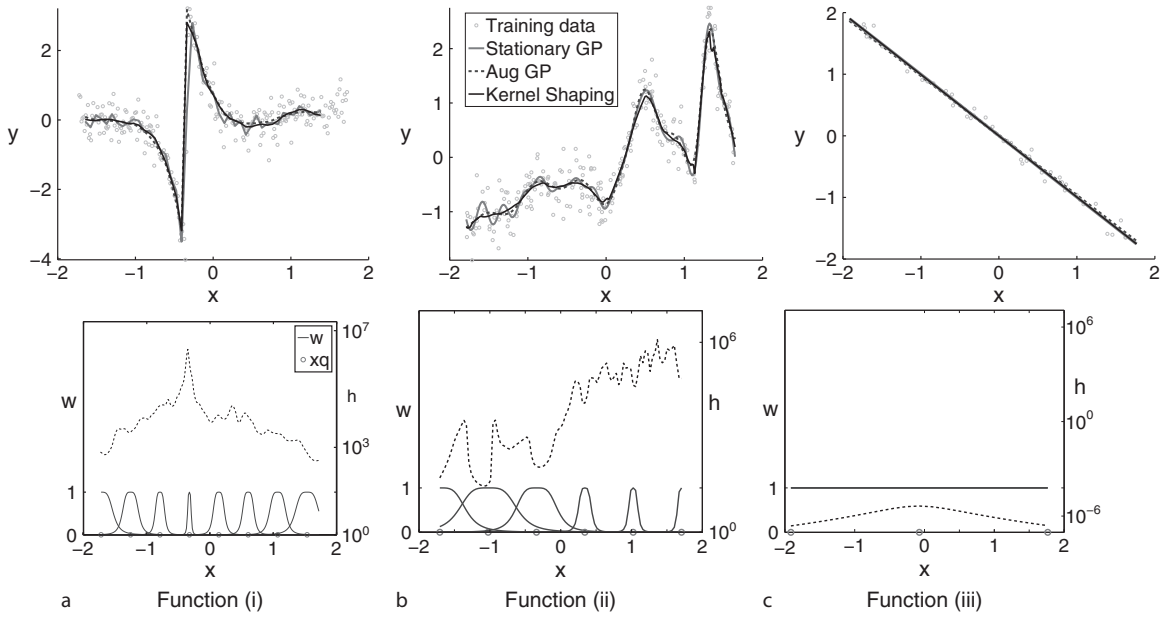
where $\{a_{hm0}, b_{hm0}\}$ are the prior parameter values of the Gamma distribution.
- Treat the model as an EM-like ▶regression problem, using ▶variational approximations to achieve analytically tractable inference of the ▶posterior probability distributions.

The initial parameters $\{a_{hm0}, b_{hm0}\}$ should be set so that the ▶prior probability distribution over h_m is uninformative and wide (e.g., $a_{hm0} = b_{hm0} = 10^{-6}$). The other ▶prior probability distribution that needs to be specified is the one over the noise variance random variable – and this is best set to reflect how noisy the data set is believed to be. More details can be found in Ting (2009).

This Bayesian method can can also be applied as general kernel shaping algorithm for global ▶kernel learning methods that are linear in the parameters (e.g., to realize nonstationary ▶Gaussian processes (Ting et al., 2008), resulting in an augmented nonstationary ▶Gaussian Process).

Figure 2 illustrates Bayesian kernel shaping's bandwidth adaptation abilities on several synthetic data sets, comparing it to a stationary ▶Gaussian Process and the augmented nonstationary ▶Gaussian Process. For the ease of visualization, the following one-dimensional functions are considered: (i) a function with a discontinuity, (ii) a spatially inhomogeneous function, and (iii) a straight line function. The data set for function (i) consists of 250 training samples, 201 test inputs (evenly spaced across the input space), and output noise with variance of 0.3025; the data set for function (ii) consists of 250 training samples, 101 test inputs, and an output signal-to-noise ratio (SNR) of 10; and the data set for function (iii) has 50 training samples, 21 test inputs, and an output SNR of 100. Figure 2 shows the predicted outputs of all three algorithms for data sets (i)–(iii). The local kernel shaping algorithm smoothes over regions where a stationary ▶Gaussian Process overfits and yet, it still manages to capture regions of highly varying curvature, as seen in Figs. 2a and 2b.

It correctly adjusts the bandwidths h with the curvature of the function. When the data looks linear, the algorithm opens up the weighting kernel so that all data samples are considered, as Fig. 2c shows.



Locally Weighted Regression for Control. Figure 2. Predicted outputs using a stationary Gaussian Process (GP), the augmented nonstationary GP and local kernel shaping on three different data sets. Figures on the bottom row show the bandwidths learned by local kernel shaping and the corresponding weighting kernels (*in dotted black lines*) for various input query points (*shown in red circles*)

From the viewpoint of ▶learning control, ▶overfitting – as seen in the ▶Gaussian Process in Fig. 2 – can be detrimental since ▶learning control often relies on extracting local linearizations to derive ▶controllers (see Applications section). Obtaining the wrong sign on a slope in a local linearization may destabilize a ▶controller.

In contrast to LWPR, the Bayesian LWR method is memory-based, although memoryless versions could be derived. Future work will also have to address how to incorporate ▶dimensionality reduction methods for robustness in high dimensions. Nevertheless, it is a first step toward a probabilistic locally weighted regression method with minimal parameter tuning required by the user.

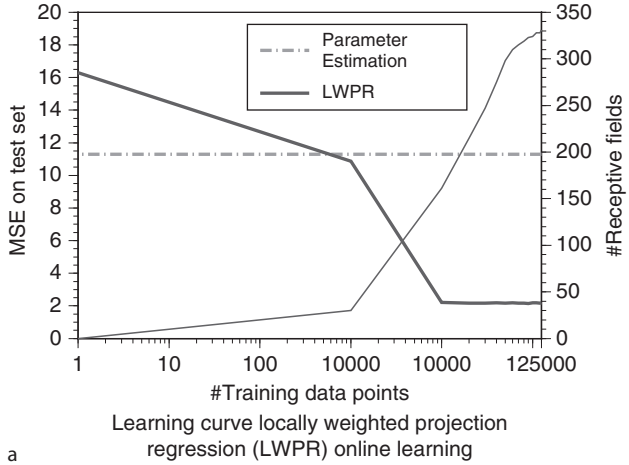
Applications

Learning Internal Models with LWPR

Learning an internal model is one of most typical applications of LWR methods for control. The model could be a forward model (e.g, the nonlinear differential equations of robot dynamics), an inverse model (e.g., the equations that predict the amount of torque to achieve a change of state in a robot), or any other function that models associations between input and output data about the environment. The models are used, subsequently, to compute a ▶controller e.g., an inverse dynamics controller similar to Eq. (12). Models for complex robots such as humanoids exceed easily a hundred input dimensions. In such high-dimensional spaces, it is hopeless to assume that a representative data set can be collected for offline training that can generalize sufficiently to related tasks. Thus, the LWR philosophy involves having a learning algorithm that can learn rapidly when entering a new part of the state space such that it can achieve acceptable ▶generalization performance almost instantaneously.

Figure 3 demonstrates ▶online learning of an inverse dynamics model for the elbow joint (cf. Eq. 12) for a Sarcos Dexterous Robot Arm. The robot starts with no knowledge about this model, and it tracks some randomly varying desired trajectories with a proportional-derivative (PD) controller. During its movements, training data consisting of tuples $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \tau)$ – which model a mapping from joint position, joint velocities and joint accelerations $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ to motor torques τ – are collected (at about every 2 ms). Every data point is used to train a LWPR function approximator, which generates a feedforward command for the controller. The ▶learning curve is shown in Fig. 3a.

Using a test set created beforehand, the model predictions of LWPR are compared every 1,000 training points with that of a parameter estimation method. The parameter estimation approach fits the minimal number of parameters to an analytical model of the robot dynamics under an idealized rigid body dynamics (RBD) assumptions, using all training data (i.e., not incrementally). Given that the Sarcos robot is a



a Learning curve locally weighted projection regression (LWPR) online learning

b Seven Degree-of-Freedom Sarcos Robot Arm

Locally Weighted Regression for Control. Figure 3. Learning an inverse dynamics model in real-time with a high-performance anthropomprohic robot arm

hydraulic robot, the RBD assumption is not very suitable, and, as Fig. 3a shows, LWPR (in thick red line) outperforms the analytical model (in dotted blue line) after a rather short amount of training. After about 5 min of training (about 125,000 data points), very good performance is achieved, using about 350 local models. This example demonstrates (i) the quality of function approximation that can be achieved with LWPR and (ii) the online allocation of more local models as needed.

Learning Paired Inverse-Forward Models

Learning inverse models (such as inverse kinematics and inverse dynamics models) can be challenging since the inverse model problem is often a relation, not a function, with a one-to-many mapping. Applying any arbitrary nonlinear function approximation method to the inverse model problem can lead to unpredictably bad performance, as the training data can form non-convex solution spaces, in which averaging is inappropriate. Architectures such as ►mixture models (in particular, mixture density networks) have been proposed to address problems with non-convex solution spaces. A particularly interesting approach in control involves learning linearizations of a forward model (which is proper function) and learning an inverse mapping within the local region of the forward model.

Ting et al. (2008) demonstrated such a forward-inverse model learning approach with Bayesian LWR to learn an inverse kinematics model for a haptic robot arm (shown in Fig. 4) in order to control the end-effector along a desired trajectory in task space. Training data was collected while the arm performed random sinusoidal movements within a constrained box volume of Cartesian space. Each sample consists of the arm's joint angles $\mathbf{q}$, joint velocities $\dot{\mathbf{q}}$, end-effector position in Cartesian space $\mathbf{x}$, and end-effector velocities $\dot{\mathbf{x}}$. From this data, a forward kinematics model is learned:

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{10}$$

where $\mathbf{J}(\mathbf{q})$ is the Jacobian matrix. The transformation from $\dot{\mathbf{q}}$ to $\dot{\mathbf{x}}$ can be assumed to be locally linear at a particular configuration $\mathbf{q}$ of the robot arm. Bayesian LWR is used to learn the forward model, and, as in LWPR, local models are only added if a training point is not already sufficiently covered by an existing local model. Importantly, the kernel functions in LWR are localized only with respect to $\mathbf{q}$, while the regression of each model is trained only on a mapping from $\dot{\mathbf{q}}$ to $\dot{\mathbf{x}}$ – these geometric insights are easily incorporated as priors in Bayesian LWR, as they are natural to locally linear models. Incorporating these priors in other function approximators, e.g., ►Gaussian Process regression, is not straightforward.

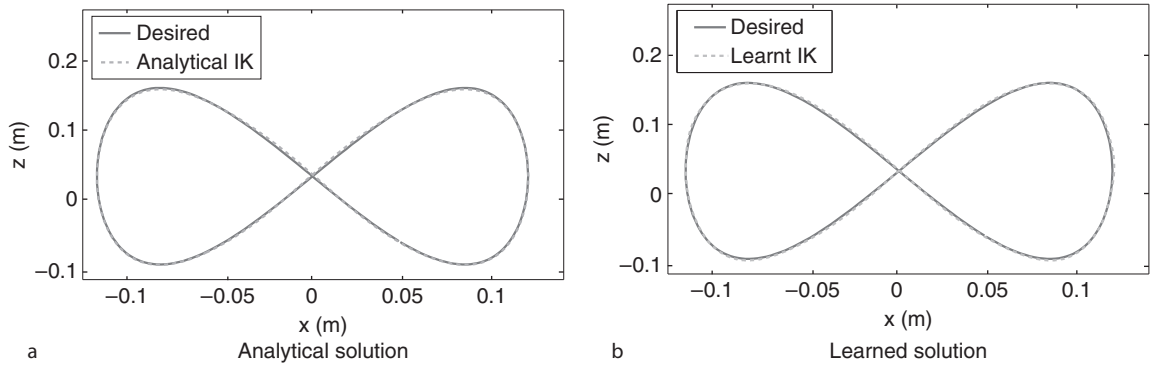
The goal of the robot task is to track a desired trajectory $(\mathbf{x}, \dot{\mathbf{x}})$ specified only in terms of x, z positions and velocities, i.e., the movement is supposed to be in a vertical plane in front of the robot, but the exact position of the vertical plane is not given. Thus, the task has one degree of redundancy, and the learning system needs to generate a mapping from $\{\mathbf{x}, \dot{\mathbf{x}}\}$ to $\dot{\mathbf{q}}$. Analytically, the inverse kinematics equation is

$$\dot{\mathbf{q}} = \mathbf{J}^{\#}(\mathbf{q})\dot{\mathbf{x}} - \alpha(\mathbf{I} - \mathbf{J}^{\#}\mathbf{J})\frac{\partial g}{\partial \mathbf{q}} \tag{11}$$

where $J^{\#}(\mathbf{q})$ is the pseudo-inverse of the Jacobian. The second term is an gradient descent optimization term for redundancy resolution, specified here by a cost function g in terms of joint angles $\mathbf{q}$.

To learn an inverse kinematics model, the local regions of $\mathbf{q}$ from the forward model can be re-used since any inverse of $\mathbf{J}$ is locally linear within these regions. Moreover, for locally linear models, all solution spaces for the inverse model are locally convex, such that an inverse can be learned without problems. The redundancy issue can be solved by applying an additional weight to each data point according to a reward function. Since the experimental task is specified in

Locally Weighted Regression for Control. Figure 4. SensAble Phantom haptic robotic arm



Locally Weighted Regression for Control. Figure 5. Desired versus actual trajectories for SensAble Phantom robot arm

terms of $\{\dot{x}, \dot{z}\}$, a reward is defined, based on a desired y coordinate, y_{des}, and enforced as a soft constraint. The resulting reward function, is $g = e^{-\frac{1}{2}h(k(y_{des}-y)-\dot{y})^2}$, where k is a gain and h specifies the steepness of the reward. This ensures that the learned inverse model chooses a solution that pushes $\dot{y}$ toward y_{des}. Each forward local model is inverted using a weighted ▶linear regression, where each data point is weighted by the kernel weight from the forward model and additionally weighted by the reward. Thus, a piecewise locally linear solution to the inverse problem can be learned efficiently.

Figure 5 shows the performance of the learned inverse model (Learnt IK) in a figure-eight tracking task. The learned model performs as well as the analytical inverse kinematics solution (Analytical IK), with root mean squared tracking errors in positions and velocities very close to that of the analytical solution.

Learning Trajectory Optimizations

Mitrovic, Klanke, and Vijayakumar (2008) have explored a theory for sensorimotor adaptation in humans, i.e., how humans replan their movement trajectories in the presence of perturbations. They rely on the iterative Linear Quadratic Gaussian (iLQG) algorithm (Todorov & Li, 2004) to deal with the nonlinear and changing plant dynamics that may result from altered morphology, wear and tear, or external perturbations. They take advantage of the "on-the-fly" adaptation of locally weighted regression methods like LWPR to learn the forward dynamics of a simulated arm for the purpose of optimizing a movement trajectory between a start point and an end point.

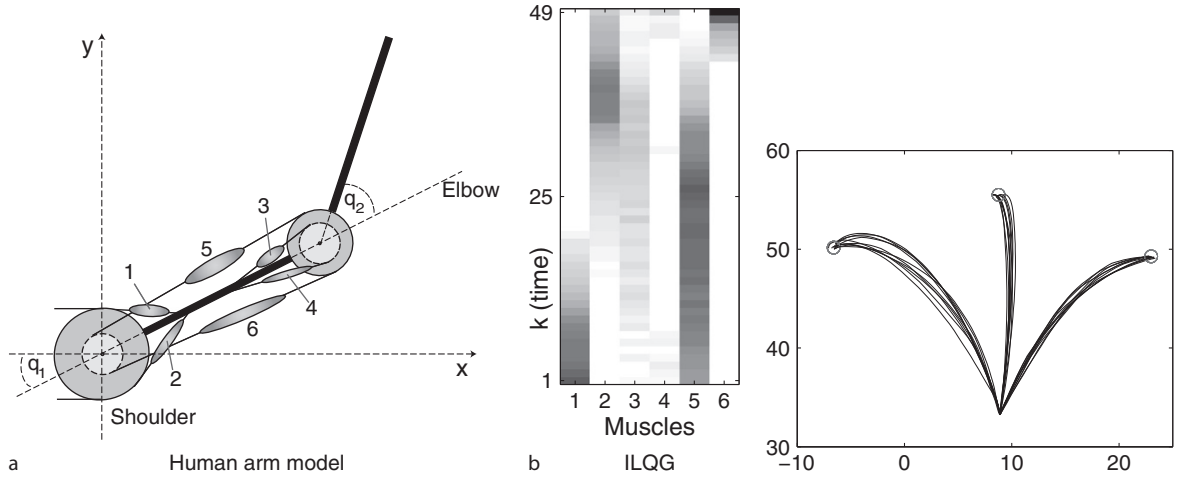
Figure 6a shows the diagram of a two degrees-of-freedom planar human arm model, which is actuated by four single-joint and two double-joint antagonistic muscles. Although kinematically simple, the system is over-actuated and, therefore, it is an interesting testbed because large redundancies in the dynamics have to be resolved. The dimensionality of the control signals makes adaptation processes (e.g., to external force fields) quite demanding.

The dynamics of the arm is, in part, based on standard RBD equations of motion:

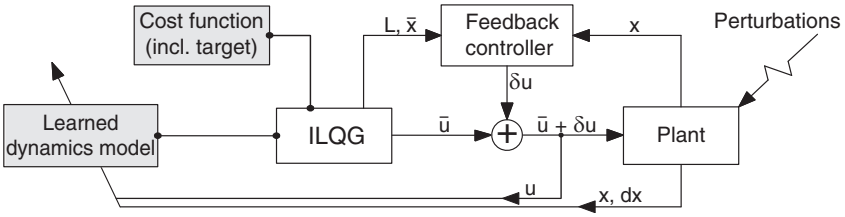
$$\tau = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} \tag{12}$$

where τ are the joint torques; $\mathbf{q}$ and $\dot{\mathbf{q}}$ are the joint angles and velocities, respectively; $\mathbf{M}(\mathbf{q})$ is the two-dimensional symmetric joint space inertia matrix; and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ accounts for Coriolis and centripetal forces. Given the antagonistic muscle-based actuation, it is not possible to command joint torques directly. Instead, the effective torques from the muscle activations $\mathbf{u}$ – which happens to be quadratic in $\mathbf{u}$ – should be used. As a result, in contrast to standard torque-controlled robots, the dynamics equation in Eq. (12) is *nonlinear in the control signals* $\mathbf{u}$.

The iLQG algorithm (Todorov & Li, 2004) is used to calculate solutions to "localized" linear and quadratic approximations, which are iterated to improve the global control solution. However, it relies on an analytical forward dynamics model $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ and finite difference methods to compute gradients. To alleviate this requirement and to make iLQG adaptive, LWPR can be used to learn an approximation of the plant's forward dynamics model. Figure 7 shows the control



Locally Weighted Regression for Control. Figure 6. (a) Human arm model with 6 muscles; (b) Optimized control sequence (*left*) and resulting trajectories (*right*) using the known analytic dynamics model. The control sequences (left target only) for each muscle (1–6) are drawn from bottom to top, with darker grey levels indicating stronger muscle activation

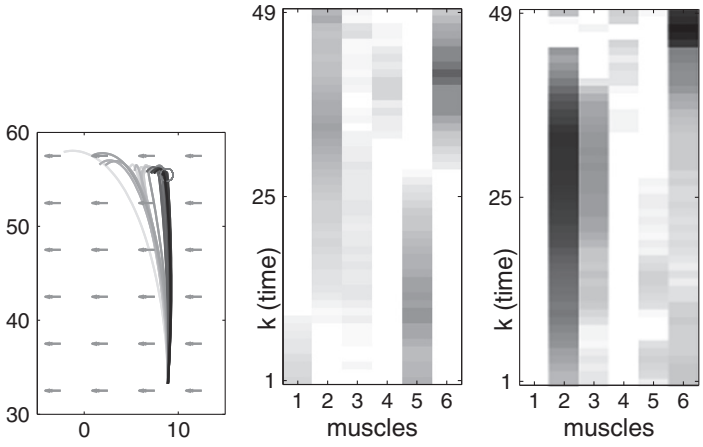


Locally Weighted Regression for Control. Figure 7. Illustration of learning and control scheme of the iterative Linear Quadratic Gaussian (iLQG) algorithm with learned dynamics

diagram, where the "learned dynamics model" (the forward model learned by LWPR) is then updated in an online fashion with every iteration to cope with changes in dynamics. The resulting framework is called iLQG-LD (iLQG with learned dynamics).

Movements of the arm model in Fig. 6a are studied for fixed time horizon reaching movement. The manipulator starts at an initial position $\mathbf{q}_0$ and reaches towards a target $\mathbf{q}_{tar}$. The cost function to be optimized during the movement is a combination of target accuracy and amount of muscle activation (i.e., energy consumption). Figure 6b shows trajectories of generated movements for three reference targets (shown in red circles) using the feedback controller from iLQG with the analytical plant dynamics. The trajectories generated with iLQG-LD (where the forward plant dynamics are learned with LWPR) are omitted as they are hardly distinguishable from the analytical solution.

A major advantage of iLQG-LD is that it does not rely on an accurate analytic dynamics model; this enables the framework to predict adaptation behavior under an ideal observer planning model. Reaching movements were studied where a constant unidirectional force field acting perpendicular to the reaching movement was generated as a perturbation (see Fig. 8 (*left*)). Using the iLQG-LD model, the manipulator gets strongly deflected when reaching for the target because the learned dynamics model cannot yet account for the "spurious" forces. However, when the deflected trajectory is used as training data and the dynamics model is updated online, the tracking improves with each new successive trial (Fig. 8 (*left*)). Please refer to Mitrovic et al. (2008) for more details. Aftereffects upon removing the force field, very similar to those observed in human experiments, are also observed.



Locally Weighted Regression for Control. Figure 8. Adaptation to a unidirectional constant force field (indicated by the arrows). Darker lines indicate better trained models. In particular, the left-most trajectory corresponds to the "initial" control sequence, which was calculated using the LWPR model *before* the adaptation process. The fully "adapted" control sequence results in a nearly straight line reaching movement

Cross References

►Bias and Variance
►Dimensionality Reduction
►Incremental Learning
►Kernel Function
►Kernel Methods
►Lazy Learning
►Linear Regression
►Mixture Models
►Online Learning
►Overfitting
►Radial Basis Functions
►Regression
►Supervised Learning

Programs and Data

http://www-clmc.usc.edu/software
http://www.ipab.inf.ed.ac.uk/slmc/software/

Recommended Reading

Aström, K. J., & Wittenmark, B. (1989). *Adaptive control*. Reading, MA: Addison-Wesley.

Atkeson, C., Moore, A., & Schaal, S. (1997). Locally weighted learning. *AI Review, 11*, 11–73.

Atkeson, C. (1989). Using local models to control movement. In *Proceedings of the advances in neural information processing systems 1* (pp. 157–183). San Francisco, CA: Morgan Kaufmann.

Cleveland, W. S. (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association, 74*, 829–836.

Hastie, T., & Loader, C. (1993). Local regression: Automatic kernel carpentry. *Statistical Science, 8*, 120–143.

Jordan, M. I., & Jacobs, R. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation, 6*, 181–214.

Klanke, S., Vijayakumar, S., & Schaal, S. (2008). A library for locally weighted projection regression. *Journal of Machine Learning Research, 9*, 623–626.

Mitrovic, D., Klanke, S., & Vijayakumar, S. (2008). Adaptive optimal control for redundantly actuated arms. In *Proceedings of the 10th international conference on the simulation of adaptive behavior*, Osaka, Japan (pp. 93–102). Berlin: Springer-Verlag.

Schaal, S., & Atkeson, C. G. (1998). Constructive incremental learning from only local information. *Neural Computation, 10*(8), 2047–2084.

Schaal, S., Atkeson, C. G., & Vijayakumar, S. (2002). Scalable techniques from nonparametric statistics. *Applied Intelligence, 17*, 49–60.

Ting, J., D'Souza, A., Yamamoto, K., Yoshioka, T., Hoffman, D., Kakei, S., et al. (2005). Predicting EMG data from M1 neurons with variational Bayesian least squares. In *Proceedings of advances in neural information processing systems 18*, Cambridge: MIT Press.

Ting, J., Kalakrishnan, M., Vijayakumar, S., & Schaal, S. (2008). Bayesian kernel shaping for learning control. In *Proceedings of advances in neural information processing systems 21* (pp. 1673–1680). Cambridge: MIT Press.

Ting, J. (2009). Bayesian methods for autonomous learning systems. Ph.D. Thesis, Department of Computer Science, University of Southern California, 2009.

Todorov, E., & Li, W. (2004). A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of 1st international conference of informatics in control, automation and robotics*, Setúbal, Portugal.

Vijayakumar, S., D'Souza, A., & Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Computation, 17*, 2602–2634.

Logic of Generality

Luc De Raedt
Katholieke Universiteit Leuven
Heverlee, Belgium

Synonyms

Generality and logic; Induction as inverted deduction; Inductive inference rules; Is more general than; Is more specific than; Specialization

Definition

One hypothesis is *more general* than another one if it covers all instances that are also covered by the latter one. The former hypothesis is called a ▶*generalization* of the latter one, and the latter a ▶*specialization* of the former. When using logical formulae as hypotheses, the generality relation coincides with the notion of logical entailment, which implies that the generality relation can be analyzed from a logical perspective. The logical analysis of generality, which is pursued in this chapter, leads to the perspective of *induction as the inverse of deduction*. This forms the basis for an analysis of various logical frameworks for reasoning about generality and for traversing the space of possible hypotheses. Many of these frameworks (such as for instance, *θ-subsumption*) are employed in the field of ▶inductive logic programming and are introduced below.

Motivation and Background

Symbolic machine learning methods typically learn by searching a hypothesis space. The hypothesis space can be (partially) ordered by the ▶generality relation, which serves as the basis for defining operators to traverse the space as well as for pruning away unpromising parts of the search space. This is often realized through the use of ▶refinement operators, that is, generalization and specialization operators. Because many learning methods employ a ▶hypothesis language that is logical or that can be reformulated in logic, it is interesting to analyze the generality relation from a logical perspective. When using logical formulae as hypotheses, the generality relation closely corresponds to logical entailment. This allows us to directly transfer results from logic to a machine learning context. In particular, machine learning operators can be derived from logical inference rules. The logical theory of generality provides a framework for transferring these results. Within the standard setting of inductive logic programming, learning from entailment, specialization is realized through deduction, and generalization through induction, which is considered to be the inverse of deduction. Different deductive inference rules lead to different frameworks for generalization and specialization. The most popular one is that of θ-subsumption, which is employed by the vast majority of contemporary inductive logic programming systems.

Theory

A hypothesis g is *more general than* a hypothesis s if and only if g covers all instances that are also covered by s, more formally, if $\text{covers}(s) \subseteq \text{covers}(g)$, in which case, $\text{covers}(h)$ denotes the set of all instances covered by the hypothesis h.

There are several possible ways to represent hypotheses and instances in logic (De Raedt, 1997, 2008), each of which results in a different setting with a corresponding covers relation. Some of the best known settings are *learning from entailment*, learning from interpretations, and learning from proofs.

Learning from Entailment

In learning from entailment, both hypotheses and instances are logical formulae, typically *definite clauses*, which underlie the programming language Prolog (Flach, 1994). Furthermore, when learning from entailment, a hypothesis h covers an instance e if and only if $h \models e$, that is, when h logically entails e, or equivalently, when e is a logical consequence of h. For instance, consider the hypothesis h:

```
flies :- bird, normal.
bird :- blackbird.
bird :- ostrich.
```

The first clause or rule can be read as `flies` *if* `normal` *and* `bird`, that is, normal birds fly. The second and third states that blackbirds are birds. Consider now the examples e_1:

```
flies :- blackbird, normal, small.
```

and e_2:

```
flies :- ostrich, small.
```

Example e_1 is covered by h, because it is a logical consequence of h, that is, $h \models e_1$. On the other hand, example e_2 is not covered, which we denote as $h \not\models e_2$.

When learning from entailment, the following property holds:

Property 1 *A hypothesis g is more general than a hypothesis s if and only if g logically entails s, that is, $g \models s$.*

This is easy to see. Indeed, g is more general than s if and only if $\text{covers}(s) \subseteq \text{covers}(g)$ if and only if for all examples $e : (s \models e) \rightarrow (g \models e)$, if and only if $g \models s$. For instance, consider the hypothesis h_1:

```
flies :- blackbird, normal.
```

Because $h \models h_1$, it follows that h covers all examples covered by h_1, and hence, h generalizes h_1.

Property 1 states that the generality relation coincides with logical entailment when learning from entailment. In other learning settings, such as when *learning from interpretations*, this relationship also holds though the direction of the relationship might change.

Learning from Interpretations

In learning from interpretations, hypotheses are logical formulae, typically sets of definite clauses, and instances are interpretations. For propositional theories, interpretations are assignments of truth-values to propositional variables. For instance, continuing the `flies` illustration, two interpretations could be

```
{blackbird, bird, normal, flies} and
{ostrich, small}
```

where we specify interpretations through the set of propositional variables that are true. An interpretation specifies a kind of possible world. A hypothesis h then covers an interpretation if and only if the interpretation is a model for the hypothesis. An interpretation is a model for a hypothesis if it satisfies all clauses in the hypothesis. In our illustration, the first interpretation is a model for the theory h, but the second is not. Because the condition part of the rule `bird :- ostrich.` is satisfied in the second interpretation (as it contains `ostrich`), the conclusion part, that is, `bird`, should also belong to the interpretation in order to have a model. Thus, the first example is covered by the theory h, but the second is not.

When learning from interpretations, a hypothesis g is more general than a hypothesis s if and only if for all examples e: (e is a model of s) $\rightarrow$ (e is a model of g), if and only if $s \models g$.

Because the learning from entailment setting is more popular than the learning from interpretations setting, we shall employ in this section the usual convention that states that one hypothesis g is more general than a hypothesis s if and only if $g \models s$.

An Operational Perspective

Property 1 lies at the heart of the theory of *inductive logic programming* and generalization because it directly relates the central notions of logic with those of machine learning (Muggleton & De Raedt, 1994). It is also extremely useful because it allows us to directly transfer results from logic to machine learning.

This can be illustrated using traditional deductive inference rules, which start from a set of formulae and derive a formula that is entailed by the original set. For instance, consider the resolution inference rule for propositional definite clauses:

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n}. \quad (1)$$

This inference rule starts from the two rules above the line and derives the so-called *resolvent* below the line. This rule can be used to infer h_1 from h. An alternative deductive inference rule adds a condition to a rule:

$$\frac{h \leftarrow a_1, \ldots, a_n}{h \leftarrow a, a_1, \ldots, a_n}. \quad (2)$$

This rule can be used to infer that h_1 is more general than the clause used in example e_1. In general, a deductive inference rule can be written as

$$\frac{g}{s}. \tag{3}$$

If s can be inferred from g and the operator is *sound*, then $g \models s$. Thus, applying a deductive inference rule realizes specialization, and hence, deductive inference rules can be used as specialization operators. A *specialization operator* maps a hypothesis onto a set of its specializations. Because specialization is the inverse of generalization, *generalization operators* – which map a hypothesis onto a set of its generalizations – can be obtained by inverting deductive inference rules. The inverse of a deductive inference rule written in format (3) works from bottom to top, that is, from s to g. Such an inverted deductive inference rule is called an *inductive* inference rule. This leads to the view of induction as the inverse of deduction. This view is operational as it implies that each deductive inference rule can be inverted into an inductive one, and, also, that each inference rule provides an alternative framework for generalization.

An example of a generalization operator is obtained by inverting the adding condition rule (2). It corresponds to the well-known "dropping condition" rule (Michalski, 1983). As will be seen soon, it is also possible to invert the resolution principle (1).

Before deploying inference rules, it is necessary to determine their properties. Two desirable properties are *soundness* and *completeness*. These properties are based on the repeated application of inference rules. Therefore, we write $g \vdash_r s$ when there exists a sequence of hypotheses $h_1, \ldots, h_n$ such that

$$\frac{g}{h_1}, \frac{h_1}{h_2}, \ldots, \frac{h_n}{s} \text{ using } r. \tag{4}$$

A set of inference rules r is *sound* whenever $g \vdash_r s$ implies $g \models s$; and *complete* whenever $g \models s$ implies $g \vdash_r s$. In practice, soundness is always enforced though completeness is not always required in a machine learning setting. When working with incomplete rules, one should realize that the generality relation "$\vdash_r$" is weaker than the logical one "$\models$."

The most important logical frameworks for reasoning about generality, such as θ-subsumption and resolution, are introduced below using the above introduced logical theory of generality.

Frameworks for Generality

Propositional Subsumption

Many propositional learning systems employ hypotheses that consist of rules, often definite clauses as in the `flies` illustration above. The propositional subsumption relation defines a generality relation among clauses and is defined through the adding condition rule (2). The properties follow from this inference rule by applying the logical theory of generalization presented above. More specifically, the generality relation $\vdash_a$ induced by the adding condition rule states that a clause g is more general than a clause s, if s can be derived from g by adding a sequence of conditions to g. Observing that a clause $h \leftarrow a_1, \ldots, a_n$ is a disjunction of literals $h \vee \neg a_1 \vee \cdots \vee \neg a_n$ allows us to write it in set notation as $\{h, \neg a_1, \ldots, \neg a_n\}$. The soundness and completeness of propositional subsumption then follow from

$$g \vdash_a s \text{ if and only if } g \subseteq s \text{ if and only if } g \models s, \tag{5}$$

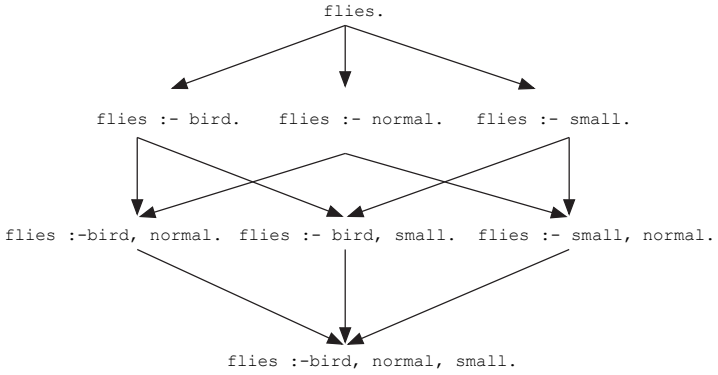
which also states that g subsumes s if and only if $g \subseteq s$.

The propositional subsumption relation induces a complete lattice on the space of possible clauses. A *complete lattice* is a partial order – a reflexive, antisymmetric, and transitive relation – where every two elements posses a unique least upper and greatest lower bound. An example lattice for rules defining the predicate `flies` in terms of `bird`, `normal`, and `small` is illustrated in the Hasse diagram depicted in Fig. 1.

The Hasse diagram also visualizes the different operators that can be used. The *generalization* operator ρ_g maps a clause to the set of its parents in the diagram, whereas the *specialization* operator ρ_s maps a clause to the set of its children. So far, we have defined such operators *implicitly* through their corresponding inference rules. In the literature, they are often defined *explicitly*:

$$\rho_g(h \leftarrow a_1, \ldots, a_n) = \{h \leftarrow a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n | i = 1, \ldots, n\}. \tag{6}$$

In addition to using the inference rules directly, some systems such as Golem (Muggleton & Feng, 1990)



Logic of Generality. Figure 1. The Hasse diagram for the predicate `flies`

also exploit the properties of the underlying lattice by computing the least upper bound of two formulae. The least upper bound operator is known under the name of least general generalization (lgg) in the machine learning literature. It returns the least common ancestor in the Hasse diagram. Using a set notation for clauses, the definition of the lgg is:

$$\text{lgg}(c_1, c_2) = c_1 \cap c_2. \tag{7}$$

The least general generalization operator is used by machine learning systems that follow a cautious generalization strategy. They take two clauses corresponding to positive examples and minimally generalize them.

θ-Subsumption

The most popular framework for generality within inductive logic programming is θ-subsumption (Plotkin, 1970). It provides a generalization relation for clausal logic and it extends propositional subsumption to first order logic.

A *definite clause* is an expression of the form $h \leftarrow a_1, \ldots, a_n$ where h and the a_i are logical atoms. An *atom* is an expression of the form $p(t_1, \ldots, t_m)$ where p is a *predicate name* (or, the name of a relation) and the t_i are terms. A *term* is either a constant (denoting an object in the domain of discourse), a variable, or a structured term of the form $f(u_1, \ldots, u_k)$ where f is a functor symbol (denoting a function in the domain of discourse) and the u_i are terms, see Flach (1994) for more details. Consider for instance the clauses

```
likes(X,Y) :- neighbours(X,Y).
likes(X,husbandof(Y)) :- likes(X,Y).
likes(X,tom) :- neighbours(X,tom),
                male(X).
```

The first clause states that `X likes Y` if `X` is a `neighbour` of `Y`. The second one that `X likes` the `husband` of `Y` if `X likes Y`. The third one that all `male neighbours` of `tom` like `tom`.

θ-Subsumption is based not only on the adding condition rule (2) but also on the *substitution rule*:

$$\frac{g}{g\theta}. \tag{8}$$

The substitution rule applies a substitution θ to the definite clause g. A substitution $\{V_1/t_1, \ldots, V_n/t_n\}$ is an assignment of terms to variables. Applying a substitution to a clause c yields the instantiated clause, where all variables are simultaneously replaced by their corresponding terms.

θ-subsumption is then the generality relation induced by the substitution and the adding condition rules. Denoting this set of inference rules by t, we obtain our definition of θ-subsumption:

g θ-subsumes s if and only if $g \vdash_t s$ if and only if
$$\exists\theta : g\theta \subseteq s. \tag{9}$$

For instance, the first clause for `likes` subsumes the third one with the substitution `{Y/tom}`.

θ-subsumption has some interesting properties:

- θ-subsumption is sound.
- θ-subsumption is complete for clauses that are not self-recursive. It is incomplete for self-recursive clauses such as

```
nat(s(X)) :- nat(X)
nat(s(s(Y))) :- nat(Y)
```

for which one can use resolution to prove that the first clause logically entails the second one, even though it does not θ-subsume it.

- Deciding θ-subsumption is an NP-complete problem.

Because θ-subsumption is relatively simple and decidable whereas logical entailment between single clauses is undecidable, it is used as the generality relation by the majority of inductive logic programming systems. These systems typically employ a specialization or refinement operator to traverse the search space. To guarantee systematic enumeration of the search space, the specialization operator ρ_s can be employed. $\rho_s(c)$ is obtained by applying the adding condition or substitution rule with the following restrictions.

- The adding condition rule only adds atoms of the form $p(V_1, \ldots, V_n)$, where the V_i are variables not yet occurring in the clause c.
- The substitution rule only employs *elementary substitutions*, which are of the form
  - $\{X/Y\}$, where X and Y are two variables appearing in c
  - $\{V/ct\}$, where V is a variable in c and ct a constant
  - $\{V/f(V_1, \ldots, V_n)\}$, where V is a variable in c, f a functor of arity n and the V_i are variables not yet occurring in c.

A generalization operator can be obtained by inverting ρ_s, which requires one to invert substitutions. Inverting substitutions is not easy. While applying a substitution θ = `{V/a}` to a clause c replaces all occurrences of `V` by `a` and yields a unique clause $c\theta$, applying the substitution rule in the inverse direction does not necessarily yield a unique clause. If we assume the elementary substitution applied to c with

$$\frac{c}{\texttt{q(a,a)}}. \tag{10}$$

was `{V/a}`, then there are at least three possibilities for c: `q(a,V)`, `q(V,a)`, and `q(V,V)`.

θ-subsumption is reflexive, transitive but unfortunately not anti-symmetric, which can be seen by considering the clauses

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- father(X,Y),
               father(U,V).
```

The first clause clearly subsumes the second one because it is a subset. The second one subsumes the first with the substitution {`X/U, V/Y`}. The two clauses are therefore equivalent under θ-subsumption, and hence also logically equivalent. The loss of the anti-symmetry complicates the search process. The naive application of the specialization operator ρ_s may yield syntactic specializations that are logically equivalent. This is illustrated above where the second clause for `parent` is a refinement of the first one using the adding condition rule. In this way, useless clauses are generated, and if the resulting clauses are further refined, there is a danger that the search will end up in an infinite loop.

Plotkin (1970) has studied the quotient set induced by θ-subsumption and proven various interesting properties. The quotient set consists of classes of clauses that are equivalent under θ-subsumption. The class of clauses equivalent to a given clause c is denoted by

$$[c] = \{c' | c' \text{ is equivalent with } c \text{ under } \theta\text{-subsumption}\}. \tag{11}$$

Plotkin proved that

- The quotient set is well-defined w.r.t. θ-subsumption.
- There is a representative, a canonical form, of each equivalence class, the so-called *reduced clause*. The reduced clause of an equivalence class is the shortest clause belonging to class. It is unique up to variable renaming. For instance, in the `parent` example above, the first clause is in reduced form.
- The quotient set forms a complete lattice, which implies that there is a least general generalization of two equivalence classes. In the inductive logic programming literature, one often talks about the least general generalization of two clauses.

Several variants of θ-subsumption have been developed. One of the most important ones is that of *OI*-subsumption (Esposito, Laterza, Malerba, & Semeraro, 1996). For functor-free clauses, it modifies the substitution rule by disallowing substitutions that unify two variables or that substitute a variable by a constant already appearing in the clause. The advantage is that the resulting relation is anti-symmetric, which avoids some of the above mentioned problems with refinement operators. On the other hand, the minimally general generalization of two clauses is not necessary unique, and hence, there exists no least general generalization operator.

Inverse Resolution

Applying resolution is a sound deductive inference rule and therefore realizes specialization. Reversing it yields inductive inference rules or generalization operators (Muggleton, 1987; Muggleton & Buntine, 1988). This is typically realized by combining the resolution principle with a copy operator. The resulting rules are called *absorption* (12) and *identification* (13). They start from the clauses below and induce the clause above the line. They are shown here only for the propositional case, as the first order case requires one to deal with substitutions as well as inverse substitutions.

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}, \quad (12)$$

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n \text{ and } h \leftarrow g, a_1, \ldots, a_n}. \quad (13)$$

Other interesting inverse resolution operators perform predicate invention, that is, they introduce new predicates that were not yet present in the original data. These operators invert two resolution steps. One such operator is the *intra-construction* operator (14). Applying this operator from bottom to top introduces the new predicate q that was not present before.

$$\frac{q \leftarrow l_1, \ldots, l_k \text{ and } p \leftarrow k_1, \ldots, k_n, q \text{ and } q \leftarrow l_1', \ldots, l_m'}{p \leftarrow k_1, \ldots, k_n, l_1, \ldots, l_k \text{ and } p \leftarrow k_1, \ldots, k_n, l_1', \ldots, l_m'}. \quad (14)$$

The idea of inverting the resolution operator is very appealing because it aims at inverting the most popular deductive inference operator, but is also rather complicated due to the non-determinism and the need to invert substitutions. Due to these complications, there are only few systems that employ inverse resolution operators.

Background Knowledge

Inductive logic programming systems employ background knowledge during the learning process. Background knowledge typically takes the form of a set of clauses B, which is then used by the covers relation. When learning from entailment in the presence of background knowledge B an example e is covered by a hypothesis h if and only if $B \cup h \models e$. This notion of coverage is employed in most of the work on inductive logic programming. In the intial `flies` example, the two clauses defining `bird` would typically be considered background knowledge.

The incorporation of background knowledge in the induction process has resulted in the frameworks for generality *relative* to a background theory. More formally, a hypothesis g is more general than a hypothesis s relative to the background theory B if and only if $B \cup g \models s$. The only inference rules that deal with multiple clauses are those based on (inverse) resolution. The other frameworks can be extended to cope with this generality relation following the logical theory of generalization. Various frameworks have been developed along these lines. Some of the most important ones are relative subsumption (Plotkin, 1971) and generalized subsumption (Buntine, 1998), which extend θ-subsumption and the notion of least general generalization toward the use of background knowledge. Computing the least general generalization of two clauses relative to the background theory is realized by first computing the most specific clauses covering the examples with regard to the background theory and then generalizing them using the least general generalization operator of θ-subsumption.

The first step is the most interesting one, and has been tackled under the name of *saturation* (Rouveirol, 1994) and *bottom-clauses* (Muggleton, 1995). We illustrate it within the framework of inverse entailment due to Muggleton (1995). The bottom clause $\bot(c)$ of a clause c with regard to a background theory B is the most specific clause $\bot(c)$ such that

$$B \cup \bot(c) \models c. \quad (15)$$

If B consist of

```
polygon :-  rectangle.
rectangle :- square.
oval :- circle.
```

and the example c is

```
positive :- red, square.
```

Then the bottom-clause $\bot(c)$ is

```
positive :- red, rectangle, square,
            polygon.
```

The bottom-clause is useful because it only lists those atoms that are relevant to the example, and only generalizations (under θ-subsumption) of $\bot(c)$ will cover the example. For instance, in the illustration, the bottom-clause mentions neither `oval` nor `circle` as clauses for `pos` containing these atoms will never cover the example clause c. Once the bottom-clause covering an example has been found the search process continues as if no background knowledge were present. Either specialization operators (typically under θ-subsumption) would search the space of clauses more general than $\bot(c)$, or the least general generalization of multiple bottom-clauses would be computed.

Equation (15) is equivalent to

$$B \cup \neg c \models \neg \bot(c), \tag{16}$$

which explains why the bottom-clause is computed by finding all factual consequences of $B \cup \neg c$ and then inverting the resulting clause again. On the example:

$\neg c$ = {¬ positive, red, square}

and the set of all consequences is

$\neg \bot(c)$ = $\neg c \cup$ {rectangle, polygon}

which then yields $\bot(c)$ mentioned above. When dealing with first order logic, bottom-clauses can become infinite, and therefore, one typically imposes further restrictions on the atoms that appear in bottom-clauses. These restrictions are part of the *language bias*.

The textbook by Nienhuys-Cheng and De Wolf (1997) is the best reference for an in-depth formal description of various frameworks for generality in logic, in particular, for θ-subsumption and some of its variants. The book by De Raedt (2008) contains a more complete introduction to inductive logic programming and relational learning, and also introduces the key frameworks for generality in logic. An early survey of inductive logic programming and the logical theory of generality is contained in Muggleton and De Raedt (1994). Plotkin (1970, 1971) pioneered the use θ-subsumption and relative subsumption (under a background theory) for machine learning. Buntine (1998) extended these frameworks toward generalized subsumption, and Esposito et al. (1996) introduced *OI*-subsumption. Inverse resolution was first used in the system Marvin (Sammut & Banerji, 1986), and then elaborated by Muggleton (1987) for propositional logic and by Muggleton and Buntine (1988) for definite clause logic. Various learning settings are studied by De Raedt (1997) and discussed extensively by De Raedt (2008). They are also relevant to ►probabilistic logic learning and ►statistical relational learning.

Recommended Reading

Buntine, W. (1998). Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence, 36*, 375–399.

De Raedt, L. (1997). Logical settings for concept learning. *Artificial Intelligence, 95*, 187–201.

De Raedt, L. (2008). *Logical and relational learning*. New York: Springer.

Semeraro, G., Esposito, F., & Malerba, D. (2006). Ideal Refinement of Datalog Programs. In *Proceedings of the 5th International Workshop on Logic Program Synthesis and Transformation, Lecture notes in computer science* (Vol. 1048, pp. 120–136). Springer.

Flach, P. A. (1994). *Simply logical: Intelligent reasoning by example*. New York: Wiley.

Michalski, R. S. (1983). A theory and methodology of inductive learning. *Artificial Intelligence, 20*(2), 111–161.

Muggleton, S. (1987). Duce, an oracle based approach to constructive induction. In *Proceedings of the 10th International Joint conference on Artificial Intelligence* (pp. 287–292). San Francisco: Morgan Kaufmann.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, 13*(3–4), 245–286.

Muggleton, S., & Buntine, W. (1988). Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Workshop on Machine Learning* (pp. 339–351). San Francisco: Morgan Kaufmann.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19/20*, 629–679.

Muggleton, S., & Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the 1st conference on Algorithmic Learning Theory* (pp. 368–381). Ohmsma, Tokyo, Japan.

Nienhuys-Cheng, S.-H., & de Wolf, R. (1997). *Foundations of inductive logic programming*. Berlin: Springer.
Plotkin, G. D. (1970). A note on inductive generalization. In *Machine intelligence* (Vol. 5, pp. 153–163). Edinburgh: Edinburgh University Press.
Plotkin, G. D. (1971). A further note on inductive generalization. In *Machine Intelligence* (Vol. 6, pp. 101–124). Edinburgh: Edinburgh University Press.
Rouveirol, C. (1994). Flattening and saturation: Two representation changes for generalization. *Machine Learning, 14*(2), 219–232.
Sammut, C., & Banerji, R. B. (1986). Learning concepts by asking questions. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2, pp. 167–192). San Francisco: Morgan Kaufmann.

Logic Program

A logic program is a set of logical rules or ►clauses. Logic programs are employed to answer queries using the ►resolution inference rule. For example, consider the following logic program:

```
grandparent(X,Y) :- parent(X,Z),
                    parent(Z,Y).

parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).

father(charles, william).
   mother(diana, william).
father(philip, charles).
   mother(elizabeth, charles).

father(john, diana).
   mother(frances, diana).
```

Using resolution we obtain the following answers to the query `:-grandparent(X,Y)`:

```
X = philip,     Y = william ;
X = john,       Y = william ;
X = elizabeth,  Y = william ;
X = frances,    Y = william.
```

Cross References

►Clause
►First-Order Logic
►Prolog

Logical Consequence

►Entailment

Logical Regression Tree

►First-Order Regression Tree

Logistic Regression

Synonyms

Logit model

Definition

Logistic regression provides a mechanism for applying the techniques of ►linear regression to ►classification problems. It utilizes a linear regression model of the form

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

where x_1 to x_n represent the values of the n attributes and β_0 to β_n represent weights. This model is mapped onto the interval [0,1] using

$$P(c_0 \mid x_1 \ldots x_n) = \frac{1}{1 + e^{-z}}$$

where c_0 represents class 0.

Recommended Reading

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning* (2nd ed.). New York: Springer.

Logit Model

►Logistics Regression

Log-Linear Models

►Maximum Entropy Models for Natural Language Processing

Long-Term Potentiation of Synapses

By a suitable induction protocol, the connection between two neurons can be strengthened. If this change persists for hours, the effect is called a long-term potentation.

LOO Error

►Leave-One-Out Error

Loopy Belief Propagation

Loopy belief propagation is a heuristic inference algorithm for ►Bayesian networks. See ►Graphical Models for details.

Loss

Synonyms
Cost

Definition
The *cost* or *loss* of a prediction y', when the correct value is y, is a measure of the relative utility of that prediction given that correct value. A common loss function used with ►classification learning is ►*zero-one loss*. Zero-one loss assigns 0 to loss for a correct classification and 1 for an incorrect classification. ►Cost sensitive classification assigns different costs to different forms of misclassification. For example, misdiagnosing a patient as having appendicitis when he or she does not might be of lower cost than misdiagnosing the patient as not having it when he or she does. A common loss function used with ►regression is ►error squared. This is the square of the difference between the predicted and true values.

Loss Function

Synonyms
Cost function

Definition
A loss function is a function used to determine ►loss.

LWPR

►Locally Weighted Regression for Control

LWR

►Locally Weighted Regression for Control

M

m-Estimate

►Rule Learning

Machine Learning and Game Playing

Johannes Fürnkranz
Darmstadt, Germany

Definition

Game playing is a major application area for research in artificial intelligence in general (Schaeffer & van den Herik, 2002) and for machine learning in particular (Fürnkranz & Kubat, 2001). Traditionally, the field is concerned with learning in strategy games such as tic-tac-toe (Michie, 1963), checkers (►Samuel's Checkers Player), backgammon (►TD-Gammon), chess (Baxter et al., 2000; Björnsson & Marsland, 2003; Donninger & Lorenz, 2006; Sadikov & Bratko, 2006), Go (Stern et al., 2006), Othello (Buro, 2002), poker (Billings, Peña, Schaeffer, & Szafron, 2002), or bridge (Amit & Markovitch, 2006). However, recently computer and video games have received increased attention (Laird & van Lent, 2001; Ponsen, Muñoz-Avila, Spronck, & Aha, 2006; Spronck, Ponsen, Sprinkhuizen-Kuyper, & Postma, 2006).

Motivation and Background

Since the early days of the field, game playing applications have been popular testbeds for machine learning. This has several reasons:

- *Games allow to focus on intelligent reasoning:* Other components of intelligence, such as perception or physical actions can be ignored.
- *Games are easily accessible:* A typical game-playing environment can be implemented within a few days, often hours. Exceptions are real-time computer games, for which only a few open-source test beds exist.
- *Games are very popular:* It is not very hard to describe the agent's task to the general public, and they can easily appreciate the achieved level of intelligence.

There are various types of problems that keep reoccurring in game-playing applications, for which solutions with machine learning methods are desirable, including opening book learning, learning of evaluation functions, player modeling, and others, which will be dealt with here.

Structure of the Learning System

Game-playing applications offer various challenges for machine learning. A wide variety of learning techniques have been used for tackling these problems. We cannot provide details on the learning algorithms here but will instead focus on the problems and give some of the most relevant and recent pointers to the literature. A more detailed survey can be found in Fürnkranz (2001).

Learning of Evaluation Functions

The most extensively studied learning problem in game playing is the automatic adjustment of the weights of an evaluation function. Typically, the situation is as follows: the game programmer has provided the program with a library of routines that compute important features of the current board position (e.g., the number of pieces of each kind on the board, the size of the territory controlled, etc.). What is not known is how to combine these pieces of knowledge and how to quantify their relative importance. Most frequently, these parameters are combined linearly, so that the learning task is to adjust the weights of a weighted sum. The main problem is that there are typically no direct target values that could be used as training signals. Exceptions are games

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

or endgames that have been solved completely, which are treated further below. However, in general, algorithms use ►Preference Learning (where pairs of moves or positions are labeled according to which one is preferred by an expert player) or ►Reinforcement Learning (where moves or positions are trained based on information about the eventual outcome of the game) for tuning the evaluation functions.

The key problem with reinforcement learning approaches is the ►Credit Assignment problem, i.e., even though a game has been won (lost), there might be bad (good) moves in the game. Reinforcement learning takes a radical stance at this problem, giving all positions the same reinforcement signal, hoping that erroneous signals will be evened out over time. An early classic in this area is MENACE (Michie, 1963), a tic-tac-toe player who simulates reinforcement learning with delayed rewards using a stack of matchboxes, one for each position. Each box contains a number of beads in different colors, which represent the different legal moves in the position. Moves are selected by randomly drawing a bead out of the box that represents the current position. After a game is won, all played moves are reinforced by adding beads of the corresponding colors to these boxes, and in the case of a lost game, corresponding beads are removed, thereby decreasing the probability that the same move will be played again.

The premier example of a system that has tuned its evaluation function to expert strength by playing millions of games against itself is the backgammon program ►TD-Gammon. Its key innovation was the use of a ►Neural Network instead of a position table, so that the reinforcement signal can be generalized to new unseen positions. Many authors have tried to copy TD-Gammon's learning methodology to other games (Ghory, 2004). None of these successors, however, achieved a performance that was as impressive as TD-Gammon's. The reason for this seems to be that backgammon has various characteristics that make it perfectly suited for learning from self-play. Foremost among these are the fact that the dice rolls guarantee sufficient variability, which allows to use training by self-play without the need for an explicit exploration/exploitation trade-off, and that it only requires a very limited amount of search, which allows to ignore the dependencies of search algorithm and search heuristic. These points have, for example, been addressed with limited success in the game of chess, where the program KnightCap (Baxter et al., 2000), which integrates ►Temporal Difference Learning into a game tree search by using the final positions of the principal variation for updates, and by using play on a game server for exploration.

Many aspects of evaluation function learning are still discussed in the current literature, including whether there are alternatives to reinforcement learning (e.g., evolutionary algorithms), which training strategies should be used (e.g., self-play vs. play against a teacher), etc. One of the key problems, which has already been mentioned in Samuel's Checkers Player, namely the automated construction of useful features remains largely unsolved. Some progress has, e.g., been made in the game of Othello, where a simple algorithm, very much like ►APriori has been shown to produce valuable conjunctions of basic features (Buro, 2002).

Learning Search Control

A more challenging, but considerably less investigated task is to automatically tune the various parameters that control the search in game-playing programs. These parameters influence, for example, the degree to which the search algorithm is aggressive in pruning the unpromising parts of the search tree and the lines that are explored in more depth. The key problem here is that these parameters are intertwined with the search algorithm, and cannot be optimized independently, making the process very tedious and expensive.

There have been a few attempts to use ►Explanation-Based Learning to automatically learn predicates that indicate which branches of the search tree are the most promising to follow. These approaches are quite related to various uses of ►Explanation-Based Learning in Planning, but these could not be successfully be carried over to game-tree search.

Björnsson & Marsland (2003) present a *gradient descent* approach that minimizes the total number of game positions that need to be searched in order to successfully solve a number of training problems. The idea is to adjust each parameter in proportion to its sensitivity to changes in the number of searched nodes, which is estimated with additional searches. The amount of positions that can be searched for each training position

is bounded to avoid infinite solution times for individual problems, and simulated annealing is used to ensure convergence.

Opening Book Learning

Human game players not only rely on their ability to estimate the value of moves and positions but are often also able to play certain positions "by heart," i.e., without having to think about their next move. This is the result of home preparation, opening study, and *rote learning* of important lines and variations. As computers do not forget, the use of an opening book provides an easy way for increasing their playing strength. However, the construction of such opening books can be quite laborious, and the task of keeping it up-to-date is even more challenging.

Commercial game-playing programs, in particular chess programs, have thus resorted to tools that support the automatic construction of opening from large game databases. The key challenge here is that one cannot rely on statistical information alone: a move that has been successfully employed in hundreds of games may be refuted in a single game. (Donninger & Lorenz, 2006) describe an approach that evaluates the "goodness" of a move based on a heuristic formula that has been found by experimentation. This value is then added to the result of a regular alpha-beta search. The technique has been so successful, that the chess program Hydra, probably the strongest chess program today, has abandoned conventional large man-made (and therefore error-prone) error books. Similar techniques have also been used in games like Othello (Buro, 2002).

Pattern Discovery

In addition to databases of common openings and huge game collections, which are mostly used for the tuning of evaluation functions or the automatic generation of opening books (see above), many games or subgames have already been solved, i.e., databases in which the game-theoretic value of positions of these subgames can be looked up are available. For example, all endgames with up to six pieces in chess have been solved. Other games, such as Connect-4, are solved completely, i.e., all possible positions have been evaluated and the game-theoretic value of the starting position has been determined. The largest game that has been solved so far is checkers Many of these databases are readily available, some of them (in the domains of chess, Connect-4, and tic-tac-toe) are part of the ▶UCI Repository for machine-learning databases.

The simplest learning task is to train a classifier that is able to decide whether a given game position is a game-theoretical win or loss (or draw). In many cases, this is insufficient. For example, in the chess endgame king-rook-king, any position in which the white rook cannot be immediately captured, and in which black is not stalemate is, in principle, won by white. However, in order to actually win the game it is not sufficient to simply make moves that avoid rook captures and stalemates. Thus, most databases contain the maximal number of moves that are needed for winning the position. Predicting this is a much harder, largely unsolved problem (some recent work can be found in (Sadikov & Bratko, 2006)). In addition to the game-specific knowledge that could be gained by the extraction of patterns that are indicative of won positions, another major application could be a knowledge-based compression of these databases (the collection of all perfect-play chess endgame databases with up to six men is 1.2 Terabytes in a very compressed database format, the win/loss checkers databases with up to ten men contain about 4×10^{13} positions compressed into 215GB (Schaeffer et al., 2003)).

Player Modeling

Player modeling is an important research area in game playing, which can serve several purposes. The goal of *opponent modeling* is to improve the capabilities of the machine player by allowing it to adapt to its opponent and exploit his weaknesses. Even if a game-theoretical optimal solution to a game is known, a system that has the capability to model its opponent's behavior may obtain a higher reward. Consider, for example, the game of *rock-paper-scissors* aka *RoShamBo*, in which either player can expect to win one third of the game (with one third of draws) if both players play their optimal strategies (i.e., randomly select one of their three moves). However, against a player who always plays *rock*, a player who is able to adapt his strategy to always playing *paper* can maximize his reward, while a player who sticks with the "optimal" random strategy will still win only one third of the game. One of the grand challenges in this line of work are games such as poker, in

which opponent modeling is crucial to improve over game-theoretical optimal play (Billings et al., 2002).

Player modeling is also of increasing importance in commercial computer games (see below). For one, ►Behavioral Cloning techniques could be used to increase the playing strength or credibility of artificial characters by copying the strategies of expert human players. Moreover, the playing strength of the characters can be adapted to the increasing skill level of the human player. Finally, agents that can be trained by non-programmers can also play an important role. For example, in massive multiplayer online role-playing games (MMORGs), an avatar that is trained to simulate a user's game-playing behavior could take his creator's place at times when the human player cannot attend to his game character.

Commercial Computer Games

In recent years, the computer games industry has discovered Artificial Intelligence as a necessary ingredient to make games more entertaining and challenging and, vice versa, AI has discovered computer games as an interesting and rewarding application area (Laird & van Lent, 2001). In comparison to conventional strategy games, computer game applications are more demanding, as the agents in these game typically have to interact with a large number of partner or enemy agents in a highly dynamic, real-time environment, with incomplete knowledge about its states. Tasks include off-line or on-line player modeling (see above), virtual agents with learning capabilities, optimization of plans and processes, etc.

Computer players in games are often controlled with scripts. *Dynamic scripting* (Spronck et al., 2006) is an on-line ►Reinforcement Learning technique that is designed to be integrated into scripting languages of game playing agents. Contrary to conventional reinforcement learning agents, it updates the weights of all actions for a given state simultaneously. This sacrifices guaranteed convergence, but this is desirable in a highly dynamic game environment. The approach was successfully applied to improving the strength of computer-controlled characters and increasing the entertainment value of the game by automated scaling of the difficult level of the game AI to the human player's skill level. Similar to the problem of constructing suitable features for the use in evaluation functions, the basic tactics of the computer player had to be handcoded. Ponsen et al. (2006) extend dynamic scripting with an ►Evolutionary Algorithm for automatically constructing the tactical behaviors.

Machine learning techniques are not only used for controlling players, but also for tasks like skill estimation. For example, TrueSkill^{TM} (Herbrich et al., 2007), a Bayesian skill rating system which is used for ranking players in games on the Microsoft's Xbox 360. SAGA-ML (Southey et al., 2005) is a machine learning system for supporting game designers in improving the playability of a game.

Despite the large commercial potential, research in this area has just started, and the number of workshops and publications on this topic is rapidly increasing. For more information on AI Game Development we refer to http://aigamedev.com.

Cross References

►Samuel's Checkers Player
►TD-Gammon

Recommended Reading

Amit, A., & Markovitch, S. (2006). Learning to bid in bridge. *Machine Learning, 63*(3), 287–327.

Baxter, J., Tridgell, A., & Weaver, L. (2000). Learning to play chess using temporal differences. *Machine Learning, 40*(3), 243–263.

Billings, D., Peña, L., Schaeffer, J., & Szafron, D. (2002). The challenge of poker. *Artificial Intelligence, 134*(1–2), 201–240. Special issue on games, computers and artificial intelligence.

Björnsson, Y., & Marsland, T. A. (2003). Learning extension parameters in game-tree search. *Information Sciences, 154*(3–4), 95–118.

Bowling, M., Fürnkranz, J., Graepel, T., & Musick, R. (2006). Special issue on machine learning and games. *Machine Learning, 63*(3), 211–215.

Buro, M. (2002). Improving heuristic mini-max search by supervised learning. *Artificial Intelligence, 134*(1–2), 85–99. Special issue on games, computers and artificial intelligence.

Donninger, C., & Lorenz, U. (2006). Innovative opening-book handling. In H. J. van den Herik, S.-C. Hsu, & H. H. L. M. Donkers, (Eds.), *Advances in computer games*. Berlin: Springer.

Fürnkranz, J. (2001). Machine learning in games: A survey. In J. Fürnkranz & M. Kubat (Eds.), *Machines that learn to play games* (Chap. 2, pp. 11–59). Huntington, NY: Nova Science Publishers.

Fürnkranz, J., & Kubat, M. (Eds.). (2001). *Machines that learn to play games*. Huntington, NY: Nova Science Publishers.

Ghory, I. (2004). *Reinforcement learning in board games*. Technical Report CSTR-04-004. Department of Computer Science, University of Bristol, Bristol, UK. http://www.cs.bris.ac.uk/Publications/pub_info.jsp?id=2000100.

Herbrich, R., Minka, T., & Graepel, T. (2007). Trueskill^{TM}: A bayesian skill rating system. In B. Schölkopf, J. C. Platt, &

T. Hoffman (Eds.), *Advances in neural information processing systems (NIPS-06)* (Vol. 19, pp. 569–576). Vancouver, BC, Canada: MIT Press.

Laird, J. E., & van Lent, M. (2001). Human-level AI's killler application: Interactive computer games. *AI Magazine, 22*(2), 15–26.

Michie, D. (1963). Experiments on the mechanization of game-learning – Part I. Characterization of the model and its parameters. *The Computer Journal, 6*, 232–236.

Ponsen, M., Muñoz-Avila, H., Spronck, P., & Aha, D. W. (2006). Automatically generating game tactics via evolutionary learning. *AI Magazine, 27*(3), 75–84.

Sadikov, A., & Bratko, I. (2006). Learning long-term chess strategies from databases. *Machine Learning, 63*(3), 329–340.

Schaeffer, J., Björnsson, Y., Burch, N., Lake, R., Lu, P., & Sutphen, S. (2003). Building the checkers 10-piece endgame databases. In H. J. van den Herik, H. Iida, & E. A. Heinz (Eds.), *Advances in computer games* (Vol. 10, pp. 193–210). Graz, Austria: Springer.

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S. (2007). Checkers is solved. *Science* 317(5844):1518–1522.

Schaeffer, J., & van den Herik, H. J. (Eds.) (2002). *Chips challenging champions: Games, computers and artificial intelligence*. Amsterdam: North-Holland. (Reprint of a special issue of *Artificial Intelligence, 134*(1–2)).

Southey, F., Xiao, G., Holte, R. C., Trommelen, M., & Buchanan, J. W. (2005). Semi-automated gameplay analysis by machine learning. In R. M. Young & J. E. Laird (Eds.), *Proceedings of the 1st artificial intelligence and interactive digital entertainment conference (AIIDE-05)* (pp. 123–128). AAAI Press: Marina del Rey, CA.

Spronck, P., Ponsen, M. J. V., Sprinkhuizen-Kuyper, I. G., & Postma, E. O. (2006). Adaptive game AI with dynamic scripting. *Machine Learning, 63*(3), 217–248.

Stern, D., Herbrich, R., & Graepel, T. (2006). Bayesian pattern ranking for move prediction in the game of Go. In *Proceedings of the 23rd international conference on machine learning (ICML-06)* (pp. 873–880). New York: ACM.

Machine Learning for IT Security

Philip K. Chan
Florida Institute of Technology, Melbourne, FL, USA

Definition

The prevalence of information technology (IT) across all segments of society, greatly improves the accessibility of information, however, it also provides more opportunities for individuals to act with malicious intent. Intrusion detection is the task of identifying attacks against computer systems and networks. Based on data/behavior observed in the past, machine learning methods can automate the process of building detectors for identifying malicious activities.

Motivation and Background

Cyber security often focuses on preventing attacks using authentication, filtering, and encryption techniques, but another important facet is detecting attacks once the preventive measures are breached. Consider a bank vault: thick steel doors prevent intrusions, while motion and heat sensors detect intrusions. Prevention and detection complement each other to provide a more secure environment.

How do we know if an attack has occurred or has been attempted? This requires analyzing huge volumes of data gathered from the network, host, or file systems to find suspicious activities. Two general approaches exist for this problem: *misuse detection* (also known as *signature detection*), where we look for patterns signaling well-known attacks, and *anomaly detection*, where we look for deviations from normal behavior.

Misuse detection usually works reliably on known attacks (though false alarms and missed detections are not uncommon), but has the obvious disadvantage of not being capable of detecting new attacks. Though anomaly detection can detect novel attacks, it has the drawback of not being capable of discerning intent; it can only signal that some event is unusual, but not necessarily hostile, thus generating false alarms. A desirable system would employ both approaches. Misuse detection methods are more well understood and widely applied; however, anomaly detection is much less understood and more challenging.

Can we automate the process of building software for misuse and anomaly detection? Machine learning techniques hold promise in efficiently analyzing large amounts of recent activities, identifying patterns, and building detectors.

Besides computer attacks, spam email messages, though not intended to damage computer systems or data, are annoying and waste system resources. To construct spam detectors from large amounts of email messages, machine learning techniques have been used (see References and Recommended Reading for more).

Structure of Learning System

Machine learning can be used to construct models for misuse as well as anomaly detection.

Misuse Detection

For misuse detection, the machine learning goal is to identify characteristics of known attacks. One approach is to learn the difference between attacks and normal events, which can be casted as a classification problem. Given examples of labeled attacks and normal events, a learning algorithm constructs a model that differentiates attacks from normal events.

Lee, Stolfo, and Mok (1999) apply machine learning to detect attacks in computer networks. They first identify frequent episodes, associations of features that frequently appear within a time frame, in attack and normal data separately. Frequent episodes that only appear in attack data help construct features for the models. For example, if the SYN flag is set for a http connection is a frequent episode within 2 s and the episode only appears in the attack data, a feature is constructed for the number of http connections with the SYN flag set within a period of 2 s. Using RIPPER and based on different sets of features, they construct three models: traffic, host-based traffic, and content models. The three models are then combined using meta-learning.

Ghosh and Schwartzbard (1999) use neural networks to identify attacks in operating systems. Based on system calls in the execution traces of normal and attack programs, they first identify a number of "examplar" sequences of system calls. For each system call sequence, they calculate the distance from the examplar sequences. The number of input nodes for the neural network is equal to the number of examplars and values for the input nodes are distances from those examplar sequences. The value for the output node is whether the system call sequence is from an attack or normal program.

Anomaly Detection

For anomaly detection, the machine learning goal is to characterize normal behavior. The learned models of normal behavior are then used to identify events that are anomalies, events that deviate from the models. Since anomalies are not always attacks, to reduce false alarms, the learned models usually provide a scoring mechanism to indicate the degree of anomaly.

Warrender, Forrest, and Pearlmutter (1999) identify anomalies in system calls in the operating systems. The model is a table of system call sequences from execution traces of normal programs. During detection, a sequence that is not in the table or occurs less than 0.001% in the training data is considered a mismatch. The number of mismatches within a locality frame of 20 sequences is the anomaly score.

Mahoney and Chan (2003) introduce the LERAD algorithm for learning rules that identify anomalies in network traffic. LERAD first uses a randomized algorithm to generate candidate rules that represent associations. It then finds a set of high quality rules that can succinctly cover the training data. Each rule has an associated probability of violating the rule. During detection, based on the probability, LERAD provides a score for anomalous events that do not conform to the rules in the learned model.

Misuse Detection: Schultz, Eskin, Zadok, and Stolfo (2001) with program executables, Maxion and Townsend (2002) with user commands.

Anomaly Detection: Sekar, Bendre, Dhurjati, and Bollinen (2001) with program execution, Apap, Honig, Hershkop, Eskin, and Stolfo (2002) with Windows Registry, Anderson, Lunt, Javitz, Tamaru, and Valdes (1995) with system resources, Lane and Brodley (1999) with user commands.

Spam detection: Bratko, Filipic, Cormack, Lynam, and Zupan (2006) with text, Fumera, Pillai, and Roli (2006) with text and embedded images.

Cross References

►Association
►Classification

Recommended Reading

Anderson, D., Lunt, T., Javitz, H., Tamaru, A., & Valdes, A. (1995). Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (NIDES). Technical Report SRI-CSL-95-06, SRI.

Apap, F., Honig, A., Hershkop, S., Eskin, E., & Stolfo, S. (2002). Detecting malicious software by monitoring anomalous windows registry accesses. In *Proceeding of fifth international symposium on recent advances in intrusion detection (RAID)*, (pp. 16–18). Zurich, Switzerland.

Bratko, A., Filipic, B., Cormack, G., Lynam, T., & Zupan, B. (2006). Spam filtering using statistical data compression models. *Journal of Machine Learning Research, 7*, 2673–2698.

Fumera, G., Pillai, I., & Roli, F. (2006). Spam filtering based on the analysis of text information embedded into images. *Journal of Machine Learning Research, 7*, 2699–2720.

Ghosh, A., & Schwartzbard, A. (1999). A study in using neural networks for anomaly and misuse detection. In *Proceeding of 8th USENIX security symposium* (pp. 141–151). Washington, DC.

Lane, T., & Brodley, C. (1999). Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security, 2*(3), 295–331.

Lee, W., Stolfo, S., & Mok, K. (1999). A data mining framework for building intrusion detection models. In *IEEE symposium on security and privacy* (pp. 120–132).

Mahoney, M., & Chan, P. (2003). Learning rules for anomaly detection of hostile network traffic. In *Proceeding of IEEE international conference data mining* (pp. 601–604). Melbourne, FL.

Maxion, R., & Townsend, T. (2002). Masquerade detection using truncated command lines. In *Proceeding of international conferernce dependable systems and networks (DSN)* (pp. 219–228). Washington, DC.

Schultz, M., Eskin, E., Zadok, E., & Stolfo, S. (2001). Data mining methods for detection of new malicious executables. In *Proceeding of IEEE symposium security and privacy* (pp. 38–49). Oakland, CA.

Sekar, R., Bendre, M., Dhurjati, D., & Bollinen, P. (2001). A fast automaton-based method for detecting anomalous program behaviors. In *Proceeding of IEEE symposium security and privacy* (pp. 144–155). Oakland, CA.

Warrender, C., Forrest, S., & Pearlmutter, B. (1999). Detecting intrusions using system calls: Alternative data models. In *IEEE symposium on security and privacy* (pp. 133–145). Los Alamitos, CA.

Manhattan Distance

Susan Craw
The Robert Gordon University, Scotland, UK

Synonyms

City Block distance; L_1-distance; 1-norm distance; Taxicab norm distance

Definition

The Manhattan distance between two points $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ in n-dimensional space is the sum of the distances in each dimension.

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} | x_i - y_i | .$$

It is called the Manhattan distance because it is the distance a car would drive in a city (e.g., Manhattan) where the buildings are laid out in square blocks and the straight streets intersect at right angles. This explains the other terms City Block and taxicab distances. The terms L_1 and 1-norm distances are the mathematical descriptions of this distance.

Cross References

►Case-Based Reasoning
►Nearest Neighbor

Margin

Definition

In a ►Support Vector Machine, a *margin* is the distance between a hyperplane and the closest example.

Cross References

►Support Vector Machines

Market Basket Analysis

►Basket Analysis

Markov Blanket

►Graphical Models

Markov Chain

►Markov Process

Markov Chain Monte Carlo

Claude Sammut
University of New South Wales
Sydney, Australia

Synonyms

MCMC

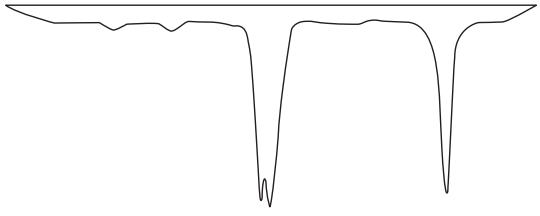
Definition

A Markov Chain Monte Carlo (MCMC) algorithm is a method for sequential sampling in which each new sample is drawn from the neighborhood of its predecessor. This sequence forms a ►Markov chain,

since the transition probabilities between sample values are only dependent on the last sample value. MCMC algorithms are well suited to sampling in high-dimensional spaces.

Motivation

Sampling from a probability density function is necessary in many kinds of approximation, including Bayesian inference and other applications in Machine Learning. However, sampling is not always easy, especially in high-dimensional spaces. Mackay (2003) gives a simple example to illustrate the problem. Suppose we want to find the average concentration of plankton in a lake, whose profile looks like this:



If we do not know the depth profile of the lake, how would we know where to sample from? If we take a boat out, would we have to sample almost exhaustively by fixing a grid on the surface of the lake and sinking our instrument progressively deeper, sampling at fixed intervals until we hit the bottom? This would be prohibitively expensive and if we had a similar problem, but with more dimensions, the problem becomes intractable. If we try to simplify the problem by drawing a random sample, how do we ensure that enough samples are taken from the canyons in the lake and not just the shallows, which account for most of the surface area?

The Algorithm

The general approach adopted in MCMC algorithms is as follows. We start sampling in some random initial state, represented by vector, $\boldsymbol{x}$. At each state, we can evaluate the probability density function, $P(\boldsymbol{x})$. We then choose a candidate next state, $\boldsymbol{x}'$, near the current state and evaluate $P(\boldsymbol{x}')$. Comparing the two, we decide whether to accept or reject the candidate. If we accept it, the candidate becomes the new current state and the process repeats for a fixed number of steps or until some convergence criterion is satisfied.

Algorithm 1 The Metropolis Algorithm

Given: target probability density function $P(\boldsymbol{x})$
a proposal distribution, Q, e.g., a Gaussian
the number of iterations, N
Output: a set of samples $\{x_i\}$ drawn from $P(\boldsymbol{x})$
Randomly select initial state vector, $\boldsymbol{x}_0$
for $i = 0$ **to** $N - 1$
  create a new candidate $\boldsymbol{x}' = \boldsymbol{x}_i + \Delta\boldsymbol{x}$,
    where $\Delta\boldsymbol{x}$ is randomly chosen from $Q(\Delta\boldsymbol{x})$
  set $\alpha = \frac{P(x')}{P(x_i)}$
  if $\alpha \geq 1$ or with probability α
    accept the new candidate and set $\boldsymbol{x}_{i+1} = \boldsymbol{x}'$
  else
    reject the candidate and set $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i$

The Metropolis Algorithm

There are several variants of the general algorithm presented above. Each variant must specify how a candidate state is proposed and what criterion should be used to accept or reject the candidate. The Metropolis algorithm assumes that the next candidate is drawn from a symmetric distribution, $Q(x)$, centered on the current state, for example, a Gaussian distribution (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953; Metropolis & Ulam, 1949). This distribution is called the *proposal distribution*. The Metropolis algorithm is shown in Algorithm 1.

To decide if a candidate should be accepted or rejected, the algorithm calculates,

$$\alpha = \frac{P(x')}{P(x_i)}$$

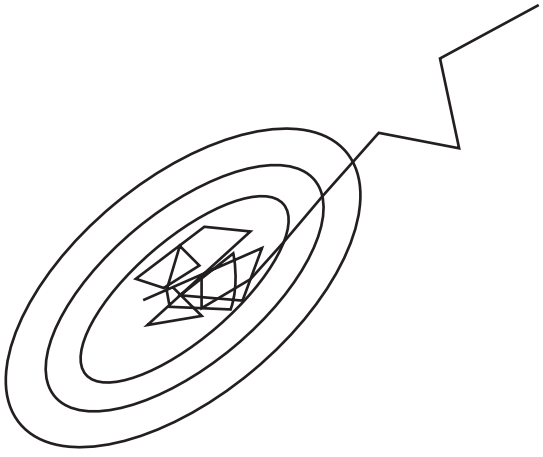
where $\boldsymbol{x}_i$ is the current state and $\boldsymbol{x}'$ is the candidate state. If $\alpha > 1$, the candidate is immediately accepted. If $\alpha < 1$, then a stochastic choice is made with the candidate being accepted with probability α, otherwise, it is rejected.

Hastings (1970) introduced a variant, the Metropolis–Hastings algorithm, which allows the proposal distribution to be asymmetric. In this case, the accept/reject calculation is:

$$\alpha = \frac{P(x')Q(x_i; x')}{P(x_i)Q(x'; x_i)}$$

Burn-in and Convergence

It can be difficult to decide how many iterations are needed before an MCMC algorithm achieves a stable distribution. Several factors affect the length of the Markov chain needed. Depending on the start state, many of the initial samples may have to be discarded, called *burn-in*, as illustrated below. The ellipses represent contours of the distribution.



The variance of the proposal distribution can also affect the chain length. If the variance is large, the jumps are large, meaning that there is varied sampling. However, this is also likely to mean that fewer samples are accepted. Narrowing the variance should increase acceptance but may require a long chain to ensure wide sampling, which is particularly necessary if the distribution has several peaks. See Andrieu et al. (2003) for a discussion of methods for improving convergence times.

Gibbs Sampling

An application of MCMC is inference in a ►Bayesian network, also known as ►Graphical Models. Here, we sample from evidence variables to find a probability for non-evidence variables. That is, we want to know what unknowns we can derive from the knowns and with what probability. Combining the evidence across a large network is intractable because we have to take into account all possible interactions of all variables, subject to the dependencies expressed in the network. Since there are too many combinations to compute in a large network, we approximate the solution by sampling.

The Gibbs sampler is a special case of the Metropolis–Hastings algorithm that is well suited to sampling from distributions over two or more dimensions. It proceeds as in Algorithm 1, except that when a new candidate is generated, only one dimension is allowed to change while all the others are held constant. Suppose we have n dimensions and $\boldsymbol{x} = (x_1, \ldots, x_n)$. One complete pass consists of jumping in one dimension, conditioned on the values for all the other dimensions, then jumping in the next dimension, and so on. That is, we initialise $\boldsymbol{x}$ to some value, and then for each x_i we resample $P(x_i|x_{j=6i})$ for j in $1 \ldots n$. The resulting candidate is immediately accepted. We then iterate, as in the usual Metropolis algorithm.

Cross References

►Bayesian Network
►Graphical Models
►Learning Graphical Models
►Markov Chain

Recommended Reading

MCMC is well covered in several text books. Mackay (2003) gives a thorough and readable introduction to MCMC and Gibbs Sampling. Russell and Norvig (2009) explain MCMC in the context of approximate inference for Bayesian networks. Hastie et al. (2009) also give a more technical account of sampling from the posterior. Andrieu et al. (2003) Machine Learning paper gives a thorough introduction to MCMC for Machine Learning. There are also some excellent tutorials on the web including Walsh (2004) and Iain Murray's video tutorial (Murray, 2009) for machine learning summer school.

Andrieu, C., DeFreitas, N., Doucet, A., & Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine Learning*, *50*(1), 5–43.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference and perception* (2nd ed.). New York: Springer.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, *57*, 97–109.

Mackay, D. J. C. (2003). *Information theory, inference and learning algorithms*. Cambridge: Cambridge University Press.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A., & Teller, H. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, *21*, 1087–1091.

Metropolis, N., & Ulam, S. (1949). The Monte Carlo method. *Journal of the American Statistical Association*, *44*(247), 335–341.

Murray, I. (2009). *Markov chain Monte Carlo*. http://videolectures.net/mlss09uk_murray_mcmc/. Retrieved 25 July 2010.

Russell, S., & Norvig, P. (2009). *Artificial intelligence: a modern approach* (3rd ed.). NJ: Prentice Hall.
Walsh, B. (2004). *Markov chain Monte Carlo and Gibbs sampling*. http://nitro.biosci.arizona.edu/courses/EEB581-2004/handouts/Gibbs.pdf. Retrieved 25 July 2010.

Markov Decision Processes

William Uther
NICTA and the University of New South Wales

Synonyms

Policy search

Definition

A *Markov Decision Process* (MDP) is a discrete, stochastic, and generally finite model of a system to which some external control can be applied. Originally developed in the Operations Research and Statistics communities, MDPs, and their extension to ▶Partially Observable Markov Decision Processes (POMDPs), are now commonly used in the study of ▶reinforcement learning in the Artificial Intelligence and Robotics communities (Bellman, 1957; Bertsekas & Tsitsiklis, 1996; Howard, 1960; Puterman, 1994). When used for reinforcement learning, firstly the parameters of an MDP are learned from data, and then the MDP is processed to choose a behavior.

Formally, an MDP is defined as a tuple: $< \mathcal{S}, \mathcal{A}, T, R >$, where $\mathcal{S}$ is a discrete set of states, $\mathcal{A}$ is a discrete set of actions, $T : \mathcal{S} \times \mathcal{A} \to (\mathcal{S} \to R)$ is a stochastic transition function, and $R : \mathcal{S} \times \mathcal{A} \to R$ specifies the expected reward received for performing the given action in each state.

An MDP carries the *Markov* label because both the transition function, T, and the reward function, R, are Markovian; i.e., they are dependent only upon the current state and action, not previous states and actions. To be a valid transition function, the distribution over the resulting states, $(\mathcal{S} \to R)$, must be a valid probability distribution, i.e., non-negative and totalling 1. Furthermore, the expected rewards must be finite.

The usual reason for specifying an MDP is to find the optimal set of actions, or *policy*, to perform. We formalize the optimality criteria below. Let us first consider how to represent a policy. In its most general form the action, $a \in \mathcal{A}$, indicated by a policy, π, might depend upon the entire history of the agent; $\pi : (\mathcal{S} \times \mathcal{A})^* \times \mathcal{S} \to \mathcal{A}$. However, for each of the common optimality criteria considered below a Markov policy, $\pi : \mathcal{S} \to \mathcal{A}$, will be sufficient. i.e., for every MDP, for each of the optimality criteria below, there exists a Markov policy that performs as well as the best full policy. Similarly, there is no requirement for an MDP that a policy be stochastic or mixed.

Optimality Criteria

Informally, one wants to choose a policy so as to maximise the long term sum of immediate rewards. Unfortunately the naive sum, $\sum_{t=0}^{\infty} r_t$ where r_t is the expected immediate reward received at time t, usually diverges. There are different optimality criteria that can than be used as alternatives.

Finite Horizon The easiest way to make sure that the sum of future expected rewards is bounded is to only consider a fixed, finite time into the future; i.e., find a policy that maximises $\sum_{t=0}^{n} r_t$ for each state.

Infinite Horizon Discounted Rather than limiting the distance we look into the future, another approach is to *discount* rewards we will receive in the future by a multiplicative factor, γ, for each time-step. This can be justified as an inflation rate, as an otherwise unmodelled probability that the simulation ends each time-step, or simply as a mathematical trick to make the criteria converge. Formally we want a policy that maximises $\sum_{t=0}^{\infty} \gamma^t r_t$ for each state.

Average Reward Unfortunately, the infinite horizon discounted optimality criterion adds another parameter to our model: the discount factor. Another approach is to optimize the average reward per time-step, or *gain*, by finding a policy that maximizes $\lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} r_t$ for each state. This is very similar to using *sensitive discount optimality*; finding a policy that maximizes the infinite horizon discounted reward as the discount factor approaches 1, $\lim_{\gamma\to 1} \sum_{t=0}^{\infty} \gamma^t r_t$, for each state.

When maximizing average reward, any finite deviation from the optimal policy will have negligible effect on the average over an infinite timeframe. This can

make the agent "lazy." To counteract this, often a series of increasingly strict optimality criteria are used. The first is the "gain" optimality criterion given above – optimizing the long term average reward. The next is a "bias" optimality which selects from among all gain optimal policies the ones that also optimize transient initial rewards.

Value Determination

For the finite horizon, infinite horizon discounted, or bias optimality criteria, the optimality criteria can be calculated for each state, or for each state-action pair, giving a *value function*. Once found, the value function can then be used to find an optimal policy.

Bellman Equations The standard approach to finding the value function for a policy over an MDP is a dynamic programming approach using a recursive formulation of the optimality criteria. That recursive formulation is known as the Bellman Equation.

There are two, closely related, common forms for a value function; the state value function, $V : \mathcal{S} \to R$ and the state-action value function, $Q : \mathcal{S} \times \mathcal{A} \to R$. For a finite horizon undiscounted optimality criterion with time horizon n and policy π:

$$\begin{aligned} Q_n^\pi(s,a) &= \sum_{t=0}^{n} r_t \\ &= R(s,a) + \mathrm{E}_{s' \in T(s,a)} V_{n-1}^\pi(s') \\ &= R(s,a) + \sum_{s' \in \mathcal{S}} T(s,a)(s') V_{n-1}^\pi(s') \\ V_n^\pi(s) &= Q_n^\pi(s, \pi(s)) \end{aligned}$$

For the infinite horizon discounted case:

$$\begin{aligned} Q^\pi(s,a) &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') V^\pi(s') \\ V^\pi(s) &= Q^\pi(s, \pi(s)) \end{aligned}$$

These equations can be turned into a method for finding the value function by replacing the equality with an assignment:

$$Q^\pi(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') Q^\pi(s', \pi(s'))$$

As long as this update rule is followed infinitely often for each state/action pair, the Q-function will converge.

Prioritised sweeping: Rather than blindly updating each state/action, intelligent choice of where to update will significantly speed convergence. One technique for this is called Prioritized Sweeping (Andre et al., 1997; Moore & Atkeson, 1993).

A priority queue of states is kept. Initially one complete pass of updates over all states is performed, but thereafter states are updated in the order they are pulled from the priority queue. Any time the value of a state, $V^\pi(s)$, changes, the priorities of all states, s', that can reach state s are updated; we update $\{s' \mid T(s', \pi(s'))(s) \neq 0\}$. The priorities are increased by the absolute change in $V^\pi(s)$.

The effect of the priority queue is to focus computation where values are changing rapidly.

Linear Programming Solutions Rather than using the Bellman equation and dynamic programming, an alternative approach is to set up a collection of inequalities and use linear programming to find an optimal value function. In particular if we minimize,

$$\sum_{s \in \mathcal{S}} V^\pi(s)$$

subject to the constraints

$$\forall_{s \in \mathcal{S}}\ 0 \leq V^\pi(s) - \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') V^\pi(s') \right],$$

then the resulting V^π accurately estimates the expected sum of discounted reward.

Bellman Error Minimization A third approach to value determination is similar to the dynamic programming solution above. Rather than replacing the equality in the Bellman equation with an assignment, it turns the equation into an error function and adjusts the Q function to minimise the sum of squared Bellman residuals (Baird, 1995):

$$\mathrm{Residual}(s) = Q^\pi(s,a) - \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') Q^\pi(s', \pi(s')) \right] \mathrm{Err} = \sum_{s \in \mathcal{S}} \mathrm{Residual}(s)^2$$

Control Methods

The previous section gave us a way to obtain a value function for a particular policy, but what we usually need is a good policy, not a value function for the policy we already have. For an optimal policy, for each state:

$$\pi(s) = argmax_{a \in \mathcal{A}} Q^{\pi}(s,a)$$

If a policy, π, is not optimal then its value function can be used to find a better policy, π'. It is common to use the greedy policy for the value function:

$$\pi'(s) \leftarrow argmax_{a \in \mathcal{A}} Q^{\pi}(s,a)$$

This process can be used iteratively to find the optimal policy.

Policy iteration: Policy iteration alternates between value determination and greedy policy updating steps until convergence is achieved. The algorithm starts with a policy, π_1. The value function is calculated for that policy, V^{π_1}. A new policy is then found from that value function, π_2. This alternation between finding the optimal value function for a given policy and then improving the policy continues until convergence. At convergence the policy is optimal.

Value iteration: Rather than explicitly updating the policy, value iteration works directly with the value function. We define an update,

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') \max_{a' \in \mathcal{A}} Q(s',a'),$$

with a maximization step included. As long as this update is performed often enough in each state, Q will converge. Once Q has converged, the greedy policy will be optimal.

Mixed policy iteration: The two previous methods, policy and value iteration, are two extremes of a spectrum. In practice updates to the policy and value function can occur asynchronously as long as the value and policy in each state are updated often enough.

Representations

In the above discussion we have discussed a number of functions, but not discussed how these functions are represented. The default representation is an array or tabular form which has no constraints on the function it can represent. However, the ▶curse of dimensionality suggests that the number of states will, in general, be exponential in the problem size. This can make even a single complete iteration over the state space intractable. One solution is to represent the functions in a more compact form so that they can be updated efficiently. This approach is known as *function approximation*. Here we review some common techniques.

A class of representations is chosen to represent the functions we need to process: e.g., the transition, T, reward, R, Value, V or Q, and/or policy, π, functions. A particular function is selected from the chosen class by a parameter vector, $\boldsymbol{\theta}$.

There are two important questions that must be answered by any scheme using function approximation; does the resulting algorithm converge to a solution, and does the resulting solution bear any useful relationship with the optimal solution?

A simple approach when using a differentiable function to represent the value function is to use a form of ▶temporal difference learning. For a given state, s, and action, a, the Bellman equation is used to calculate a new value, $Q^{\text{new}}(s,a)$, and then $\boldsymbol{\theta}$ is updated to move the value function toward this new value. This gradient based approach usually has a learning rate, $\alpha \in [0,1]$, to adjust the speed of learning.

$$Q^{\text{new}}(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') V^{\text{old}}(s')$$

$$\boldsymbol{\Delta}_{s,a}\boldsymbol{\theta} = \alpha \frac{\partial Q}{\partial \boldsymbol{\theta}} (Q^{\text{new}}(s,a) - Q^{\text{old}}(s,a))$$

This approach is known not to converge in general, although it does converge in some special cases. A similar approach with full Bellman Error minimization will not oscillate, but it may cause the $\boldsymbol{\theta}$ to diverge even as the Bellman residual converges.

Contraction mappings: The first class of function approximators that was shown to converge with the above update, apart from a complete tabular representation, was the class of contraction mappings (Gordon, 1995). Simply put, these are function approximation classes where changing one value by a certain amount changes every other value in the approximator by no more than that amount. For example, linear interpolation and *tile coding* (Tile codings are also known as Cerebellar Motor Action Controllers (CMAC) in early work (Albus, 1981)) are each contraction mappings whereas linear extrapolation is not.

Formally, let $\mathcal{S}$ be a vector space with max norm $||.||_\infty$. A function f is a contraction mapping if,

$$\forall a, b \in \mathcal{S}, ||f(a) - f(b)||_\infty < ||a - b||_\infty$$

The class of function approximations that form contraction mappings includes a number of common approximation techniques including *tile coding*. Tile coding represents a function as a linear combination of basis functions, $\boldsymbol{\varphi}(s, a)$,

$$\hat{Q}(s, a) = \boldsymbol{\theta} \,.\, \varphi(s, a),$$

where the individual elements of φ are binary features on the underlying state.

Linear approximations: The linear combination of basis functions can be extended beyond binary features. This will converge when temporal differencing updates are performed in trajectories through the state space following the policy being evaluated (Tsitsiklis & Van Roy, 1997).

Variable resolution techniques: One technique for representing value functions over large state spaces is use a non-parametric representation. Munos gives a technique that introduces more basis functions for their approximation over time as needed (Munos & Moore, 2001).

Dynamic Bayesian networks: ►Bayesian Networks are an efficient representation of a factored probability distribution. Dynamic Bayesian Networks use the Bayesian Network formalism to represent the transition function, $\mathcal{T}$, in an MDP (Guestrin et al., 2003). The reward and value functions are usually represented with linear approximations. The policy is usually represented implicitly by the value function.

Decision diagrams: Arithmetic Decision Diagrams (ADDs) are a compact way of representing functions from a factored discrete domain to a real range. ADDs can also be efficiently manipulated, with operators for the addition and multiplication of ADDs as well as taking the maximum of two ADDs. As the Bellman equation can be re-written using operators, it is possible to implement mixed policy iteration using this efficient representation (St-Aubin et al., 2000).

Hierarchical representations: ►Hierarchical Reinforcement Learning factors out common substructure in the functions that represent an MDP in order to solve it efficiently. This has been done in many different ways. Dietterich's *MAXQ* hierarchy allowed a prespecified hierarchy to re-use common elements in a value function (Dietterich, 2000). Sutton's *Options* framework focussed on temporal abstraction and re-use of policy elements (Sutton et al., 1998). Moore's *Airports* hierarchy allowed automatic decomposition of a problem where the specific goal could change over time, and so was made part of the state (Moore et al., 1999). Andre's *A-Lisp* system takes the hierarchical representation to an extreme by building in a Turing complete programming language (Andre & Russell, 2002).

Greedy Algorithms Versus Search

In the previous sections the control problem was solved using a greedy policy for a value function. If the value function was approximate, then the resulting policy may be less than optimal. Another approach to improving the policy is to introduce search during execution. Given the current state, the agent conducts a forward search looking for the sequence of actions that produces the best intermediate reward and resulting state value combination.

These searches can be divided into two broad categories: deterministic and stochastic searches. Deterministic searches, such as LAO* (Hansen & Zilberstein, 1998), expand through the state space using the supplied model of the MDP. In contrast stochastic, or Monte-Carlo, approaches sample trajectories from the model and use statistics gathered from those samples to choose a policy (Kocsis & Szepesvári, 2006).

Cross References

►Bayesian Network
►Curse of Dimensionality
►Monte-Carlo Simulation
►Partially Observable Markov Decision Processes
►Reinforcement Learning
►Temporal Difference Learning

Recommended Reading

Albus, J. S. (1981). *Brains, behavior, and robotics*. Peterborough: BYTE. ISBN: 0070009759.

Andre, D., Friedman, N., & Parr, R. (1997). Generalized prioritized sweeping. *Neural and Information Processing Systems*, pp. 1001–1007.

Andre, D., Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*.

Baird, L. C. (1995). Residual algorithms: reinforcement learning with function approximation. In A. Prieditis & S. Russell (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference (ICML95)* (pp. 30–37). San Mateo: Morgan Kaufmann.

Bellman, R. E. (1957). *Dynamic programming*. Princeton: Princeton University Press.

Bertsekas, D. P., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research 13*, 227–303.

Gordon, G. J. (1995). *Stable function approximation in dynamic programming* (Technical report CMU-CS-95-103). School of Computer Science, Carnegie Mellon University.

Guestrin, C., et al. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research, 19*, 399–468.

Hansen, E. A., & Zilberstein, S. (1998). Heuristic search in cyclic AND/OR graphs. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. http://rbr.cs.umass.edu/shlomo/papers/HZaaai98.html

Howard, R. A. (1960). *Dynamic programming and Markov processes*. Cambridge: MIT Press.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. *European Conference on Machine Learning (ECML). Lecture Notes in Computer Science 4212*, Springer, pp. 282–293.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: reinforcement learning with less data and less real time. *Machine Learning, 13*, 103–130.

Moore, A. W., Baird, L., & Pack Kaelbling, L. (1999). Multi-value-functions: efficient automatic action hierarchies for multiple goal MDPs. *International Joint Conference on Artificial Intelligence (IJCAI99)*.

Munos, R., & Moore, A. W. (2001). Variable resolution discretization in optimal control. *Machine Learning, 1*, 1–31.

Puterman, M. L. (1994). Markov decision processes: discrete stochastic dynamic programming. Wiley series in probability and mathematical statistics. Applied probability and statistics section. New York: Wiley. ISBN: 0-471-61977-9.

St-Aubin, R., Hoey, J., & Boutilier, C. (2000). APRICODD: approximate policy construction using decision diagrams. NIPS-2000.

Sutton, R. S., Precup, D., & Singh, S. (1998). Intra-option learning about temporally abstract actions. *Machine Learning: Proceedings of the Fifteenth International Conference (ICML98)*, Morgan Kaufmann, Madison, pp. 556–564.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control, 42*(5), 674–690.

Markov Model

►Markov Process

Markov Net

►Markov Network

Markov Network

Synonyms

Markov net; Markov random field

Definition

A Markov network is a form of undirected ►graphical model for representing multivariate probability distributions.

Cross References

►Graphical Models

Markov Process

Synonyms

Markov chain; Markov model

A stochastic process in which the conditional probability distribution of future states of the process, given the present state and all past states, depends only upon the present state. A process with this property may be called Markovian. The best known Markovian processes are *Markov chains*, also known as *Markov Models*, which are discrete-time series of states with transition probabilities. Markov chains are named after Andrey Markov (1865–1922), who introduced several significant new notions to the concept of stochastic processes. Brownian motion is another well-known phenomenon that, to close approximation, is a Markov process.

Recommended Reading

Meyn, S. P., & Tweedie, R. L. (1993). *Markov chains and stochastic stability*. Springer-Verlag, London

Markov Random Field

►Markov Network

Markovian Decision Rule

Synonyms

Randomized decision rule

Definition

In a ►Markov decision process, a *decision rule*, d_t, determines what *action* to take, based on the history to date at a given *decision epoch* and for any possible *state*. It is *deterministic* if it selects a single member of $A(s)$ with probability 1 for each $s \in S$ and for a given h_t, and it is *randomized* if it selects a member of $A(s)$ at random with probability $q_{d_t(h_t)}(a)$. It is *Markovian* if it depends on h_t only through s_t. That is, $d_t(h_t) = d_t(s_t)$.

Maxent Models

►Maximum Entropy Models for Natural Language Processing

Maximum Entropy Models for Natural Language Processing

Adwait Ratnaparkhi
Yahoo! Labs, Santa Clara
California, USA

Synonyms

Log-linear models; Maxent models; Statistical natural language processing

Definition

The term maximum entropy refers to an optimization framework in which the goal is to find the probability model that maximizes entropy over the set of models that are consistent with the observed evidence.

The information-theoretic notion of entropy is a way to quantify the uncertainty of a probability model; higher entropy corresponds to more uncertainty in the probability distribution. The rationale for choosing the maximum entropy model – from the set of models that meet the evidence – is that any other model assumes evidence that has not been observed (Jaynes, 1957).

In most natural language processing problems, observed evidence takes the form of co-occurrence counts between some prediction of interest and some linguistic context of interest. These counts are derived from a large number of linguistically annotated examples, known as a corpus. For example, the frequency in a large corpus with which the word *that* co-occurs with the tag corresponding to determiner, or *DET*, is a piece of observed evidence. A probability model is consistent with the observed evidence if its calculated estimates of the co-occurrence counts agree with the observed counts in the corpus.

The goal of the maximum entropy framework is to find a model that is consistent with the co-occurrence counts, but is otherwise maximally uncertain. It provides a way to combine many pieces of evidence into a single probability model. An iterative parameter estimation procedure is usually necessary to find the maximum entropy probability model.

Motivation and Background

The early 1990s saw a resurgence in the use of statistical methods for natural language processing (Church & Mercer, 1993). In particular, the IBM TJ Watson Research Center was a prominent advocate in this field for statistical methods such as the maximum entropy framework. Language modeling for speech recognition (Lau, Rosenfeld, & Roukos, 1993) and machine translation (Berger, Della Pietra, & Della Pietra, 1996) were among the early applications of this framework.

Structure of Learning System

The goal of a typical natural language processing application is to automatically produce linguistically motivated categories or structures over freely occurring text. In statistically based approaches, it is convenient to produce the categories with a conditional probability model p such that $p(a|b)$ is the probability of seeing a prediction of interest a (e.g., a part-of-speech tag) given a linguistic context of interest b (e.g., a word).

The maximum entropy framework discussed here follows the machine learning approach to NLP, which assumes the existence of a large corpus of linguistically annotated examples. This annotated corpus is used to create a training set, which in turn is used to estimate the probability model p.

Representing Evidence

Evidence for the maximum entropy model is derived from the training set. The training set is a list of (prediction, linguistic context) pairs that are generated from the annotated data. However, in practice, we do not record the entire linguistic context. Instead, linguistically motivated Boolean-valued questions reduce the entire linguistic context to a vector of question identifiers. Therefore, each training sample looks like:

Prediction	Question vector
a	$q_1 \ldots q_n$

where a is the prediction and where $q_1 \ldots q_n$ is a vector of questions that answered true for the linguistic context corresponding to this training sample. The questions must be designed by the experimenter in advance, and are specifically designed for the annotated data and the problem space.

In the framework discussed here, any piece of evidence is represented with a *feature*. A feature f_j correlates a prediction a with an aspect of a linguistic context b, captured by some question:

$$f_j(a,b) = \begin{cases} 1 & \text{if } a = x \text{ and } q(b) = \text{true} \\ 0 & \text{otherwise} \end{cases}$$

Combining the Evidence

The maximum entropy framework provides a way to combine all the features into a probability model. In the conditional maximum entropy formulation (Berger et al., 1996), the desired model p^* is given by:

$$P = \{p | E_p f_j = E_{\tilde{p}} f_j, j = 1 \ldots k\}, \quad (1)$$
$$H(p) = -\sum_{a,b} \tilde{p}(b) p(a|b) \log p(a|b),$$
$$p^* = argmax_{p \in P} H(p),$$

where $H(p)$ is the conditional entropy of p, $\tilde{p}(b)$ is the observed probability of the linguistic context b in the training set, and P is the set of models that are consistent with the observed data. A model p is consistent if its own feature expectation $E_p f_j$ is equal to the observed feature expectation $E_{\tilde{p}} f_j$, for all $j = 1 \ldots k$ features. $E_{\tilde{p}} f_j$ can be interpreted as the observed count of f_j in the training sample, normalized by the training sample size. Both are defined as follows:

$$E_p f_j = \sum_{a,b} \tilde{p}(b) p(a|b) f_j(a,b),$$
$$E_{\tilde{p}} f_j = \sum_{a,b} \tilde{p}(a,b) f_j(a,b).$$

According to the maximum entropy framework, the optimal model p^* is the most uncertain model among those that satisfy the feature constraints. It is possible to show that the form of the optimal model must be log-linear:

$$p^*(a|b) = \frac{1}{Z(b)} \prod_{j=1 \ldots k} \alpha_j^{f_j(a,b)}, \quad (2)$$

$$Z(b) = \sum_{a'} \prod_{j=1 \ldots k} \alpha_j^{f_j(a',b)}.$$

Here $Z(b)$ is a normalization factor, and $\alpha_j > 0$. Each model parameter α_j can be viewed as the "strength" of its corresponding feature f_j; the conditional probability is the normalized product of the feature weights of the active features.

Relationship to Maximum Likelihood

The maximum entropy framework described here has an alternate interpretation under the more commonly used technique of maximum likelihood estimation.

$$Q = \left\{ p | p(a|b) = \frac{1}{Z(b)} \prod_{j=1 \ldots k} \alpha_j^{f_j(a,b)} \right\},$$
$$L(p) = \sum_{a,b} \tilde{p}(a,b) \log p(a|b),$$
$$q^* = argmax_{p \in Q} L(p).$$

Here Q is the set of models of form (2), $\tilde{p}(a,b)$ is the observed probability of prediction a together with linguistic context b, $L(p)$ is the log-likelihood of the training set, and q^* is the maximum likelihood model. It can be shown that $p^* = q^*$; maximum likelihood estimation for models of the form (2) gives the same answer as maximum entropy estimation over the constraints on feature counts (1). The difference between

approaches is that the maximum likelihood approach assumes the form of the model, whereas the maximum entropy approach assumes the constraints on feature expectations, and *derives* the models form.

Parameter Estimation

The Generalized Iterative Scaling (GIS) algorithm (Darroch & Ratcliff, 1972) is the easiest way to estimate the parameters for this kind of model. The iterative updates are given below:

$$\alpha_j^{(0)} = 1,$$

$$\alpha_j^{(n)} = \alpha_j^{(n-1)} \left[\frac{E_{\tilde{p}} f_j}{E_p f_j} \right]^{\frac{1}{C}}.$$

GIS requires the use of a "correction" feature g and constant $C > 0$, which are defined so that $g(a,b) = C - \sum_{j=1\ldots k} f_j(a,b)$ for any (a,b) pair in the training set. Normally, the correction feature g must be trained in the model along with the k original features, although Curran and Clark (2003) show that GIS can converge even without the correction feature. The number of iterations needed to achieve convergence depends on certain aspects of the data, such as the training sample size and the feature set size, and is typically tuned for the problem at hand.

Other algorithms for parameter estimation include the Improved Iterative Scaling (Berger et al., 1996) algorithm and the Sequential Conditional GIS (Goodman, 2002) algorithm. The list given here is not complete; many other numerical algorithms can be applied to maximum entropy parameter estimation, see (Malouf, 2002) for a comparison.

It is usually difficult to assess the reliability of features that occur infrequently in the training set, especially those that occur only once. When the parameters are trained from low frequency feature counts, maximum entropy models – as well as many other statistical learning techniques – have a tendency to "overfit" the training data. In this case, performance on training data appears very high, but performance on the intended test data usually suffers. *Smoothing* algorithms are designed to alleviate this problem for statistical models; some smoothing techniques for maximum entropy models are reviewed in (Chen & Rosenfeld, 1999).

Applications

This framework has been used as a generic machine learning toolkit for many problems in natural language processing. Like other generic machine learning techniques, the core of the maximum entropy framework is invariant across different problem spaces. However, some information is specific to each problem space:

1. Predictions: The space of predictions for this model.
2. Questions: The space of questions for this model.
3. Feature Selection: Any possible (question, prediction) pair can be used as a feature. In complex models, only a small subset of all the possible features are used in a model. The feature selection strategy specifies how to choose the subset.

For a given application, it suffices to give the above three pieces of information to fully specify a maximum entropy probability model.

Part-of-Speech Tagging

Part-of-speech tagging is a well-known task in computational linguistics in which the goal is to disambiguate the part-of-speech of all the words in a given sentence. For example, it can be non trivial for a computer to disambiguate the part-of-speech of the word *flies* in the following famous examples:

- Fruit *flies* like a banana
- Time *flies* like an arrow

The word *flies* behaves like a noun in the first case, and like a *verb* in the second case. In the machine learning approach to this problem, co-occurrence statistics of tags and words in the linguistic context are used to create a predictive model for part-of-speech tags.

The computational linguistics community has created annotated corpora to help build and test algorithms for tagging. One such corpus, known as the Penn treebank (Marcus, Santorini, & Marcinkiewicz, 1994), has been used extensively by machine learning and statistical NLP practitioners for problems like tagging. In this corpus, roughly 1M words from the Wall Street Journal have manually been assigned part-of-speech tags. This corpus can be converted into a set of training samples, which in turn can be used to train a maximum entropy model.

Model Specification For tagging, the goal is a maximum entropy model p that will produce a probability of seeing a tag at position i, given the linguistic context of

the ith word, the surrounding words, and the previously predicted tags, written as $p(t_i|t_{i-1}\ldots t_1, w_1\ldots w_n)$. The intent is to use the model left-to-right, one word at a time. The maximum entropy model for tagging (Ratnaparkhi, 1996) is specified as:

1. Predictions: The 45 part-of-speech tags of the Penn treebank
2. Questions: Listed below are the questions and question patterns. A question pattern has a placeholder variable (e.g., X, Y) that is instantiated by scanning the annotated corpus for examples in which the patterns match. Let i denote the position of the current word in the sentence, and let w_i and t_i denote the word and tag at position i, respectively.
   - Does $w_i = X$?
   - Does $w_{i-1} = X$?
   - Does $w_{i-2} = X$?
   - Does $w_{i+1} = X$?
   - Does $w_{i+2} = X$?
   - Does $t_{i-1} = X$?
   - Does $t_{i-1}t_{i-2} = X, Y$?
   - For words that occur fewer than 5 times in the training set:
     - Are the first K (for $K \le 4$) characters $X_1 \ldots X_K$?
     - Are the last K (for $K \le 4$) characters $X_1 \ldots X_K$?
     - Does the current word contain a number?
     - Does the current word contain a hyphen?
     - Does the current word contain an uppercase character?
3. Feature Selection: Any (question, prediction) pair whose count in the training data is ≥10 is retained as a feature.

While the features for each probability decision could in theory look at the entire linguistic context, they actually only look at a small window of words surrounding the current word, and a small window of tags to the left. Therefore each decision effectively makes the Markov-like assumption given in (3).

$$p(t_i|t_{i-1}\ldots t_1, w_1\ldots w_n) = p(t_i|t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i w_{i+1}w_{i+2}) \quad (3)$$

$$= \frac{\prod_{j=1\ldots k}\alpha_j^{f_j(t_i, t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i w_{i+1}w_{i+2})}}{Z(t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i w_{i+1}w_{i+2})} \quad (4)$$

Equation (4) is the maximum entropy model for tagging. Each conditional probability of a prediction t_i given some context $t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i w_{i+1}w_{i+2}$ is the product of the features that are active for that (prediction, context) pair.

Training Data The training set is created by applying the questions to each word in the training set. For example, when scanning the word *flies* in the sentence "Time flies like an arrow" the training example would be:

Prediction	Question vector
verb	w_i = flies, w_{i-1} = Time, w_{i-2} = *bd*,
	w_{i+1} = like, w_{i+2} = an,
	t_{i-1} = *noun*, $t_{i-1}t_{i-2}$ = *noun*, *bd*

Here *bd* is a special symbol for boundary. The tags have been simplified for this example; the actual tags in the Penn treebank are more fine-grained than *noun* and *verb*.

Hundreds of thousands of training samples are used to create candidate features. Any possible (prediction, question) pair that occurs in training data is a candidate feature. The feature selection strategy is a way to eliminate unreliable or noisy features from the candidate set. For the part-of-speech model described here, a simple frequency threshold is used to implement feature selection.

Given a selected feature set, the GIS algorithm is then used to find the optimal value for the corresponding α_j parameters. For this application, roughly 50 iterations of GIS sufficed to achieve convergence.

Search for Best Sequence The probability model described thus far will produce a distribution over tags, given a linguistic context including and surrounding the current word. In practice we need to tag entire sentences, which means that the model must produce a *sequence* of tags. Tagging is typically performed left-to-right, so that each decision has the left context of previously predicted tags. The probability of the best tag sequence for an n-word sentence is factored as:

$$p(t_1\ldots t_n|w_1\ldots w_n) = \prod_{i=1\ldots n} p(t_i|t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i w_{i+1}w_{i+2}).$$

The desired tag sequence is the one with the highest conditional sequence probability:

$$t_1^* \ldots t_n^* = arg_{t_1 \ldots t_n} maxp(t_1 \ldots t_n | w_1 \ldots w_n).$$

A dynamic programming procedure known as the Viterbi algorithm is typically used to find the highest probability sequence.

Other NLP Applications

Other NLP applications have used maximum entropy models to predict a wide variety of linguistic structure. The statistical parser in (Ratnaparkhi, 1999) uses separate maximum entropy models for part-of-speech, chunk, and parse structure prediction. The system in (Borthwick, 1999) uses maximum entropy models for named entity detection, while the system in (Ittycheriah, Franz, Zhu, & Ratnaparkhi, 2001) uses them as sub-components for both answer type prediction and named entity detection. Typically, such applications do not need to change the core framework, but instead need to modify the meaning of the predictions, questions, and feature selection to suit the intended task of the application.

Future Directions

Conditional random fields (Lafferty, McCallum, & Pereira, 2001), or CRFs, are an alternative to maximum entropy models that address the *label bias* issue. Label bias affects sequence models that predict one element at a time, in which features at a given state (or word, in the case of POS tagging) compete with each other, but do not compete with features at any other state in the sequence. In contrast, a CRF model has a single model for the probability of the entire sequence, and therefore allows global competition of features across the entire sequence. Parameter estimation for CRFs is related to the GIS algorithm used for maximum entropy models. See Sha & Pereira (2003) for an example of CRFs applied to noun phrase chunking.

Recommended Reading

Berger, A. L., Della Pietra, S. A., & Della Pietra, V. J. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, *22*(1), 39–71.

Borthwick, A. (1999). *A maximum entropy approach to named entity recognition*. Unpublished doctoral dissertation, New York University.

Chen, S., & Rosenfeld, R. (1999). *A Gaussian prior for smoothing maximum entropy models* (Tech. Rep. No. CMUCS-99-108). Carnegie Mellon University.

Church, K. W., & Mercer, R. L. (1993). Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, *19*(1), 1–24.

Curran, J., & Clark, S. (2003). Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the 11th annual meeting of the european chapter of the association for computational linguistics (EACL'03)* (pp. 91–98). Budapest, Hungary.

Darroch, J., & Ratcliff, D. (1972). Generalized iterative scaling for log-linear models. *The Annals of Statistics*, *43*(5), 1470–1480.

Goodman, J. (2002). Sequential conditional generalized iterative scaling. In *Proceedings of the Association for Computational Linguistics (ACL)* (pp. 9–16). Philadelphia, Pennsylvania.

Ittycheriah, A., Franz, M., Zhu, W., & Ratnaparkhi, A. (2001). Question answering using maximum-entropy components. In *Proceedings of the North American association for computational linguistics. (NAACL)*, Pittsburgh, Pennsylvania.

Jaynes, E. T. (1957, May). Information theory and statistical mechanics. *Physical Review*, *106*(4), 620–630.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings 18th international conference on machine learning* (pp. 282–289). San Francisco: Morgan Kaufmann.

Lau, R., Rosenfeld, R., & Roukos, S. (1993). Adaptive language modeling using the maximum entropy principle. In *Proceedings of the ARPA Human Language Technology Workshop* (pp. 108–113). San Francisco: Morgan Kaufmann.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Sixth conference on natural language learning (CoNLL)* (pp. 49–55). Taipei, Taiwan.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1994). Building a large annotated corpus of english: "The Penn Treebank". *Computational Linguistics*, *19*(2), 313-330.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In E. Brill & K. Church (Eds.), *Proceedings of the conference on empirical methods in natural language processing* (pp. 133–142). Somerset, NJ: Association for Computational Linguistics.

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, *34*(1–3), 151–175.

Sha, F., & Pereira, F. (2003). Shallow parsing with conditional random fields. In *Proceedings of the human language technology conference (HLT-NAACL)* (pp. 213–220). Edmonton, Canada.

McDiarmid's Inequality

Synonyms

Bounded differences inequality

Definition

McDiarmid's inequality shows how the values of a bounded function of independent random variables

concentrate about its mean. Specifically, suppose $f : \mathcal{X}^n \to R$ satisfies the bounded differences property. That is, for all $i = 1, \ldots, n$ there is a $c_i \geq 0$ such that for all $x_1, \ldots, x_n, x' \in \mathcal{X}$

$$|f(x_1, \ldots, x_n) - f(x_1, \ldots, x_{i-1}, x', x_{i+1}, \ldots, x_n)| \leq c_i.$$

If $\mathbf{X} = (X_1, \ldots, X_n) \in \mathcal{X}^n$ is a random variable drawn according to P^n and $\mu = E_{P^n}[f(\mathbf{X})]$ then for all $\epsilon > 0$

$$P^n\left(f(\mathbf{X}) - \mu \geq \epsilon\right) \leq \exp\left(\frac{2\epsilon^2}{\sum_{i=1}^n c_i^2}\right).$$

McDiarmid's is a generalization of Hoeffding's inequality, which can be obtained by assuming $\mathcal{X} = [a, b]$ and choosing $f(\mathbf{X}) = \sum_{i=1}^n X_i$. When applied to empirical risks this inequality forms the basis of many ►generalization bounds.

MCMC

►Markov Chain Monte Carlo

MDL

►Minimum Description Length Principle

Mean Absolute Deviation

►Mean Absolute Error

Mean Absolute Error

Synonyms

Absolute error loss; Mean absolute deviation; Mean error

Definition

Mean Absolute Error is a ►model evaluation metric used with ►regression models. The mean absolute error of a model with respect to a ►test set is the mean of the absolute values of the individual prediction errors on over all ►instances in the ►test set. Each prediction error is the difference between the true value and the predicted value for the instance.

$$mae = \frac{\sum_{i=1}^n abs\left(y_i - \lambda(x_i)\right)}{n}$$

where y_i is the true target value for test instance x_i, $\lambda(x_i)$ is the predicted target value for test instance x_i, and n is the number of test instances.

Cross References

►Mean Squared Error

Mean Error

►Mean Absolute Error

Mean Shift

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

Mean shift (Comaniciu & Meer, 2002) is a nonparametric algorithm for ►partitional clustering which does not require specifying the number of clusters, and can form any shape of clusters.

Given n data points $x_i, i = 1, \ldots, n$, in the d-dimensional space R^d, the multivariate kernel density estimator obtained with kernel $K(x)$ and window radius h is given by

$$f(x) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right). \quad (1)$$

Given the gradient of the density estimator, the mean shift is defined as the difference between the weighted (using the kernel as weights) mean and x, the center of the kernel,

$$m_h(x) = \frac{\sum_{i=1}^n x_i g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)} - x. \quad (2)$$

The mean shift vector is proportional to the normalized density gradient estimate, and thus points to the

direction of the maximum increase in the density. By successively computing the mean shift vector and translating the kernel (window) by the vector, the mean shift procedure can guarantee converging at a nearby point where the gradient of density function is zero.

Recommended Reading

Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions of the Pattern Analysis and Machine Intelligence, 24*(5), 603–619.

Mean Squared Error

Synonyms

Quadratic loss; Squared error loss

Definition

Mean Squared Error is a ►model evaluation metric often used with ►regression models. The mean squared error of a model with respect to a ►test set is the mean of the squared prediction errors over all ►instances in the ►test set. The prediction error is the difference between the true value and the predicted value for an instance.

$$mse = \frac{\sum_{i=1}^{n} (y_i - \lambda(x_i))^2}{n}$$

where y_i is the true target value for test instance x_i, $\lambda(x_i)$ is the predicted target value for test instance x_i, and n is the number of test instances.

Cross References

►Mean Absolute Error

Measurement Scales

Ying Yang
Australian Taxation Office, Box Hill, Australia

Definition

Turning to the authority of introductory statistical textbooks (Bluman, 1992; Samuels & Witmer, 1999), there are two parallel ways to classify data into different types. Data can be classified into either ►categorical or ►numeric. Data can also be classified into different levels of ►measurement scales.

There are two parallel ways to classify data into different types. Data can be classified into either categorical or numeric. Data can also be classified into different *levels of* measurement scales.

Categorical versus Numeric

Variables can be classified as either categorical or numeric. **Categorical variables**, also often referred to as **qualitative variables**, are variables that can be placed into distinct categories according to some characteristics. Categorical variables sometimes can be arrayed in a meaningful rank order. But no arithmetic operations can be applied to them. Examples of categorical variables are

- Gender of a fish: male and female
- Student evaluation: fail, pass, good, and excellent

Numeric variables, also often referred to as *quantitative variables*, are numerical in nature. They can be ranked in order. They can also have meaningful arithmetic operations. Numeric variables can be further classified into two groups, discrete or continuous.

A *discrete variable* assumes values that can be counted. The variable cannot assume all values on the number line within its value range. An example of a discrete variable is *the number of children in a family*.

A *continuous variable* can assume all values on the number line within the value range. The values are obtained by measuring. An example of a continuous variable is *Fahrenheit temperature*.

Levels of Measurement Scales

In addition to being classified as either categorical or numeric, variables can also be classified by how they are categorized, counted, or measured. This type of classification uses measurement scales, and four common types of scales are used: nominal, ordinal, interval, and ratio.

The *nominal* level of measurement scales classifies data into mutually exclusive (nonoverlapping), exhaustive categories in which no order or ranking can be imposed on the data. An example of a nominal variable is *gender of a fish*: male and female.

The *ordinal* level of measurement scales classifies data into categories that can be ranked. However, the differences between the ranks cannot be calculated by arithmetic. An example of an ordinal variable is *student*

evaluation, with values fail, pass, good, and excellent. It is meaningful to say that the student evaluation of pass ranks is higher than that of fail. It is not meaningful in the same way to say that the gender female ranks higher than the gender male.

The **interval** level of measurement scales ranks the data, and the differences between units of measure can be calculated by arithmetic. However, *zero* in the interval level of measurement means neither "nil" nor "nothing" as *zero* in arithmetic means. An example of an interval variable is *Fahrenheit temperature*. It is meaningful to say that the temperature A is two points higher than the temperature B. It is not meaningful in the same way to say that the student evaluation of pass is two points higher than that of fail. Besides, 0°F does not mean the absence of heat.

The **ratio** level of measurement scales possesses all the characteristics of interval measurement, and there exists a *zero* that, the same as arithmetic *zero*, means "nil" or "nothing." In consequence, true ratios exist between different units of measure. An example of a ratio variable is *number of children in a family*. It is meaningful to say that the number of children in the family A is twice that of the family B. It is not meaningful in the same way to say that the Fahrenheit temperature A is twice that of B.

The nominal level is the lowest level of measurement scales. It is the least powerful in terms of including data information. The ordinal level is higher. The interval level is even higher. The ratio level is the highest level. Any data conversion from a higher level of measurement scales to a lower level of measurement scales, such as ►discretization, will lose information. Table 1 gives a summary of the characteristics of different levels of measurement scales.

Measurement Scales. Table 1 Characteristics of different levels of measurement scales

Level	Ranking?	Arithmetic Operation?	Arithmetic Zero?
Nominal	No	No	No
Ordinal	Yes	No	No
Interval	Yes	Yes	No
Ratio	Yes	Yes	Yes

Summary

In summary, the following taxonomy applies to variable types:

- Categorical (qualitative) variables:
  - Nominal
  - Ordinal
- Numeric (quantitative) variables:
  - Interval, either discrete or continuous
  - Ratio, either discrete or continuous

Recommended Reading

Bluman, A. G. (1992). *Elementary statistics: A step by step approach*. Wm. C. Brown Publishers Dubuque, Iowa, USA.

Samuels, M. L. & Witmer, J. A. (1999). *Statistics for the life sciences* (2nd ed.). Upper Saddle River, NJ: Prentice-Hall Publishers, USA.

Medicine: Applications of Machine Learning

Katharina Morik
Technische Universität Dortmund, Dortmund, Germany

Motivation

Health care has been an important issue in computer science since the 1960s. In addition to databases storing patient records, library resources (e.g., PubMed, a service of the U.S. National Library of Medicine that includes over 16 million citations from journals for biomedical articles back to the 1950s), administrative and financial systems, more sophisticated support of health care has been the aim of artificial intelligence (AI) from the very beginning on. Starting with expert systems which abstract laboratory findings and other vital parameters of a patient before they heuristically classify the patient into one of the modeled diagnoses (Shortliffe, 1976), knowledge acquisition was discovered to be the bottleneck of systems for the automatic medical diagnosis. Machine learning came into play as a means of knowledge acquisition. Learning rules for (medical) expert systems focused on the heuristic classification step within expert systems. Given conveniently abstracted measurements of the patient's state, the classification was learned in terms of rules or ►decision

trees. Since the early days, the use of machine learning for health care progressed in two ways:

- The abstraction of measurements of a patient's vital parameters is a learning task in its own right. Diverse kinds of data are to be handled: laboratory data, online measurements at the bedside, x-rays or other imaging data, genetic data,... Machine learning is confronted with a *diversity of representations* for the examples.
- Diagnosis is just one task in which physicians are to be supported. There are many more tasks which machine learning can ease. In intensive care, the addressee of the learning results can be a machine, e.g., the respirator. Financing health care and planning the medical resources (e.g., for a predicted epidemia) are yet another important issue. Machine learning is placed in a *diversity of medical tasks*.

The urgent need for sophisticated support of health care follows from reports which estimate up to 100,000 deaths in the USA each year due to medical error (Kohn, Corrigan, & Donaldson, 2000).

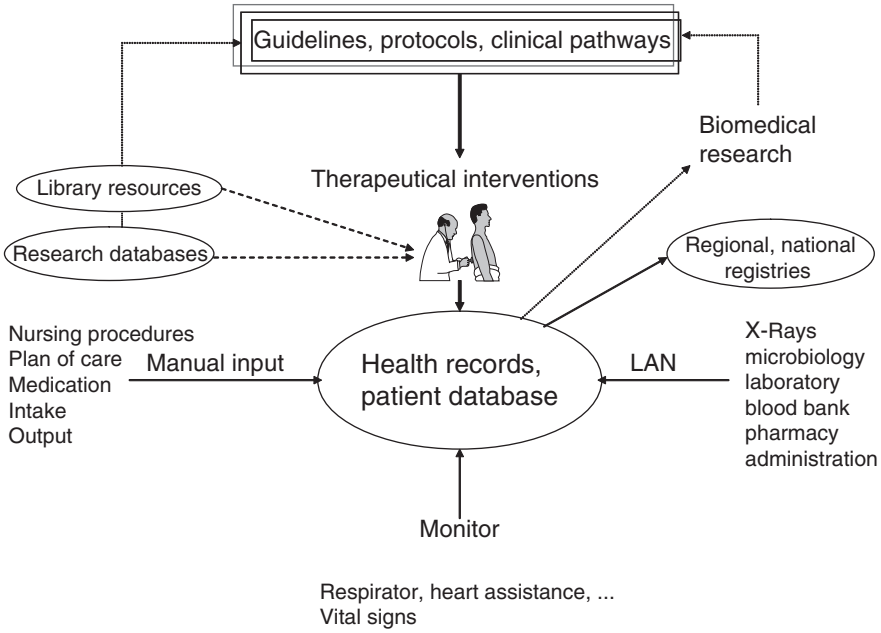
Structure of the Problem

The overall picture of the medical procedures shows the kinds of data and how they are entered into the database of health records (a synonym is "patient database.") A monitoring system is given in intensive care units, which acquires ▶time series from minute measurements. The observations at the bedside are entered manually into the system. The information from the hospital is entered via a local area network. The physician accesses information from libraries and research databases (dashed lines). Libraries, research databases, and biomedical research also influence the development of guidelines, protocols, and clinical pathways (dotted lines). Guidelines are rather abstract. Protocols of certain actions are integrated to become a clinical pathway which is a plan of both diagnostic and therapeutical actions for a typical patient with a specific diagnosis. The bold arrow shows the intended high-quality therapy. Guidelines and protocols promote evidence-based practices, reduce inter-clinician practice variations and support decision-making in patient care while constraining the costs of care. Computerized protocols can be generated based on guidelines. They have been proved useful in improving the quality and consistency of healthcare but the protocol development process is time-consuming (Ten Teije, Lucas, & Miksch, 2006). This is where machine learning offers support. Usually, ontologies (e.g., in description logic) or other knowledge-based techniques (in medicine-specific formats like the Arden Syntax, GuideLine Interchange Format (GLIF), PROforma, Asbru, and EON) are used to support the development of protocols (de Clercq, Blomb, Korstenb, & Hasman, 2004). By contrast, machine learning induces the current practices and their outcome from the health records (Smith, Doctor, Meyer, Kalet & Philips, 2009). To reflect such use of Machine Learning, the bold arrows of the picture would need to be turned the other way around, protocols are learned from the data or evaluated based on the data. All (reversed) arrows mark possible applications of machine learning.

Diversity of Representations

The overall health record of a patient includes several types of data, not all of them are digital.

- Laboratory data consist of attributes almost always with numerical values, sometimes with discrete ordinal values, sometimes just binary values like "positive," "negative."
- Plain text states anamneses, diagnosis, and observations. From the text, key words can be transformed into attributes for machine learning.
- Online measurements at the bedside are time series. They are analyzed in order to find level changes or trends (Gather, Schettlinger, & Fried, 2006) and alarm the physician (Sieben & Gather, 2007). In order to exploit the time series for further learning tasks, they often are abstracted (e.g., Bellazzi, Larizza, Magni, & Bellazi (2002)). Recently, online measurements from body sensors have raised attention in the context of monitoring patients at home (Amft & Tröster, 2008).
- Sequences can also be considered time series, but the measurements are not equidistant and not restricted to numerical values. Examples are data gathered at doctors' visits and long-term patient observations.
- X-rays or other imaging data (e.g., ultrasound imaging or more novel molecular imaging techniques like positron emission tomography, magnetic resonance imaging, or computer tomography) cannot be



analyzed directly by machine learning algorithms. They require the extraction of features. It has been shown that the adequate extraction of features is more important than the selection of the best suited learning algorithm (Mavroforakis, Georgiou, Dimitropoulos, Cavouras, & Theodoridis, 2006). The number of extracted features can become quite large. For instance, from 1,619 images of skin lesion, each 752 × 582 pixels, 107 features were extracted in order to detect melanoma using diverse learning algorithms (Dreiseitl et al., 2001). Hence, feature selection is also an important task in medical applications (Lucaces, Taboada, Albaiceta, Domingues, Enriques & Bahamonde, 2009; Withayachumnankul et al., 2006). Often, different techniques are applied to gather data for the detection of the same disease. For instance, glaucoma detection uses standard automated perimetry or scanning laser or Heidelberg Retina Tomograph or stratus optical coherence tomography. It is not yet clear how important the choice among measurement types (devices) is with respect to feature extraction and machine learning.

- Tissue and blood: In vitro "data" also belong to health records. Immediately after biopsy or surgery, the tissue is transferred to the pathology department. After the pathologist has taken the sample needed for proper diagnosis, a representative tissue sample will be snap frozen and stored in liquid nitrogen or at −80∘C. Also blood cells are stored in a blood bank. From the specimen, the RNA is extracted and the so-called microarrays of gene expressions are developed and then scaled. The huge prognostic value of gene expression in patients with breast cancer has been shown by van't Veer et al. (2002). Genome research aims at revealing the impact of gene regulation and protein expression-regulation (taking into account the regulation of protein synthesis, protein ubiquitination, and post-translational modification) on, e.g., cancer diagnosis and response to therapies. Machine learning, particularly clustering, frequent itemset mining, and classification have been applied successfully (see ►learning from gene expression microarray data).

In addition to patient records, there are knowledge bases describing particular diseases or computerized protocols for particular therapies.

Medical Tasks

Diagnosis and Medication

Diagnosis is primarily a classification task. Given the description of the patient's state and a set of diseases, the learning algorithm outputs the classification

into one of the classes. If physicians want to inspect the learned classifier, logic-based algorithms are preferred. Decision trees and the conceptual clustering algorithm AQ were used to diagnose breast cancer from nine abstracted descriptions like `tumor size`: 0 – 4, 5 – 9, ⋯, 50 – 54, 55 – 59 (Cestnik, Kononenko, & Bratko, 1987; Michalski, Mozetic, Hong, & Lavrac, 1986).

►Bayesian methods were used to classify, e.g., diseases of the lymph node. Based on the examination of the extracted tissue, a pathologist enters the description. The Bayesian network (BN) outputs not only just one diagnosis, but the conditional probabilities for the diseases (Heckerman, 1990). In particular, diagnosis for rather vague symptoms such as abdominal pain or lower back pain is well supported by BNs (McNaught, Clifford, Vaughn, Foggs, & Foy, 2001). BNs are capable of incorporating given expert knowledge as priors. In order to combine textbook knowledge with empirical data, electronic literature was transformed into priors for BN structures. Then, from health records, the BN was learned as a model of ovarian tumors (Antal, Fannes, Timmerman, Moreau, & De Moor, 2004).

►Inductive logic programming (ILP) also allows to take into account background knowledge. This was used for an enhanced learning of medical diagnostic rules (Lavrac, Dzeroski, Prinat, & Krizman, 1993). The identification of glaucomatous eyes was effectively learned by ILP (Mizoguchi, Ohwada, Daidoji, & Shirato, 1997). One advantage of ILP is that the learned logic clauses can easily be integrated into a knowledge-based system and, hence, become operational for clinical practice.

Since some tests which deliver information about the patient's state can be costly – both, financially and in terms of a risk for the patient – ►cost-sensitive learning may be applied.

Since the error of classifying a patient as ill where he or she is not (false positives) is less harmful than classifying a patient as healthy where he or she is not (false negatives), the evaluation of the learning result most often is used in a biased way. The evaluation can be summarized in Table 1.

Precision is the proportion $\frac{A}{A+B}$, and *recall* is the proportion $\frac{A}{A+C}$. *Sensitivity* is synonymous to recall. In medical applications, sensitivity is balanced with respect to *specificity* being the proportion $\frac{B}{B+D}$ (synonym *false positives rate*). The analysis of the Receiver Operator Characteristic (ROC) allows to evaluate learning according to sensitivity and specificity (see ►ROC analysis).

Medicine: Applications of Machine Learning. Table 1
Evaluation measures

	True +	False –
Predicted +	A	B
Predicted –	C	D

If not the understandability but only sensitivity and specificity are important, numerical learning algorithms are used to classify the patient's data. In particular, if the patient's state is described by numerical features, no discretization is necessary for numerical learners as is needed for the logic-based ones. Multilayer perceptrons (see ►Neural Networks), ►support vector machines (SVM), ►mixtures of Gaussians, and mixture of generalized Gaussian classifiers were trained on the numerical data of 189 normal eyes and 156 glaucomatous eyes (Goldbaum et al., 2002). The numerical description of the visual field is given by standard automated threshold perimetry. The medical standard procedure to interpret the visual field is to derive global indices. The authors compared performance of the classifiers with these global indices, using the area under the ROC curve. Two human experts were judged against the machine classifiers and the global indices by plotting their sensitivity–specificity pairs. The mixture of Gaussian had the greatest area under the ROC curve of the machine classifiers, and human experts were not better at classifying visual fields than the machine classifiers or the global indices.

Other approaches to glaucoma detection use different features describing the patient's state (Zangwill et al., 2004) or other numerical learners, e.g., ►logistic regression (Huang, Chen, & Hung, 2006). For testing the learning from numerical attributes, the UCI Machine Learning Repository offers the arrhythmia database. The aim is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups. About 279 attributes are given, 206 of them being numerical ones.

As has been shown in an application to intensive care, medication can be transformed into a set of classification tasks (Morik, Imhoff, Brockhausen, Joachims,

& Gather, 2000). Given measurements of eight vital signs, a decision is made for each of six drugs, whether to increase or to decrease it. This gives a set of classification tasks, which the ►SVM learned. Depending on the drug, the accuracy ranged from 81.3% with 2.5 standard error to 86.9% with 7 standard error. Additionally, on 41 cases, the SVM decision was compared with an expert's decisions when confronted with the same data. In 32 cases the expert chose the same direction of change as did the learned decision function. In 34 cases the learned decision was equal to the actual therapy. Another set of classification tasks were to decide every minute whether to increase, decrease, or leave the doses as it is. Again, each of these classifiers was learned by the SVM. From 1,319 examples decision functions were learned and tested on 473 examples. For training, an unbalanced cost function was used. The SVM cost factor for error was chosen according to $\frac{C_+}{C_-} = \frac{number\ of\ negative\ examples}{number\ of\ positive\ examples}$. The results again differed depending on the drug. For adrenaline, 79% of the test cases were equally handled by the physician and the decision function. For adrenaline as well as for dobutamine, only in 1.5% of the test cases the learned rule recommended the opposite direction of change. Again, a blind test with an expert showed that the learned recommendations' deviation from actual therapy was comparable to that of the human expert. Combining the two sets of classifications, for each minute and each patient, the support vector machine's decision function outputs a recommendation of treatment (Morik, Imhoff, Brockhausen, Joachims, & Gather, 2000).

Prognosis and Quality of Care Assessment

Prognosis or outcome prediction is important for the evaluation of the quality of care provided. The standard statistical models use only a small set of covariates and a score variable, which indicates the severity of the illness. Machine learning may also rely on the aggregated score features, but is in addition capable of handling the features underlying the scores. Given health records of patients including the therapy, machine learning is to predict the outcome of care, e.g., classifies into mortal or surviving cases. The prediction of breast cancer survivability has been tested on a very large database comparing three learning methods (Delen, Walker, & Kadam, 2004). The results indicated that decision trees (here: C5) result in the best predictor with 93.6% accuracy on the holdout sample (this prediction accuracy is better than any reported in the literature), artificial neural networks came out to be the second with 91.2% accuracy, and the ►logistic regression models came out to be the worst of the three with 89.2% accuracy.

Prediction of survival is a hard task for patients with serious stroke, because there is a long-term risk after the stay at the hospital. The scoring schemes (e.g., the Glasgow coma scale and the Ranking score) are not sufficient for predicting the outcome. In a data situation where 29 attributes (or features) were given for only 327 patient records, BNs were learned and compared with a handmade causal network. The results were encouraging – as soon as more electronic health records become available, the BNs will become closer to medical knowledge. Moreover, the discovery of relations on the basis of empirical data may enhance medical knowledge (Wu, Lucas, Kerr, & Dijkhuisen, 2001).

Carcinogenesis prediction was performed using ILP methods. As has become usual with cancer diagnosis and prognosis, there is a close link with microbiology (Srinivasan, Muggleton, King, & Sternberg, 1994)(see Learning from gene expression microarray data).

Prognosis need not be restricted to mortality rates. In general, it is a means of quality assessment of clinical treatments. For instance, hemodialysis services have been assessed through temporal data mining by Bellazzi et al. (2002).

Finding subgroups of patients with devious reactions to a therapy might lead to a better understanding of a certain medical process (Atzmueller, Baumeister, Hensing, Richter, & Puppe, 2005). While the before mentioned study aims at an enhanced expert – system interaction, a Dutch study aims at a more precise modeling of prognoses (Abu-Hanna & Lucas, 2001). In an extensive study for eight different hospitals and 7,803 patients, two different models were combined: one for determining the subgroups and the other for building a model for each subgroup. For the prognoses of patients in an intensive care unit, subgroups have been detected using decision trees. The decision tree was trained to classify patients into the survival class and the mortality class on the basis of the nonaggregated features underlying the illness score. The leaves of the

tree become subgroups. These are then used for training a logistic regression model of mortality based on the aggregated features.

Verification and Validation

Verification is the process of testing a model against a specification. In medicine, this often means to check clinical practice against expert protocols, or to check an actual diagnosis against one derived from textbook knowledge. Since many logic-based machine learning algorithms consist of a generalization and a specialization step, they can be used for verification. Generalization delivers rules from clinical data which can then be compared with given expert rules (protocols). Specialization is triggered by facts that contradict a learning hypothesis. Hence, using an expert rule as hypothesis, the learning algorithm counts the contradicting clinical cases and specializes the rule. For an early case study on verification and rule enhancement see, e.g., (Morik, Potamias, Moustakis, & Charissis, 1994). A more recent study compares a given clinical protocol for intensive care with actual therapies at another hospital (Scholz, 2002). Decision trees and association rules have been learned in order to inspect and enhance the knowledge base of a web-based teledermatology system (Ou, West, Lazarescu, & Clay, 2007). While verification means to build the system right, validation means to build the right system. The borderline between verification and validation is fuzzy. On the one hand, medical practice is investigated with respect to the guidelines (verification), on the other hand, the guidelines are enhanced on the basis of medical practice (validation).

Moreover, learned models can be verified with respect to expert knowledge and validated with respect to clinical practice. A study on the hemodynamic monitoring of the critically ill integrated machine learning into a knowledge-based approach to evidence-based medicine. A knowledge base on drug effects was verified using patient records. Only 18% of the observations showed vital signs of patients in the opposite direction than predicted by the knowledge base. Then, the knowledge base was used to validate therapeutical interventions proposed by a learned model. Accuracy measures of a model only reflect how well the learning result fits actual behavior of the physician and not how well it fits the "gold standard." Hence, a proposed intervention should be validated with respect to its effects on the patient. If the known effects push vital signs in the direction of the desired value range, the recommendation is considered sound, otherwise it is rejected. Using past data, the learned model was found to recommend an intervention with the desired effects in 81% of the cases (Morik, Joachims, Imhoff, Brockhausen, & Rüping, 2002).

Intelligent Search in Medical Literature

Intelligent search in the overwhelming number of research publications supplies the information when it is needed. ILP has been successfully put to use for finding relevant medical documents (Dimec, Dzeroski, Todorovski, & Hristovski, 1999). Also the intelligent search in clinical free-text guidelines is an issue (Moskovitch et al., 2006). The techniques for text categorization can be applied to medical texts in the usual way. If the search engine not only labels the overall document but, in addition, phrases within it, the search could become more focused and also deliver paragraphs instead of complete texts. The biomedical challenge for *named entity recognition* requires the automatic extraction and classification of words referring to DNA, RNA, proteins, cell types, and cell lines from texts (Kim, Ohta, Tsuruoka, Tateisi, & Collier, 2004). Even more difficult is the discovery of medical knowledge from texts (Sanchez & Moreno, 2005).

Epidemiology and Outbreak Detection

Understanding the transmission of infectious diseases and forecasting epidemics is an important task, since infections are distributed globally. Statistical approaches to spatio-temporal analysis of scan data are regularly used. There, a grid partitions the map into regions where occurrences of the disease are shown as points. "Hot spot" partitions are those of high density. By contrast, clustering detects hot spot regions depending on the data, hence, the shape of regions is flexible. Taking into account the a priori density of the population, a risk-adjusted nearest neighbor hierarchical clustering discovers "hot spot" regions. Also a risk-adjusted support vector machine with Gaussian kernel has successfully been applied to the problem of detecting regions with infectious disease outbreak. The

discovery of hot spot regions can be exploited for predicting virus activity, if an indicator is known which can easily be observed. For instance, dead crows indicate activity of the West Nile virus. An overview of infectious disease informatics is given by (Zeng, Chen, Lynch, Eidson, & Gotham, 2005).

Machine learning can also contribute to the understanding of the transmission of infectious diseases. A case study on tuberculosis epidemiology uses BNs to identify the distribution of tuberculosis patient attributes. The learning results captured the known statistical relationships. A relational model learned from the database directly using structured statistical models revealed several novel associations (Getoor, Rhee, Koller, & Small, 2004).

Cross References

- ▶Class Imbalance Problem
- ▶Classification
- ▶Classifier Systems
- ▶Cost-Sensitive Learning
- ▶Decision Trees
- ▶Feature Selection
- ▶Inductive Logic Programming
- ▶ROC Analysis
- ▶Support Vector Machine
- ▶Time Series

Recommended Reading

Abu-Hanna, A., & Lucas, P. J. F. (2001). Prognostic models in medicine: AI and statistical approaches [Editorial]. *Methods of Information in Medicine*, *40*(1), 1–5.

Amft, O., & Tröster, G. (2008). Recognition of dietary events using on-body sensors. *Artifical Intelligence in Medicine, 42*(2), 121–136.

Antal, P., Fannes, G., Timmerman, D., Moreau, Y., & De Moor, B. (2004). Using literature and data to learn BNs as clinical models of ovarian tumors. *Artificial Intelligence in Medicine, 30*(3), 257–281.

Atzmueller, M., Baumeister, J., Hensing, A., Richter, E.-J., & Puppe, F. (2005). Subgroup mining for interactive knowledge refinement. In *Artificial intelligence in medicine (AIME)* (pp. 453–462). Berlin/Heidelberg: Springer.

Bellazzi, R., Larizza, C., Magni, P., & Bellazi, R. (2002). Quality assessment of dialysis services through intelligent data analysis and temporal data mining. In *Workshop at the 15th European conference on AI about intelligent data analysis in medicine and pharmacology* (pp. 3–9). Lyon, France.

Cestnik, B., Kononenko, I., & Bratko, I. (1987). ASSISTANT 86: A knowledge-elicitation tool for sophisticated users. In I. Bratko, & N. Lavrac (Eds.), *Progress in machine learning* (pp. 31–45). Wilmslow, GB: Sigma Press.

de Clercq, P. A., Blomb, J. A., Korstenb, H. H., & Hasman, A. (2004). Approaches for creating computer-interpretable guidelines that facilitate decision support. *Artificial Intelligence in Medicine, 31*(1), 1–27.

Delen, D., Walker, G., & Kadam, A. (2004). Predicting breast cancer survivability: A comparison of three data mining methods. *Artificial Intelligence in Medicine, 34*(2), 113–127.

Dimec, B., Dzeroski, S., Todorovski, L., & Hristovski, D. (1999). WWW search engine for slovenian and english medical documents. In *Proceedings of the 15th international congress for medical informatics* (pp. 547–552). Amsterdam: IOS Press.

Dreiseitl, S., Ohn-Machado, L., Kittler, H., Vinterbo, S., Billhardt, H., & Binder, M. (2001). A comparison of machine learning methods for the diagnosis of pigmented skin lesions. *Journal of Biomedical Informatics, 34*, 28–36.

Gather, U., Schettlinger, K., & Fried, R. (2006). Online signal extraction by robust linear regression. *Computational Statistics, 21*(1), 33–51.

Getoor, L., Rhee, J. T., Koller, D., & Small, P. (2004). Understanding tuberculosis epidemiology using structured statistical models. *Artificial Intelligence in Medicine, 30*(3), 233–256.

Goldbaum, M. H., Sample, P. A., Chan, K., Williams, J., Lee, T-W., Blumenthal, E., et al. (2002). Comparing machine learning classifiers for diagnosing glaucoma from standard automated perimetry. *Investigative Ophthalmology and Visual Science, 43*, 162–169.

Heckerman, D. (1990). *Probabilistic similarity networks*. Technical report STAN-CS-1316, Department of Computer Science and Medicine at Stanford.

Huang, M. L., Chen, H. Y., & Hung, P. T. (2006). Analysis of glaucoma diagnosis with automated classifiers using stratus optical coherence tomography. *Optical Quantum Electronics, 37*, 1239–1249.

Kim, J. D., Ohta, T., Tsuruoka, Y., Tateisi, Y., & Collier, N. (2004). Introduction to the bio-entity recognition task at JNLPBA. In N. Collier, P. Ruch, & A. Nazarenko, (Eds.), *Proceedings of the international joint workshop on natural language processing in biomedicine and its applictions* (pp. 70–76). Morristown, NJ: ACL.

Kohn, L. T., Corrigan, J. M., & Donaldson, M. (Eds.) (2000). *To err is human – building a safer health system*. Washington, DC: National Academic Press.

Lavrac, N., Dzeroski, S., Prinat, V., & Krizman, V. (1993). The utility of background knowledge in learning medical diagnostic rules. *Applied Artificial Intelligence, 7*, 273–293.

Lucaces, O., Taboada, F., Albaiceta, G., Domingues, L. A., Enriques, P., & Bahamonde, A. (2009). Predicting the probability of survival in intensive care unit patients from a small number of variables and training examples. *Artificial Intelligence in Medicine, 45*(1), 63–76.

Mavroforakis, M., Georgiou, H., Dimitropoulos, N., Cavouras, D., & Theodoridis, S. (2006). Mammographic masses characterization based on localized texture and dataset fractal analysis using linear, neural and support vector machine classifiers. *Artificial Intelligence in Medicine, 37*(2), 145–162.

McNaught, K., Clifford, S., Vaughn, M., Foggs, A., & Foy, M. (2001). A Bayesian belief network for lower back pain diagnosis. In P. Lucas, L. C. van der Gaag, & A. Abu-Hanna (Eds.), *Bayesian models in medicine – Workshop at AIME*. Caseais, Portugal.

Michalski, R., Mozetic, I., Hong, J., & Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application on three medical domains. In *Proceedings of the 5th national conference on artificial intelligence* (pp. 1041–1045). San Mateo,CA: Morgan Kaufmann.

Mizoguchi, F., Ohwada, H., Daidoji, M., & Shirato, S. (1997). Using inductive logic programming to learn classification rules that identify glaucomatous eyes. In N. Lavrač, E. Keravnou, & B. Zupan, (Eds.), *Intelligent data analyis in medicine and pharmacology* (pp. 227–242). Norwell, MA: Kluwer.

Morik, K., Imhoff, M., Brockhausen, P., Joachims, T., & Gather, U. (2000). Knowledge discovery and knowledge validation in intensive care. *Artificial Intelligence in Medicine, 19*(3), 225–249.

Morik, K., Joachims, T., Imhoff, M., Brockhausen, P., & Rüping, S. (2002). Integrating kernel methods into a knowledge-based approach to evidence-based medicine. In M. Schmitt, H. N. Teodorescu, A. Jain, A. Jain, S. Jain, & L. C. Jain, (Eds.), *Computational intelligence processing in medical diagnosis*, (Vol. 96) Studies in fuzziness and soft computing, (pp. 71–99). New York: Physica-Verlag.

Morik, K., Potamias, G., Moustakis, V. S., & Charissis, G. (1994). Knowledgeable learning using MOBAL: A medical case study. *Applied Artificial Intelligence, 8*(4), 579–592.

Moskovitch, R., Cohen-Kashia, S., Drora, U., Levya, I., Maimona, A., & Shahar, Y. (2006). Multiple hierarchical classification of free-text clinical guidelines. *Artificial Intelligence in Medicine, 37*(3), 177–190.

Ou, M., West, G., Lazarescu, M., & Clay, C. (2007). Dynamic knowledge validation and verification for CBR teledermatology system. *Artificial Intelligence in Medicine, 39*(1), 79–96.

Sanchez, D., & Moreno, A. (2005). Web mining techniques for automatic discovery of medical knowledge. In *Proceedings of the 10th conference on artificial intelligence in medicine*. Aberdeen, Scotland.

Scholz, M. (2002). Using real world data for modeling a protocol for ICU monitoring. In P. Lucas, L. Asker, & S. Miksch, (Eds.), *Working notes of the IDAMAP 2002 workshop*, (pp. 85–90). Lyon, France.

Shipp, M. A., Ross, K. N., Tamayo, P., Weng, A. P., Kutok, J. L., Aguiar, R. C., et al. (2002). Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature Medicine, 8*(1), 68–74.

Shortliffe, E. H. (1976). *Computer based medical consultations: MYCIN*. New York, Amsterdam: Elsevier.

Sieben, W., & Gather, U. (2007). Classifying alarms in intensive care–analogy to hypothesis testing. In *11th conference on artifical intelligence in medicine (AIME)* (pp. 130–138). Berlin: Springer.

Smith, W. P., Doctor, J., Meyer, J., Kalet, I. J., & Philips, M. H. (2009). A decision aid for intensity-modulated radiation-therapy plan selection in prostate cancer based on a prognostic Bayesian network and a Markov model. *Artificial Intelligence in Medicine, 46*(2), 119–130.

Srinivasan, A., Muggleton, S. H., King, R. D., & Sternberg, M. J. E. (1994). Carcinogenesis prediction using inductive logic programming. In B. Zupan, E. Keravnou, & N. Lavrac (Eds.), *Intelligent data analysis in medicine and pharmacology* (pp. 243–260). Norwell, MA: Kluwer.

Ten Teije, A., Lucas, P., & Miksch, S. (Eds.), (2006). *Workshop on AI techniques in healthcare: Evidence-based guidelines and protocols, held in conjunction with ECAI-2006*. Italy.

van't Veer, L. J., Dai, H. Y., van de Vijver, M. J., He, Y. D. D., Hart, A. A., Mao, M., et al. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature, 415*, 530–536.

Withayachumnankul, W., Ferguson, B., Rainsford, T., Findlay, D., Mickan, S. P., & Abbott, D. (2006). T-ray relevant frequencies for osteosarcoma classification. In D. Abbott, Y. S. Kivshar, H. H. Rubinstein-Dunlop, & S.-H. Fan, (Eds.), *Proceedings of SPIE*. Brisbane, Australia.

Wu, X., Lucas, P., Kerr, S., & Dijkhuisen, R. (2001). Learning bayesian-network topologies in realistic medical domains. In *Intelligent data analysis in medicine and pharmacology*. Medical Data Analysis, (pp. 302–307). Berlin/Heidelberg: Springer.

Zangwill, L. M., Chan, K., Bowd, C., Hao, J., Lee, T. W., Weinreb, R. N., et al. (2004). Heidelberg retina tomograph measurements of the optic disc and parapillary retina for detecting glaucoma analyzed by machine learning classifiers. *Investigative Ophthalmology and Visual Science, 45*(9), 3144–3151.

Zeng, D., Chen, H., Lynch, C., Eidson, M., & Gotham, I. (2005). Infectious disease informatics and outbreak detection. In H. Chen, S. Fuller, C. Friedman, & W. Hersh, (Eds.), *Medical informatics: knowledge management and data mining in biomedicine* (pp. 359–395). New York: Springer.

Memory Organization Packets

►Dynamic Memory Model

Memory-Based

►Instance-Based Learning

Memory-Based Learning

►Case-Based Reasoning

Merge-Purge

►Entity Resolution

Message

In ►Minimum Message Length inference, a binary sequence conveying information is called a message.

Meta-Combiner

A *meta-combiner* is a form of ►ensemble learning technique used with ►missing attribute values. Its common topology involves base learners and classifiers at the first level, and meta-learner and meta-classifier at the second level. The meta-classifier combines the decisions of all the base classifiers.

Metaheuristic

Marco Dorigo, Mauro Birattari,
Thomas Stützle

A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework that can be applied to different optimization problems with relatively few modifications. Examples of metaheuristics include simulated annealing, tabu search, iterated local search, evolutionary algorithms, and ant colony optimization.

Metalearning

Pavel Brazdil[1], Ricardo Vilalta[2],
Christophe Giraud-Carrier[3], Carlos Soares[1]
[1]University of Porto, Porto, Portugal
[2]University of Houston,
Houston TX, USA
[3]Brigham Young University, UT, USA

Synonyms

Adaptive learning; Dynamic selection of bias; Learning to learn; Ranking learning methods; self-adaptive systems

Definition

Metalearning allows machine learning systems to benefit from their repetitive application. If a learning system fails to perform efficiently, one would expect the learning mechanism itself to adapt in case the same task is presented again. Metalearning differs from base-learning in the scope of the level of adaptation; whereas learning at the base-level is focused on accumulating experience on a specific task (e.g., credit rating, medical diagnosis, mine-rock discrimination, fraud detection, etc.), learning at the metalevel is concerned with accumulating experience on the performance of multiple applications of a learning system.

Briefly stated, the field of metalearning is focused on the relation between tasks or domains, and learning algorithms. Rather than starting afresh on each new task, metalearning facilitates evaluation and comparison of learning algorithms on many different previous tasks, establishes benefits and disadvantages, and then recommends the learning algorithm, or combination of algorithms that maximizes some utility function on the new task. This problem can be seen as an algorithm selection task (Rice, 1976).

The utility or usefulness of a given learning algorithm is often determined through a mapping between characterization of the task and the algorithm's estimated performance (Brazdil & Henery, 1994). In general, metalearning can recommend more than one algorithm. Typically, the number of recommended algorithms is significantly smaller than the number of all possible (available) algorithms (Brazdil, Giraud-Carrier, Soares, & Vilalta, 2009).

Motivation and Background

The application of machine learning systems to ►classification and ►regression tasks has become a standard, not only in research but also in commerce and industry (e.g., finance, medicine, and engineering). However, most successful applications are custom-designed, the result of skillful use of human expertise. This is due, in part, to the large, ever increasing number of available machine learning systems, their relative complexity, and the lack of systematic methods for discriminating among them. The problem is further compounded by the fact that, in ►Knowledge Discovery from Databases, each operational phase (e.g., preprocessing, model generation) may involve a choice among various possible alternatives (e.g., progressive vs. random sampling, neural network vs. decision tree learning), as observed by Bernstein, Provost, and Hill (2005).

Current data mining systems are only as powerful as their users. These tools provide multiple algorithms within a single system, but the selection and combination of these algorithms must be performed before the system is invoked, generally by an expert user. For some researchers, the choice of learning and data transformation algorithms should be fully automated if machine learning systems are to be of any use to nonspecialists. Others claim that full automation of the data mining process is not within the reach of current technology. An intermediate solution is the design of assistant systems aimed at helping to select the right learning algorithm(s). Whatever the proposed solution, there seems to be an implicit agreement that metaknowledge should be integrated seamlessly into the data mining system. Metalearning focuses on the design and application of learning algorithms to acquire and use metaknowledge to assist machine learning users with the process of model selection. A general framework for this purpose, together with a survey of approaches, is in (Smith-Miles, 2008).

Metalearning is often seen as a way of redefining the space of inductive hypotheses searched by the learning algorithm(s). This issue is related to the idea of ▶search bias, that is, search factors that affect the definition or selection of inductive hypotheses (Mitchell, 1997). In this sense, metalearning studies how to choose the right bias dynamically and thus, differs from base-level learning, where the bias is fixed or user-parameterized. Metalearning can also be viewed as an important feature of self-adaptive systems, that is, learning systems that increase in efficiency through experience (Vilalta & Drissi, 2002).

Structure of the Metalearning System

A metalearning system is essentially composed of two parts. One part is concerned with the acquisition of metaknowledge for machine learning systems. The other part is concerned with the application of metaknowledge to new problems, with the objective of identifying an optimal learning algorithm or technique. The latter part – application of metaknowledge – can be used to help select or adopt suitable machine learning algorithms. So, for instance, if we are dealing with a ▶classification task, metaknowledge can be used to select a suitable ▶classifier for the new problem. Once this has been done, one can train the classifier and apply it to some unclassified sample for the purpose of class prediction.

In the following sections we begin by describing scenarios corresponding to the case when metaknowledge has already been acquired. We then provide an explanation of how this knowledge is acquired.

Employing Metaknowledge to Select Machine Learning Algorithms

The aim of this section is to show that metaknowledge can be useful in many different settings. We start by considering the problem of selecting suitable machine learning algorithms from a given set. The problem can be seen as a search problem. The search space includes the individual machine learning algorithms, and the aim is to identify the best algorithm. This process can be divided into two separate phases (see Fig. 1). In the first phase the aim is to identify a suitable subset of machine learning algorithms based on an input dataset. The selection method used in this process can exploit metaknowledge. This is in general advantageous, as it often leads to better choices. In some work the result of this phase is represented in the form of a ranked subset of machine learning algorithms. The subset of algorithms represents the reduced bias space. The ranking (i.e., ordering of different algorithms) represents the procedural search bias.

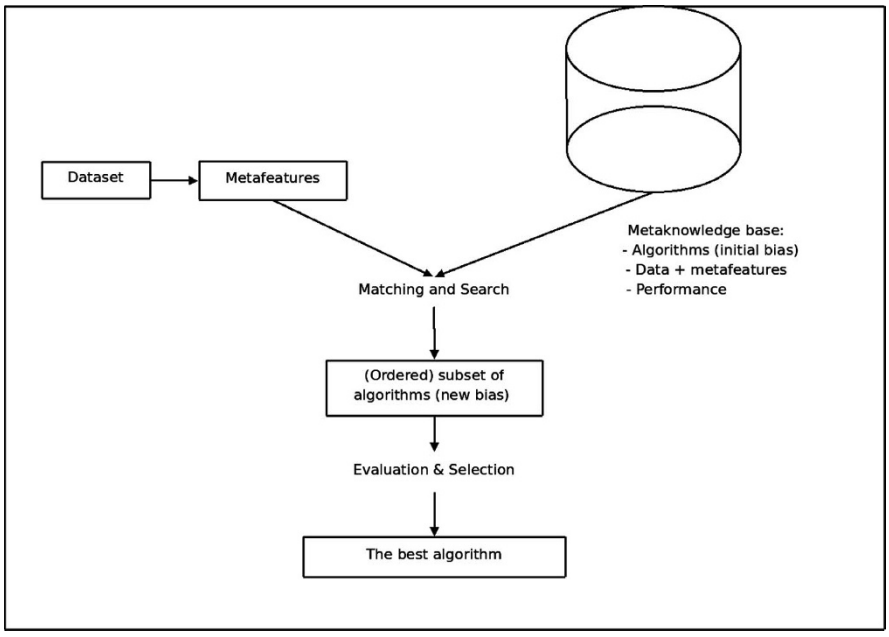
The second phase is used to search through the reduced space. Each option is evaluated using a given performance criteria (e.g., accuracy). Typically, cross-validation will be used to identify the best alternative.

We note that metaknowledge does not completely eliminate the need for the search process, but rather provides a more effective search. The effectiveness of the search depends on the quality of metaknowledge.

How the Subset of Algorithms Is Identified

Let us return to the algorithm selection problem. A metalearning approach to solving this problem relies on dataset characteristics or metafeatures to provide some information that would differentiate the performance of a set of given learning algorithms. These include various types of measures discussed in detail below.

Much previous work in dataset characterization has concentrated on extracting statistical and information-theoretic parameters estimated from the training set.



Metalearning. Figure 1. Selection of machine learning algorithms: Determining the reduced space and selecting the best alternative

Measures include number of classes, number of features, ratio of examples to features, degree of correlation between features and target concept, average class entropy, etc. (Engels & Theusinger, 1998). The disadvantage of this approach is that there is a limit to how much information these features can capture, given that all these measures are uni- or bi-lateral measures only (i.e., they capture relationships between two attributes only or between one attribute and the class).

Another idea is based on what are called *landmarkers;* these are simple and fast learners (Pfahringer, Bensusan & Giraud-Carrier, 2000). The accuracy of these simplified algorithms is used to characterize a dataset and to identify areas where each type of learner can be regarded as an expert. An important class of measures related to landmarkers uses information obtained on simplified versions of the data (e.g., samples). Accuracy results on these samples serve to characterize individual datasets and are referred to as *subsampling landmarks*.

One different class of techniques does not acquire the information in one step, but rather uses a kind of ▶active learning approach. This approach has been used to characterize algorithms by exploiting performance results on samples. The process of obtaining a characterization is divided into several steps. The result of one step affects what is done in the next step. In each step, a decision as to whether the characterization process should be continued is made first. If the answer is positive, the system determines which characteristics should be obtained in the next step (Brazdil et al., 2009).

All the measures discussed above are used to identify a subset of learning algorithms to reduce the search space (Fig. 1). The second phase in the algorithm selection problem can be done using a metalevel system that maps data characteristics to learning algorithms. One particular approach uses the *k*-NN method at the metalevel. The *k*-NN method is used to identify the most similar datasets. For each of these datasets, a ranking of the candidate algorithms is generated based on user-defined performance criteria, such as accuracy and learning time (Nakhaeizadeh & Schnabl, 1997). The rankings obtained are aggregated to generate a final recommended ranking of algorithms.

Acquisition of Metaknowledge

We now address how metaknowledge can be acquired. One possibility is to rely on expert knowledge. Another possibility is to use an automatic procedure. We explore both alternatives briefly below.

One way of representing metaknowledge is in the form of rules that match domain (dataset) characteristics with machine learning algorithms. Such rules can be hand-crafted, taking into account theoretical results, human expertise, and empirical evidence. For example, in decision tree learning, a heuristic rule can be used to switch from univariate tests to linear tests if there is a need to construct nonorthogonal partitions over the input space. This method has serious disadvantages, however. First, the resulting rule set is likely to be incomplete. Second, timely and accurate maintenance of the rule set as new machine learning algorithms become available is problematic. As a result, most research has focused on automatic methods, discussed next.

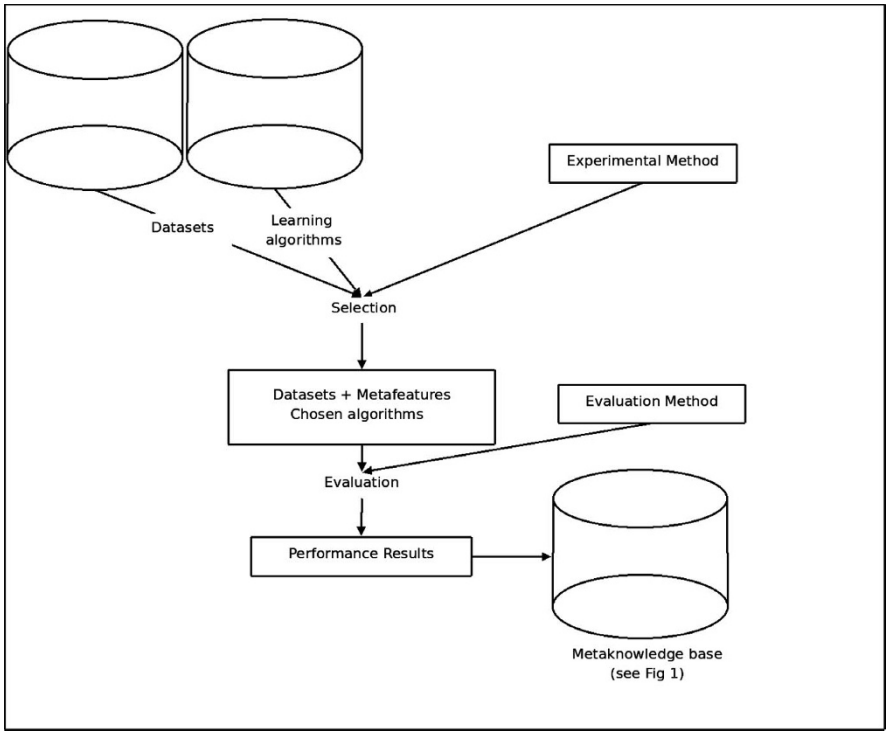
One other way of acquiring metaknowledge relies on automatic experimentation. For this we need a pool of problems (datasets) and a set of machine learning algorithms that we wish to consider. Then we need to define also the experimental method which determines which alternatives we should experiment with and in which order (see Fig. 2 for details).

Suppose that we have a dataset (characterized using certain metafeatures), in combination with certain machine learning algorithms. The combination is assessed using an evaluation method (e.g., cross-validation) to produce performance results. The results, together with the characterization, represent a piece of metadata that is stored in the metaknowledge base. The process is then repeated for other combinations of datasets and algorithms.

In this context it is useful to distinguish between two different types of methods potentially available at the metalevel. One group involves ▶lazy learning methods. These delay the generalization of metadata to the application phase. The other group involves learning algorithms whose aim is to generate a generalization model (e.g., a decision tree or decision rules). This generalization model, applied to the metadatabase, represents in effect the acquired metaknowledge.

Inductive Transfer

As we mentioned before, learning should not be viewed as an isolated task that starts from scratch on every



Metalearning. Figure 2. Acquisition of metadata for the metaknowledge base

new problem. As experience accumulates, the learning mechanism is expected to perform increasingly better. One approach to simulate the accumulation of experience is by transferring metaknowledge across domains or tasks. This process is known as *inductive transfer*. In many cases the goal is not simply to generate explicit metaknowledge, but rather to incorporate it in the given base-level system(s). The resulting base-level system thus becomes a generic solution applicable across domains. More details about this can be found in a separate entry on this topic.

Cross References

►Inductive Transfer

Recommended Reading

Bernstein, A., Provost, F., & Hill, S. (2005). Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification. *IEEE Transactions on Knowledge and Data Engineering, 17*(4), 503–518.

Brazdil, P., Giraud-Carrier, C., Soares, C., & Vilalta, R. (2009). *Metalearning – applications to data mining*. Berlin: Springer.

Brazdil, P., & Henery, R. (1994). Analysis of results. In D. Michie, D. J. Spiegelhalter, & C. C. Taylor (Eds.), *Machine learning, neural and statistical classification*. England: Ellis Horwood.

Engels, R., & Theusinger, C. (1998). Using a data metric for offering preprocessing advice in data-mining applications. In H. Prade (Ed.), *Proceedings of the 13th European conference on artificial intelligence* (pp. 430–434). Chichester, England: Wiley.

Mitchell, T. (1997). *Machine learning*. New York: McGraw Hill.

Nakhaeizadeh, G., & Schnabl, A. (1997). Development of multi-criteria metrics for evaluation of data mining algorithms. In *Proceedings of the 3rd international conference on knowledge discovery and data mining* (pp. 37–42). Newport Beach, CA: AAAI Press.

Pfahringer, B., Bensusan, H., & Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In *Proceedings of the 17th international conference on machine learning* (pp. 743–750).

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers, 15*, 65–118.

Smith-Miles, K. A. (2008). Cross-disciplinary perpectives on meta-learning for algorithm selection. *ACM Computing Surveys, 41*(1), Article No. 6.

Vilalta, R., & Drissi, Y. (2002). A perspective view and survey of metalearning. *Artificial Intelligence Review, 18*(2), 77–95.

Minimum Cuts

►Graph Clustering

Minimum Description Length Principle

Jorma Rissanen
Helsinki Institute of Information Technology, Helsinki, Finland
Tampere University of Technology, Finland
University of London, England

Synonyms

Information theory; MDL; Minimum encoding inference

Definition

The original "general" minimum description length (MDL) principle for estimation of statistical properties in observed data y^n, or the *model* $f(y^n;\theta,k)$, represented by parameters $\theta = \theta_1,\ldots,\theta_k$, can be stated thus,

- "*Find the model with which observed data and the model can be encoded with shortest code length*":

$$\min_{\theta,k}\,[\,\log 1/f(y^n;\theta,k) + L(\theta,k)\,],$$

where $L(\theta,k)$ denotes the code length for the parameters.

The principle is very general and produces a model defined by the estimated parameters. It leaves the selection of $L(\theta,k)$ open, and in complex applications the code length can be calculated by visualizing a coding process. The only requirement is that the data must be decodable.

Motivation and Background

The MDL principle is based on the fact that it is not possible to compress data well without taking advantage of the regular features in them. Hence, estimation and data compression have similar goals although they are not identical. In estimation, we must describe the model explicitly, while the algorithm to compress the data may take advantage of regular features implicitly without isolating them. This means that such an algorithm does not produce any model that could be used for machine learning.

We describe here a new sharper *Complete MDL* principle, which corrects the shortcomings of the old principle. It also separates estimation from data compression and delivers an explicit model. The objective in this entry is to show that the complete MDL principle is not only intuitively appealing but it plays a fundamental role in all estimations, and in all inductive inferences for that matter, since any meaningful inference about data must be based on a good model of it. In fact, we argue that there cannot be a rational comprehensive theory of estimation unless it is founded on the MDL principle or some of its equivalent forms.

Theory

We begin by outlining the problem of model building and estimation. The objective is to fit parametric models of type $f(y^n|\mathbf{x}^n, \theta)$ to data $y^n = y_1, \ldots, y_n$, given explanatory variables $\mathbf{x}^n = \mathbf{x}_1, \ldots, \mathbf{x}_n$, where $\theta = \theta_1, \ldots, \theta_k$ are real-valued parameters. To simplify the notations we drop the explanatory variables and consider classes of models $\mathcal{M}_k = \{f(y^n;\theta)\}$. For fixed number k of parameters the fitting is done by some *estimator* function

$$\bar{\theta}(\cdot) : y^n \mapsto \bar{\theta}(y^n)$$

taking data to parameter values, which in turn pick out estimated models of data. For simplicity we discuss first the estimation of the real-valued parameters. We do not assume the existence of a special "true" model defined by a parameter θ^*, which raises the problem of how to assess the goodness of the estimators and the estimated models. Clearly, it cannot be done by any distance measure between the estimate $\bar{\theta}(y^n)$ and θ^*. We think that the only way the assessment can be done in general without narrow and special criteria is in terms of the probability which the estimator $\bar{\theta}(\cdot)$ assigns to the observed data. A large probability means a good fit while a small probability means a bad fit. How do we calculate this probability? Importantly, notice that it cannot be the number $f(y^n;\bar{\theta}(y^n))$, bearing the mystical name "likelihood," because its integral over all data y^n is not unity. However, by normalization we get a valid yardstick for the goodness measure

$$\bar{f}(y^n;k) = \frac{f(y^n;\bar{\theta}(y^n),k)}{\bar{C}_k}$$

$$\bar{C}_k = \int f(y^n;\bar{\theta}(y^n),k)dy^n,$$

where we now show the number of parameters k. When even the number of parameters is to be estimated, the yardstick is as follows

$$\bar{f}(y^n) = \bar{f}(y^n;\bar{k}(y^n))/\bar{C} \quad (1)$$

$$\bar{C} = \sum_k \int_{\bar{k}(y^n)=k} \bar{f}(y^n;k)dy^n, \quad (2)$$

where $\bar{k}(y^n)$ denotes an estimator for k.

Optimal Yardstick

We view estimation as analogous to measuring a physical property like weight or mass of an object: The object here is the observed data and the property is the model in a selected class defined by the parameters, while the probability an estimated model assigns to the data corresponds to the accuracy.

We need a yardstick as the instrument like the scale with which the measuring is done. It will be defined by a special estimator and the distribution it defines. Clearly, the yardstick must not depend on the data set whose property we want to measure no more than the scale for weighing an object must not depend on the object. The requirement then is that it should be determined by the model class. We also want a yardstick that assigns a large probability to the data, or, equivalently, a small negative logarithm of the probability, which can be interpreted as code length. However, there is the fundamental difficulty that no distribution $\bar{f}(y^n;k)$ exists which assigns the largest probability to all data. Quite remarkably, there is a unique yardstick that satisfies the two requirements, repeated here:

1. $\bar{f}(\cdot;k)$ to be determined by the model class $\mathcal{M}_k$
2. Minimal code length $\log 1/\bar{f}(y^n;k)$ for *all* data y^n

and, similarly, when even the number of parameters is to be estimated.

The unique yardstick when the real-valued parameters are estimated is defined by the ML (maximum Likelihood) estimator, $\hat{\theta}(y^n)$, which maximizes the probability the model assigns to data, or $\max_\theta f(y^n;\theta,k)$:

$$\hat{f}(y^n;k) = \frac{f(y^n;\hat{\theta}(y^n),k)}{\hat{C}_k} \quad (3)$$

$$\hat{C}_k = \int f(y^n;\hat{\theta}(y^n),k)dy^n$$

$$= \int d\hat{\theta} \int_{\hat{\theta}(y^n)=\hat{\theta}} f(y^n;\hat{\theta},k)dy^n. \quad (4)$$

The proof of that there is a unique distribution $\bar{f}(y^n;k) = \hat{f}(y^n;k)$ satisfying the two requirements amounts to noticing that the ratio $\bar{f}(y^n;k)$ cannot be maximum unless the numerator is maximized. Notice that the famous maximized likelihood, the numerator of $\hat{f}(y^n;k)$, in itself means nothing.

The unique yardstick when even the number of parameters is estimated is

$$\hat{f}(y^n) = \frac{\max_k f(y^n;\hat{\theta}(y^n),k)/\hat{C}_k}{\hat{C}} \tag{5}$$

$$\hat{C} = \sum_k \int_{\hat{k}(y^n)=k} \hat{f}(y^n;k)dy^n, \tag{6}$$

where $\hat{k}(y^n)$, or the maximizing k, is not the maximum likelihood estimator.

The evaluation of these yardsticks on an observed data string y^n gives the MDL criterion. The calculation of the normalizing coefficients is the main problem. It can be evaluated most easily for finite alphabets. Asymptotically the optimal estimation criterion amounts to this

$$\min_k \left[\log 1/f(y^n;\hat{\theta}(y^n),k) + \frac{k}{2}\log\frac{n}{2\pi} + \log\int |J(\theta)|^{1/2}d\theta\right],$$

where

$$J(\theta) = \lim n^{-1}E\left\{\frac{\partial^2 \log 1/f(y^n;\theta,k)}{\partial\theta_i\partial\theta_j}\right\}.$$

is the Fisher information matrix. Hence, this term is a positive constant and can be ignored for large amounts of data.

Cross References

►Minimum Message Length

Recommended Reading

Grünwald, P. D. (2007). *The minimum description length principle* (703 pp.). Cambridge/London: The MIT Press.

Rissanen, J. (2007). *Information and complexity in statistical modeling* (142 pp.). Springer: New York.

Rissanen, J. (September 2009). Optimal estimation. *IEEE Information Theory Society Newsletter*, 59(3).

Minimum Encoding Inference

►Minimum Description Length Principle
►Minimum Message Length

Minimum Message Length

Rohan A. Baxter
Australian Taxation Office, ACT, Australia

Synonyms

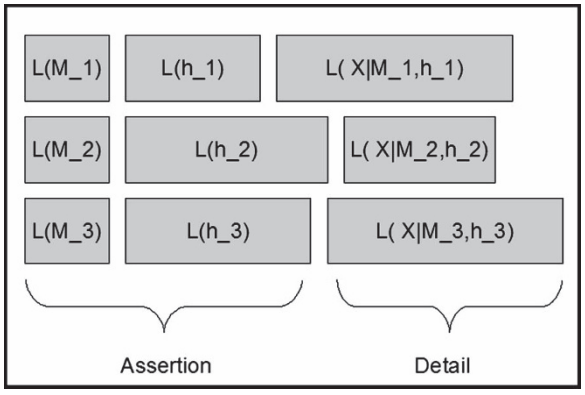
Minimum encoding inference

Definition

Minimum message length (MML) is a theory of ►inductive inference whereby the preferred model is the one minimizing the expected message length required to explain the data with the prior information.

Given data, represented in a finite binary string, E, is an "►explanation" of the data which is a two-part ►message or binary string encoding the data to be sent between a sender and receiver. The first part of the message (the "►assertion") states a hypothesis, model, or theory about the source of the data. The second part (the "►detail") states those aspects of E which cannot be deduced from this assertion and prior knowledge. The sender and receiver are assumed to have agreed on the prior knowledge, the assertion code, and the detail code before the message is constructed and sent. The shared prior knowledge captures their belief about the data prior to seeing the data and is needed to provide probabilities or, equivalently, optimum codes, for the set of models. The assertion and detail codes can be equivalently considered to be the shared language for describing models (for the assertion code) and for describing data (for the detail code).

Out of all possible models which might be advanced about the data, MML considers the best inference as that model which leads to the shortest explanation. The length of the explanation can be calculated using ►Shannon's information, $L(E) = -\log(P(E))$, where $L(E)$ is the length of the shortest string encoding an



Minimum Message Length. Figure 1. A view of model selection by minimum message length (MML). The data is coded assuming a model and parameters in the assertion. The model and parameters are coded in the assertion. As shown here, often different models have same probability, while the code lengths for model parameters and data detail differ between the models

event, E, and $P()$ is the probability of a message containing E.

To compare models, we calculate the explanation length for each and prefer the one with shortest explanation length. Figure 1 shows three models being evaluated and the different lengths of the assertion and details for each. Model 2 is preferred as it has the MML.

Motivation and Background

The original motivation for MML inductive inference is the idea that the best explanation of the facts is the shortest (Wallace & Boulton, 1968). By inductive inference, we mean the selection of a best model of truth. This goal is distinct from a best model for prediction of future data or for choosing a model for making the most beneficial decisions. In the field of Machine Learning, greater focus has been on models for prediction and decision, but inferences of the best models of truth have an important separate application.

For discrete models, MML looks like Bayesian model selection since choosing H to minimize the explanation length of data X:

$$-\log P(H) - \log P(X|H) = -\log(P(H)P(X|H),$$

is often, but not always, as discussed below, equivalent to choosing H to maximize the probability

$$P(H|X):$$
$$P(H|X) = \frac{P(H)P(X|H)}{P(X)},$$

where $P(X)$ is a constant for a given detail code.

For models with real-valued parameters, the equivalence between MML and Bayesian model selection always breaks down (Wallace, 2005, p. 117). Stating the $P(H)$ in a message requires real-valued parameters in H to be stated to a limited precision. The MML coding approach replaces a continuum of possible hypotheses with a discrete subset of values, and assigns a nonzero prior probability to each discrete theory. The discrete subsets are chosen to optimize the expected message length given the prior knowledge assumptions.

For models with only discrete-valued parameters, the equivalence between MML and Bayesian model selection may break down if the discrete values chosen involve the merging of values in the assumed prior distribution, $P(H)$ (Wallace, 2005, p. 156). This may occur with a small dataset if the data is insufficient to justify a codebook distinguishing individual members of H.

Other than a discretized hypothesis space, MML shares many properties of Bayesian learning such as sufficiency, avoidance of overfitting, and consistency (Wallace, 2005). One difference arising from the discretized hypothesis space is that MML allows inductive inference to be invariant under arbitrary monotonic transformations of parameter spaces. The Bayesian learning options for model choice such as the maximum a posteriori (MAP) estimate are not invariant under such transformations. Other theoretical benefits include consistency and guarantees against overfitting.

Message lengths of an explanation can be based on the theory of algorithmic complexity (AC) (Wallace & Dowe, 1999), instead of Shannon's information. The AC of a string with respect to a Universal Turing Machine, T, can be related to Shannon's information by

regarding T as defining a probability distribution over binary strings, $P(S)$, such that:

$$P_{T(S)} = 2^{-\mathrm{AC}(S)} \quad \forall S.$$

The connection with AC has some appeal for applications involving data that are not random in a probabilistic sense, such as function approximation where data seems to be from a deterministic source. In these cases, after fitting a model, the data residuals can be encoded using AC randomness, since the probabilistic sense of randomness does not apply (Wallace, 2005, p. 275).

Theory

Strict MML (SMML) estimators refer to the estimator functions which exactly minimize the expected message length (Wallace & Boulton, 1975). Most practical MML estimators are not strict and are discussed in a separate section on Approximations.

An SMML estimator requires (Dowe, Gardner, & Oppy, 2007):

- X, a data space, and a set of observations from the dataspace, $\{x_i : i \in N\}$
- $p(x|h)$, a conditional probability function over data given a model, h
- H is a model space. For example, H can be a simple continuum of known dimension k
- $P(h)$: a prior probability density on the parameter space $H : \int_H P(h)\,\mathrm{d}h = 1$

X, H, and the functions $\mathrm{P}(h), p(x|h)$ are assumed to be known a priori by both sender and receiver of the explanation message. Both sender and receiver agree on a code for X, using knowledge of X, H, $p(h)$, and $f(x|h)$ only.

The marginal prior probability of the data x follows from the assumed background knowledge:

$$r(x) = \int_H p(x|h)P(h)\,\mathrm{d}h.$$

The SMML estimator is a function $m : X \to H : m(x) = h$, which names the model to be selected.

The assertion, being a finite string, can name at most a countable subset of H. Call the subset $H^* = \{h_j : j = 1, 2, 3, \ldots\}$. The choice of H^* implies a coding distribution over $H^* : f(h_j) = q_j > 0 : j = 1, 2, 3, \ldots$ with $\sum_j q_j = 1$. So choice of H^* and q_j lead to a message length:

$$-\log q_j - \log p(x|h_j).$$

The sender, given x, will choose an h to make the explanation short. This choice is described by an estimator function: $m(x) : X \to H$ so that the length of the explanation is:

$$I_1(x) = -\log q(m(x)) - \log p(x|m(x)),$$

and the expected length is (Wallace, 2005, p. 155)

$$I_1 = -\sum_{x \in X} r(x)[\log q(m(x)) + \log p(x_i|m(x_i))].$$

Consider how to choose H^* and coding distribution q_j to minimize I_1. This will give the shortest explanation on average, prior to the sender seeing the actual data.

Define $t_j = \{x : m(x) = h_j\}$, so that t_j is the set of data which results in assertion h_j being used in the explanation. I_1 can now be written as two terms:

$$I_1 = -\sum_{h_j \in H_{\text{start}}} \left(\sum_{x_i \in t_j} r_i \right) \log q_j - \sum_{h_j \in H_{\text{start}}} \sum_{x_i \in t_j} r_i \log p(x_i|h_j).$$

The first term of I_1 is minimized by choosing:

$$q_j = \sum_{x_i \in t_j} r_j.$$

So the coding probability assigned to estimate h_j is the sum of the marginal probabilities of the data values resulting in h_j. It is the probability that estimate h_j will be used in the explanation based on the assumptions made.

The second term of I_1 is the average of the log likelihood over the data values used in h_j.

Example with Binomial Distribution

This section describes the SMML estimator for the binomial distribution. For this problem with 100 independent trials giving success or failure, we have $p(x|p) = p^n(1-p)^{100} - s, h(p) = 1$, where s is the observed number of successes and p is the unknown probability of success.

Minimum Message Length. Table 1 A strict MML (SMML) estimator for binomial distribution (Farr & Wallace, 2002; Wallace, 2005, p. 159)

j	s	p_j
1	0	0
2	1–6	0.035
3	7–17	0.12
4	18–32	0.25
5	33–49	0.41
6	50–66	0.58
7	67–81	0.74
8	82–93	0.875
9	94–99	0.965
10	100	1

We have an SMML estimator minimizing I_1 in Table 1. I_1 has 52.068 nits. Note that the partition p_j in Table 1 is not unique due to asymmetry in having 10 partitions of 101 success counts. Note the difference between the SMML estimate, p_j, and the MAP estimate $s/100$ in this case. For example of 50 observed successes, the MAP estimate is 0.5 while SMML estimate is 0.58. With 49 successes, the SMML estimate jumps to 0.41, so it is very discrete. The SMML estimate spacings are consistent with the expected error and so the MAP estimates are arguably overly precise and smooth. This is less than 0.2 nits more than the optimal one-part code based on the marginal probability of the data $-\log r(x)$.

Approximations

SMML estimators are hard to find in practice and various approximations of SMML estimators have been suggested. We focus on the quadratic approximation here, often called the MML estimator or MML87 (Wallace & Freeman 1987). Other useful approximations have been developed and are described in Wallace, (2005). The use of approximations in applications requires careful checking of the assumptions made by the approximation (such as various regularity conditions) to ensure that the desirable theoretical properties of MML inductive inference still apply.

$$I_1(x) \approx -\log \frac{f(h')}{\sqrt{\frac{F(h')}{12}}} + [-\log p(x|h')] + \frac{0.5F(h',x)}{F(h')},$$

where $F(h)$ is the Fisher Information:

$$F(h') = -E\frac{\partial^2}{(\partial h')^2}\log p(x|h')$$

$$= -\sum_{x\in X} p(x|h')\frac{\partial^2}{(\partial h')^2}\log p(x|h').$$

The assumptions are (Wallace, 2005; Wallace & Freeman, 1987):

- $f(x|h)$ is approximately quadratic on theta near its maximum
- H has a locally Euclidean metric
- Fisher information is defined everywhere in H
- $f(h)$and $F(h)$ vary little over theta of order $1/\sqrt{F(h)}$

A further approximation has the third term simplify to 0.5 only (Wallace, 2005, p. 226) which assumes $F(h,x) \approx F(h)$.

The MML estimator is a discretized MAP estimator with the prior $P(h)$ being discretized as:

$$f(h') \approx \frac{P(h')}{\sqrt{F(h')}}.$$

In practice, note that the Fisher Information may be difficult to evaluate. Various approximations have been made for the Fisher Information where appropriate for particular applications.

Applications

MML estimators have been developed for various probability distributions such as binomial, multinomial, and Poisson. MML estimators have also been developed for probability densities such as Normal, von-Mises, and Student's t (Wallace, 2005). These estimators and associated coding schemes are then useful components for addressing more complex model selection problems in Machine Learning.

There have been many applications of MML estimators to model spaces from Machine Learning (Allison, 2009; O'Donnell, Allison, & Korb, 2006; Wallace, 2005). We will now briefly note MML applications for mixture models, regular grammars, decision trees, and causal nets. MML estimators have also been developed for multiple ▶linear regression (Wallace, 2005), polynomial regression (Wallace, 2005), ▶neural networks (Allison, 2009), ARMA time series, Hidden Markov Models (Edgoose & Allison, 1999), sequence alignment (Allison, 2009), phylogenetic trees (Allison, 2009), factor analysis (Wallace, 2005), cut-point estimation (Wallace, 2005), and image segmentation.

Model-Based Clustering or Mixture Models

Clustering was the first MML application from Wallace and Boulton's 1968 paper (Wallace & Boulton, 1968). Some changes to the coding scheme have occurred over the decades. A key development was the switch from definite assignment of classes to things to probabilistic assignment in the 1980s. The MML model selection and a particularly efficient search involving dynamic splitting and merging of clusters was implemented in a FORTRAN program called Snob (since it discriminated between things).

The assertion code consists of:

1. The number of classes
2. For each class
   (a) The population proportion
   (b) Parameters of the statistical distribution for each attribute (or an insignificant flag)

The detail code consists of, for each datum the class to which it belongs and attribute values assuming the distribution parameters of the class. Bits-back coding is used to partially or probabilistically assign a class to each datum. This efficiency is needed to get consistent estimates.

Probabilistic Finite State Machines

Probabilistic finite state machines (PFSM) can represent probabilistic regular grammars (Wallace, 2005). A simple assertion code for the discrete finite state machines (FSM) structure, as developed by Wallace and Georgeff, is as follows:

- Provide number of states, S, using a prior $P(S)$
- For each state, code the number of arcs leaving the state, $\log(K+1)$ where $K+1$ is maximum number of arcs possible
- Code the symbols labeling the arcs, $\log \binom{K+1}{C\ a_s}$
- For each arc, code the destination state, $a_s \log S$

The number of all states other than state 1 is arbitrary, so the code permits $(S-1)!$, equal length, different descriptions of the same FSM. This inefficiency can be adjusted for by subtracting $\log(S-1)!$

A candidate detail code used to code the sentences is an incremental code where each transition from state to state is coded incrementally, using $\log n_{sk} + 1/\nu_s + a_s$, where n_{sk} is the number of times this arc has already been followed and ν_s is the number of times the state has already been left.

This application illustrates some general issues about assertion codes for discrete structures:

1. There can be choices about what to include in the assertion code. For example, the transition probabilities are not part of the assertion code above, but could be included, with adjustments, in an alternative design (Wallace, 2005).
2. Simple approaches with interpretable priors may be desirable even if using non-optimal codes. The assumptions made should be validated. For example, arcs between states in FSMs are usually relatively sparse (a_s = S) so a uniform distribution is not a sensible prior here.
3. Redundancy comes from being able to code equivalent models with different descriptions. For some model spaces, determining equivalence is either not possible or very expensive computationally.
4. Redundancy can come from the code allowing description of models that cannot arise. For example, the example assertion code could describe a FSM with states with no arcs.
5. Exhaustive search of model space can only be done for small FSMs. For larger applications, the performance of the MML model selection is conflated with performance of the necessary search space heuristics. This issue also occurs with decision trees, causal nets, etc.

In a particular application, it may be appropriate to trade-off redundancy with interpretability in assertion code design.

Decision Trees

Assertion codes for Decision trees and graphs have been developed (Wallace, 2005; Wallace & Patrick, 1993). An assertion describes the structure of the tree, while the detail code describes the target labels. The number of attributes, the arity of each attribute, an agreed attribute order, and probability that a node is a leaf or split node are assumed known by sender and receiver. Like the PFSM transition probabilities, the leaf class distributions are not explicitly included in the decision tree model (a point of distinction from Bayesian tree approaches).

An assertion code can be constructed by performing a prefix traversal of the tree describing each node. Describing a node requires –log_2 P_L if it is a leaf and –log_2 P_s if it is a split node. If it is a split node, the attribute that it splits on must be specified, requiring log_2 (number of available attributes). If it is a leaf node, the data distribution model should be specified, for example, the parameters of a binomial distribution if the data consists of two classes.

Causal Nets

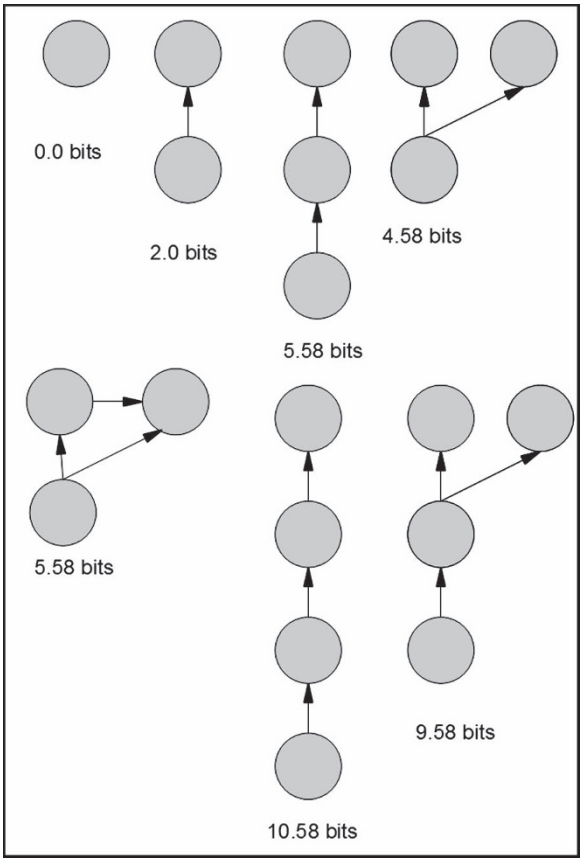
(Dai, Korb, Wallace, & Wu, 1997; Neil, Wallace, & Korb, 1999; O'Donnell et al., 2006)
The assertion code has two parts.
First part: DAG:

1. Specify an ordering of variables, $\log N!$
2. Specify which of M_a possible arcs are present, $\log(N(N-1)/2)$ bits on assumption probability an arc is present is 0.5

Second part: Parameters:

1. For each variable, state form of conditional distribution, then parameters of the distribution. Then encode all N values of v_j according to the distribution (Fig. 2)

Note that the assertion code is intermixed with the detail code for each variable (Wallace, 2005). Further adjustments are made to deal with grouping of causal nets with various equivalences or near-equivalences. This requires a further approximation because no attempt is made to code the best representative causal net from the group of causal nets described.



Minimum Message Length. Figure 2. Assertion code lengths for different DAGS using the example coding scheme

Future Directions

There seems potential for further development of feasible approximations that maintain the key SMML properties. Crossover of exciting new developments in coding theory may also help with development of MML estimators. Examples include stochastic encoding such as bits-back coding, discovered by Wallace (1990) and since expanded to many new application areas showing connections between MML with variational learning and ensemble learning (Honkela & Valpola,

2004). Another area is the relationship between optimum hypothesis discretization and indices of resolvability and rate-distortion optimization (Lanterman, 2001).

MML estimators will continue to be developed for the new model spaces that arise in Machine Learning. MML relevance seems assured because with complex models, such as social networks, the best model is the useful outcome, rather than a prediction or posterior distribution of networks.

Open source software using MML estimators for difference machine learning models is available (MML software).

Cross References

►Bayesian Methods
►Inductive Inference
►Minimum Description Length

Recommended Reading

Allison, L. (2009). 27/1/2010 MML website, http://www.allisons.org/ll/MML/

Dai, H., Korb, K. B., Wallace, C. S., & Wu, X. (1997). A study of causal discovery with weak links and small samples. In *Proceedings of the fifteenth international joint conference on artificial intelligence* (pp. 1304–1309). San Francisco: Morgan Kaufman.

Dowe, D. L., Gardner, S. B., & Oppy, G. (2007). Bayes not bust!: Why simplicity is no problem for Bayesians. *The British Journal for the Philosophy of Science, 58*, 709–754.

Edgoose, T., & Allison, L. (1999). MML Markov classification of sequential data. *Statistics and Computing, 9*(4), 269–278.

Farr, G. E, & Wallace, C. S. (2002). The complexity of strict minimum message length inference. *The Computer Journal, 45*(3), 285–292.

Honkela, A., & Valpola, H. (2004). Variational learning and bits-back coding: An information-theoretic view to Bayesian learning. *IEEE Transactions on Neural Networks, 15*(4), 800–810.

Lanterman, A. D. (2001). Schwarz, Wallace and Rissanen: Intertwining themes in theories of model selection. *International Statistical Review, 69*(2), 185–212.

MMLsoftware: www.datamining.monash.edu.au/software, 27/1/2010 http://allisons.org/ll/Images/People/Wallace/FactorSnob/

Neil, J. R., Wallace, C. S., & Korb, K. B. (1999). Learning Bayesian networks with restricted interactions. In K. B. Laskey & H. Prade (Eds.), *Proceedings of the fifteenth conference of uncertainty in artificial intelligence (UAI-99)* (pp. 486–493). San Francisco: Morgan Kaufmann.

O'Donnell, R., Allison, L., & Korb, K. (2006) *Learning hybrid Bayesian networks by MML. Lecture notes in computer science: AI 2006 – advances in artificial intelligence* (Vol. 4304, pp. 192–203). Berlin: Springer.

Wallace, C. S. (1990). Classification by minimum-message length inference. In S. G. Akl, et al. (Eds.), *Advances in computing and information-ICCI 1990, No. 468 in Lecture notes in computer science*. Berlin: Springer.

Wallace, C. S. (2005). *Statistical and inductive inference by MML: Information sciences and statistics*. Berlin: Springer.

Wallace, C. S., & Boulton, D. M. (1968). An information measure for classification. *Computer Journal, 11*, 185–194.

Wallace, C. S., & Boulton, D. M. (1975). An information measure for single-link classification. *The Computer Journal, 18*(3), 236–238.

Wallace, C. S., & Dowe, D. L. (1999). Minimum message length and Kolmogorov complexity. *Computer Journal, 42*(4), 330–337.

Wallace, C. S., & Freeman, P. R. (1987). Estimation and inference by compact coding. *Journal of the Royal Statistical Society (Series B), 49*, 240–252.

Wallace, C. S., & Patrick, J. D. (1993). Coding decision trees. *Machine Learning, 11*, 7–22.

Missing Attribute Values

Ivan Bruha
McMaster University, Hamilton, ON, Canada

Synonyms

Missing values; Unknown attribute values; Unknown values

Definition

When inducing ►decision trees or ►decision rules from real-world data, many different aspects must be taken into account. One important aspect, in particular, is the processing of *missing* (*unknown*) ►attribute values. In machine learning (ML), instances (objects, observations) are usually represented by a list of attribute values; such a list commonly has a fixed length (i.e., a fixed number of attributes).

The topic of missing attribute values has been analyzed in the field of ML in many papers (Brazdil & Bruha, 1992; Bruha and Franek, 1996; Karmaker & Kwer, 2005; Long & Zhang, 2004; Quinlan, 1986, 1989). Grzymala-Basse (2003) and Li and Cercone (2006) discuss the treatment of missing attribute values using the rough set strategies.

There are a few directions in which missing (unknown) attribute values as well as the corresponding routines for their processing may be studied and designed. First, the *source of "unknownness"* should be investigated; there are several such sources (Kononenko, 1992):

- A value is *missing* because it was forgotten or lost
- A certain attribute is *not applicable* for a given instance (e.g., it does not exist for a given observation)
- An attribute value is *irrelevant* in a given context
- For a given observation, the designer of a training database does not care about the value of a certain attribute (the so-called *dont-care* value)

The first source may represent a random case, while the remaining ones are of structural character.

Moreover, it is important to define formulas for *matching instances* (examples) containing missing attribute values with decision trees and decision rules as different matching routines vary in this respect.

Strategies for Missing Value Processing

The aim of this section is to survey the well-known strategies for the processing of missing attribute values. Quinlan (1989) surveys and investigates quite a few techniques for processing unknown attribute values processing for the TDIDT family. This chapter first introduces the seven strategies that are applied in many ML algorithms. It then discusses particular strategies for the four paradigms: Top Down Induction Decision Trees (TDIDT), (also known as the decision tree paradigm, or divide-and-conquer), covering paradigm (also known as the decision rules paradigm), Naive Bayes, and induction of ▶association rules. The conclusion compares the above strategies and then portrays possible directions in combining these strategies into a more robust system.

To deal with real-world situations, it is necessary to process incomplete data – i.e., data with missing (unknown) attribute values. Here we introduce the seven strategies (routines) for processing missing-attribute-values. They differ in the style of the solution of their matching formulae. There are the following natural ways of dealing with unknown attribute values:

1. Ignore the example (object, observation) with missing values: strategy *Ignore* (*I*)
2. Consider the missing (unknown) value as an additional regular value for a given attribute: strategy *Unknown* (*U*) or
3. Substitute the missing (unknown) value for matching purposes by a suitable value which is either
   - The most common value: strategy *Common* (*C*)
   - A proportional fraction: strategy *Fraction* (*F*)
   - Any value: strategy *Anyvalue* (*A*)
   - Random value: strategy *Random* (*Ran*)
   - A value determined by a ML approach: strategy *Meta-Fill-In* (*M*) of the known values of the attribute that occur in the training set

Dealing with missing attribute values is in fact determined by matching a selector (see the corresponding definitions below) with an instance. A matching procedure of a selector with a fully specified instance returns the uniform solution: the instance either matches or not. Dilemmas arise when a partially defined instance is to be matched.

We now informally introduce a couple of definitions. An inductive algorithm generates a knowledge base (decision tree or a set of decision rules) from a *training set* of K training examples, each accompanied by its desired ▶class $C_r, r = 1, \ldots, R$. Examples are formally represented by N ▶attributes, which are either discrete (symbolic) or numerical (continuous). A discrete attribute $A_n, n = 1 \ldots, N$, comprises $J(n)$ distinct values $V_1, \ldots, V_{J(n)}$. A numerical attribute may attain any value from a continuous interval. The symbolic/logical ML algorithms usually process the numerical attributes by ▶discretization/fuzzification procedures, either on-line or off-line; see e.g., Bruha and Berka (2000).

An example (object, observation) can thus be expressed as an N-tuple $\mathbf{x} = [x_1, \ldots, x_N]$, involving N attribute values. A *selector* S_n is defined as an attribute-value pair of the form $x_n = V_j$, where V_j is the jth value of the attribute A_n (or the jth interval of a numerical attribute A_n).

To process missing values, we should know in advance (for $r = 1, \ldots, R, n = 1, \ldots, N, j = 1, \ldots, J(n)$):

- The overall *absolute* frequencies $F_{n,j}$ that express the number of examples exhibiting the value V_j for each attribute A_n

- The *class-sensitive absolute* frequencies $F_{r,n,j}$ that express the number of examples of the class C_r exhibiting the value V_j for each attribute A_n
- The *overall relative* frequencies $f_{n,j}$ of all known values V_j for each attribute A_n
- The *class-sensitive relative* frequencies $f_{r,n,j}$ of all known values V_j for each attribute A_n and for a given class C_r

The underlying idea for learning relies on the class distribution; i.e., the class-sensitive frequencies (overall and class-sensitive frequencies) are utilized. As soon as we substitute a missing value by a suitable one, we take the desired class of the example into consideration in order not to increase the noise in the data set. On the other hand, the overall frequencies are applied within classification.

We can now define the matching of an example **x** with a selector S_n by the so-called *matching ratio* $= 0$ if $x_n \neq V_j$

$$\mu(\mathbf{x}, S_n)\{= 1 \text{ if } x_n = V_j \quad (1)$$

$$\in [0;1] \text{ if } x_n \text{ is unknown (missing)}$$

A particular value of the matching ratio is determined by the selected routine (strategy) for missing value processing.

(I) Strategy Ignore: Ignore Missing Values: This strategy simply ignores examples (instances) with at least one missing attribute value before learning. Hence, no dilemma arises when determining matching ratios within learning. However, this approach does not contribute to any enhancement of processing of noisy or partly specified data.

As for classification, a missing value does not match any regular (known) value of a selector. Thus, a selector's matching ratio is equal to 0 for any missing value. Consequently, only a path of nodes in a decision tree or a decision rule that tests only the regular values during classification may succeed. If there is no such path of nodes in a decision tree or such a rule has not been found, then the default principle is applied; i.e., the instance with missing value(s) is classified as belonging to the majority class.

(U) Strategy Unknown: Unknown Value as a Regular One: An unknown (missing) value is considered as an additional attribute value. Hence, the number of values is increased by one for each attribute that depicts an unknown value in the training set. The matching ratio of a selector comprising the test of the selector S_n and an instance with the nth attribute missing is equal to 1 if this test (selector) is of the form $x_n =?$ where "?," represents the missing (unknown) value.

Note that selectors corresponding to the numerical (continuous) attributes are formed by tests $x_n \in V_j$ (where V_j is a numerical interval) or $x_n =?$.

(C) Strategy Common: The Most Common Value: This routine needs the class-sensitive absolute frequencies $F_{r,n,j}$ to be known before the actual learning process, and the overall frequencies $F_{n,j}$ before the classification. A missing value of a discrete attribute A_n of an example belonging to the class C_r is replaced by the *class-sensitive common* value, which maximizes the Laplacian formula $\frac{F_{r,n,j}+1}{F_{n,j}+R}$ over j for the given r and n. If the maximum is reached for more than one value of A_n, then the value V_j with the greatest frequency $F_{r,n,j}$ is selected as the common value.

A missing value within the classification is replaced by the *overall common* value, which maximizes $F_{n,j}$ over the subscript j. Consequently, the matching ratio yields 0 or 1, as every missing value is substituted by a concrete, known value.

The Laplacian formula utilized within the learning phase prefers those attribute values that are more predictive for a given class, contrary to the conventional "maximum frequency" scheme. For instance, let an attribute have two values: the value V_1 with the absolute frequencies [4, 2] for the classes C_1 and C_2, and the value V_2 with frequencies [3, 0] for these two classes. Then, when looking for the most common value of this attribute for the class C_1, the maximum frequency chooses the value V_1 as the most common value, whereas the Laplacian formula prefers the value V_2 as the more predictive for the class C_1.

(F) Strategy Fraction: Split into Proportional Fractions:

- Learning phase

The learning phase requires that the relative frequencies $f_{r,n,j}$ above the entire training set be known. Each example **x** of class C_r with a missing value of a discrete attribute A_n is substituted by a collection of examples

before the actual learning phase, as follows: the missing value of A_n is replaced by all known values V_j of A_n and C_r. The weight of each split example (with the value V_j) is

$$w_j = w(\mathbf{x}) * f_{r,n,j}, j = 1, \ldots, J(n)$$

where $w(\mathbf{x})$ is the weight of the original example $\mathbf{x}$. The weight is assigned by the designer of the training set and represents the designer's subjective judgment of the importance of that particular example within the entire training set. The matching ratio of the split examples is accomplished by (1) in a standard way.

If a training example involves more missing attribute values, then the above splitting is done for each missing value. Thus, the matching ratio may rapidly decrease. Therefore, this strategy, *Fraction,* should involve a methodology to avoid explosion of examples, so that only a predefined number of split examples with the largest weights is used for replacement of the original example.

- Classification phase

The routine *Fraction* works for each paradigm in a different way. In case of a decision tree, the example with a missing value for a given attribute A_n is split along all branches, with the weights equal to the overall relative frequencies $f_{n,j}$.

As for the decision rules, the matching ratio for a selector $x_n = V_j$ is defined by (1) as $\mu = f_{n,j}$ for a missing value of A_n. An instance with a missing value is tested with the conditions of all the rules, and is attached to the rule whose condition yields the maximum matching ratio – i.e., it is assigned to the class of this rule.

(A) Strategy Anyvalue: Any Value Matches: A missing value matches any existing attribute value, both in learning and classification. Therefore, a matching ratio μ of any selector is equal to 1 for any missing value.

It should be noticed that there is no uniform scheme in machine learning for processing the "any-value." In some systems, an example with a missing value for attribute A_n is replaced by $J(n)$ examples in which the missing value is in turn substituted by each regular value $V_j, j = 1, \ldots, J(n)$. In other systems, the missing "any-value" is substituted by any first attribute value involved in a newly generated rule when covered examples are being removed from the training set; see Bruha and Franek (1996) for details.

(Ran) Strategy Random: Substitute by Random Value A missing value of an attribute A_n is substituted by a randomly selected value from the set of its values $V_j, j = 1, \ldots, J(n)$. In case of the numerical attributes, the process used in the routine *Common* is first applied, i.e., the entire numerical range is partitioned into a pre-specified number of equal-length intervals. A missing value of the numerical attribute is then substituted by the mean value of a randomly selected interval.

At least two possibilities exist in the random procedure. Either

- A value is randomly chosen according to the uniform distribution – i.e., all the values have the same chance
- A value is chosen in conformity with the value distribution – i.e., the most frequent value has the greatest chance of being selected

To illustrate the difference of the strategies Anyvalue and Random, consider this scheme. Let the attribute A have three possible values, V_1, V_2, V_3 with the relative distribution [0.5, 0.3, 0.2]. (Here, of course, we consider class-sensitive distribution for the learning phase, overall one for classification.)

Strategy Anyvalue for TDIDT replaces the missing value A =? by each possible value $A = V_j$, $j = 1, 2, 3$, and these selectors (attribute-value pairs) are utilized for selecting a new node (during learning), or pushed down along an existing decision tree (classification).

Strategy Anyvalue for covering algorithms: if the corresponding selector in a complex is for example, $A = V_3$ then the selector A =? in an instance is replaced by $A = V_3$, so that the matching always succeeds.

Let the pseudo-random number be for example, 0.4 in the strategy Random. Then, in the first case – i.e., uniform distribution (one can consider the relative distribution has been changed to [0.33, 0.33, 0.33]) – the missing value A =? is replaced by $A = V_2$. In the second possibility – i.e., the actual distribution – the missing value is replaced by $A = V_1$.

(M) Strategy Meta Fill In: Use Another Learning Topology for Substitution: This interesting strategy utilizes another ML algorithm in order to fill in the missing attribute values. This second (or *meta*) learning algorithm uses the remaining attribute values of a given example (instance, observation) for determining (inducing) the missing value of the attribute A_n. There are several approaches to this strategy.

The first one was designed by Breiman; it uses a *surrogate split* in order to determine the missing attribute value. We can observe that a surrogate attribute has the highest correlation with the original one.

Quinlan (1989) was the first to introduce the meta-fill-in strategy; in fact, this method was proposed by A. Shapiro during their private communication. It builds a decision tree for each attribute that attempts to derive a value of the attribute with a missing value for a given instance in terms of the values of other attributes of the given instance.

Lakshminarayan et al. (1996) introduced a more robust approach where a ML technique (namely, C4.5) is used to fill in the missing values.

Ragel and Cremilleux (1998) developed a fill-in strategy by using the association rules paradigm. It induces a set of association rules according to the entire training set. This method is able to efficiently process the missing attribute values.

Missing Value Processing Techniques in Various ML Paradigms

As mentioned above, various missing value processing techniques have been embedded into various ML paradigms. We introduce four such systems.

Quinlan (1986, 1989) applied missing value techniques into ID3, the most famous TDIDT (decision tree inducing) algorithm. His list exhibits two additional routines that were not discussed above:

- The evaluation of an attribute uses the routines *I*, *C*, *M*, and *R* (i.e., reduce the apparent information gain from evaluating an attribute by the proportion of training examples with the missing value for this attribute)
- When partitioning a training set using the selected attribute, the routines *I*, *U*, *C*, *F*, *A*, *M* were used
- The classification phase utilizes the strategies *U*, *C*, *F*, *M*, and *H* (i.e., halt the classification and assign the instance to the most likely class)

Quinlan then combined the above routine into triples each representing a different overall strategy; however, not all the possible combinations of these routines make sense.

His experiments revealed that the strategies starting with *R* or *C* behave reasonably accurately among them the strategy *RFF* is the best. Brazdil and Bruha (1992) improved this strategy for partitioning a training set. They combined the strategies *U* and *F*; therefore, they call it *R(UF)(UF)* strategy.

Bruha and Franek (1996) discusses the embedding of missing value strategies into the covering algorithm CN4 (Bruha and Kockova 1994), a large extension of the well-known CN2 (Clark and Niblett 1989). A condition of a decision rule has the form:

$$\text{Cmplx} = S_{q_1} \& \ldots \& S_{q_M}$$

where S_{qm}, $m = 1, \ldots, M$, is the mth selector testing the jth value V_j of the q_mth attribute, (i.e., exhibiting the form $x_{qm} = V_j$). For the purposes of processing missing values, we need to define the *matching ratio* of the example **x** and the rule's condition *Cond*. (Bruha and Franek 1996) uses two definitions:

- The product of matching ratios of its selectors:

$$\mu(x, \text{Cmplx}) = w(x) \prod_{m=1}^{M} \mu(x, S_{qm}) \qquad (2)$$

- or their average:

$$\mu(x, \text{Cmplx}) = \frac{w(x)}{M} \sum_{m=1}^{M} \mu(x, S_{qm}), \qquad (3)$$

where $w(\mathbf{x})$ is the weight of the example **x** (1 by default), and μ on the right-hand side is the selector's matching ratio (1).

The Naive Bayes algorithm can process missing attribute values in a very simple way, because the probabilities it works with are, in fact, the relative frequencies discussed above: the class-sensitive relative frequencies $f_{r,n,j}$ (for the learning phase) and the overall

relative frequencies $f_{n,j}$ (for the purposes of classification). When learning relative frequencies, all strategies can by applied. Only routine Fraction is useless because it copies the distribution of the rest of a training set. When classifying an instance with missing value A_n =?, all strategies can be applied as well. Section Fraction substitutes this instances with $J(n)$ instances by each known attribute value, and each "fractioned" instance is attached by the weight $f_{n,j}$, and classified separately.

Ragel and Cremilleux (1998) present the missing value processing strategy for the algorithm that induced ►association rules. Their algorithm uses a modified version of the routine *Ignore*. The instances with missing attribute values are not removed from the training database but the missing values are ignored (or "hidden").

The experiments with the above techniques for handling missing values have revealed the following. In both decision tree and decision rules inducing algorithms, the routine *Ignore* is evidently the worst strategy. An Interesting issue is that the association rule inducing algorithms use its modified version. In case of the decision tree inducing algorithms, the strategy *Fraction* is one of the best; however, the decision rules inducing algorithms found it not so efficient. The explanation for this fact is based on different ways of processing examples in these two paradigms: in TDIDT, all training examples are eventually incorporated into the decision tree generated by the learning algorithm; on the other hand, the covering paradigm algorithm generates rules that may not cover all of the examples from the training set (as some of the examples are found not to be representable).

Although the routine *Unknown* is one of the "winners" (at least in the rule inducing algorithms and Brazdil and Bruha (1992), it is not quite clear how one can interpret, on a philosophical as well as a semantic level, a branch in a decision tree or a decision rule that involves a selector with an attribute equal to "?" (missing value). Strategy Fraction can be faced by "problems": if an example /instance exhibits too many missing values, then this strategy generates too many "fractioned" examples with very negligible weights.

One can find out that each dataset has more or less its own "favorite" routine for processing missing attribute values. It evidently depends on the magnitude of noise and the source of unknownness in each dataset. The problem of a "favorite" strategy can be solved by various approaches. One possibility is to create a small "window" within a training set, and to check the efficiency of each strategy in this window, and then choose the most efficient one. Bruha (2003) discusses another possibility: investigating the advantages of utilizing the external background (domain-specific, expert) knowledge on an attribute hierarchical tree.

Also, the concept of the so-called ►meta-combiner (Fan, Chan & Stolfo, 1996) can be utilized. A learning algorithm processes a given training base for each strategy for missing values independently; thus, all the missing value strategies are utilized in parallel and the meta-classifier makes up its decision from the results of the base level (Bruha, 2004).

The above issue – i.e., selection or combination of various strategies for missing value processing – is an open field for future research.

Recommended Reading

Brazdil, P. B., & Bruha, I. (1992). A note on processing missing attribute values: A modified technique. *Workshop on Machine learning, Canadian Conference AI*, Vancouver.

Bruha, I. (2003). Unknown attribute value processing by domain-specific external expert knowledge. *7th WSEAS international conference on systems*, Corfu, Greek.

Bruha, I. (2004). Meta-learner for unknown attribute values processing: Dealing with inconsistency of meta-databases. *Journal of Intelligent Information Systems, 22*(1), 71–84.

Bruha, I., & Franek, F. (1996). Comparison of various routines for unknown attribute value processing: covering paradigm. *International Journal of Pattern Recognition and Artificial Intelligence, 10*(8), 939–955.

Bruha, I., & Berka, P. (2000). Discretization and fuzzification of numerical attributes in attribute-based learning. In P. S. Szczepaniak, P. J. G. Lisboa, & J. Kacprzyk (Eds.), *Fuzzy systems in medicine* (pp. 112–138). Physica, Springer.

Bruha, I., & Kockova, S. (1994). A support for decision making: Cost-sensitive learning system. *Artificial Intelligence in Medicine, 6*, 67–82.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*, 261–283.

Fan, D. W., Chan, P. K., & Stolfo, S. J. (1996). A comparative evaluation of combiner and stacked generalization. *Workshop integrating multiple learning models, AAAI*, Portland.

Grzymala-Busse, J. W. (2003). Rough set strategies to date with missing attribute values, *Proceedings of workshop on foundations and new directions in data mining, IEEE Conference on data mining* (pp. 56–63).

Karmaker, A., & Kwer, S. (2005). Incorporating an EM-approach for handling missing attribute-values in decision tree induction. *International Conference on Hybrid Intelligent Systems*, 6–11.

Kononenko, I. (1992). Combining decisions of multiple rules. In B. du Boulay & V. Sgurev (Eds.), *Artificial intelligence V: Methodology, systems, applications* (pp. 87–96). Elsevier.

Lakshminarayan, K. et al. (1996). Imputation of missing data using machine learning techniques. *Conference Knowledge Discovery in Databases (KDD-96)*, 140–145.

Li, J., & Cercone, N. (2006). Assigning missing attribute values based on rough sets theory. *IEEE international conference on granular computing* (pp. 31–37). Atlanta.

Long, W. J., & Zhang, W. X. (2004). A novel measure of compatibility and methods of missing attribute values treatment in decision tables. *International Conference on Machine Learning and Cybernetics*, 2356–2360.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.

Quinlan, J. R. (1989). Unknown attribute values in ID3. *Proceedings of International Workshop on Machine Learning*, 164–168.

Ragel, A., & Cremilleux, B. (1998). Treatment of missing values for association rules. *Lecture Notes in Computer Science, 1394*, 258–270.

Missing Values

►Missing Attribute Values

Mistake-Bounded Learning

►Online Learning

Mixture Distribution

►Mixture Model

Mixture Model

Rohan A. Baxter
Australian Taxation Office

Synonyms

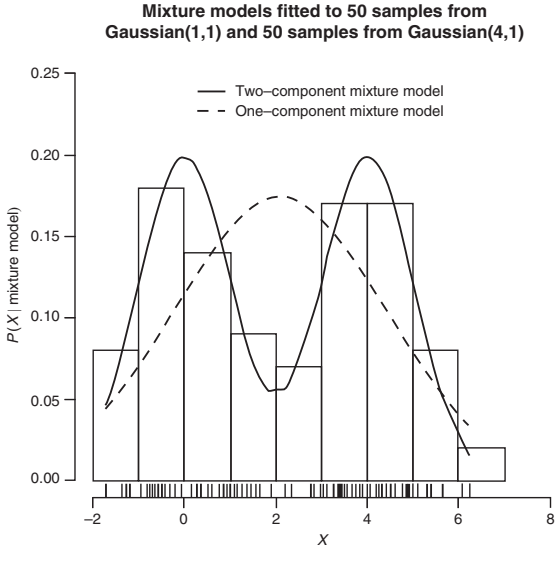
Finite mixture model; Latent class model; Mixture distribution; Mixture modeling

Definition

A mixture model is a collection of probability distributions or densities $D_1, \ldots, D_k$ and mixing weights or proportions $w_1, \ldots, w_k$, where k is the number of component distributions (Duda, Hart, & Stork, 2000; Lindsey, 1996; McLachlan & Peel, 2000).

The mixture model, $P(x \mid D_1, \ldots, D_k, w_1, \ldots, w_k) = \sum_{j=1}^{k} w_j P(x \mid D_j)$, is a probability distribution over the data conditional on the component distributions of the mixture and their mixing weights. It can be used for density estimation, model-based clustering or unsupervised learning, and classification.

Figure 1 shows one-dimensional data plotted along the x-axis with tick marks and a histogram of that data. The probability densities of two mixture models fitted to that data are then shown. The one-component mixture model is a Gaussian density with mean around 2 and standard deviation of 2.3. The two-component mixture model has one component with mean around 0 and the other with mean around 4, which reflects how these simple example data was artificially generated. This model can be used for clustering by considering each of its components as a cluster and assigning cluster membership based on the relative probability of a data item belonging to that component. Data less than 2 will have higher probability of belonging to the Gaussian with mean 0 component.



Mixture Model. Figure 1. Mixture model example for one-dimensional data

Motivation and Background

Mixture models are easy and convenient to apply. They trade off good power in data representation with relative ease in building the models. When used in clustering, a mixture model will have a component distribution covering each cluster, while the mixing weights reflect the relative proportion of a cluster's population. For example, a two-component mixture model of seal skull lengths from two different seal species may have one component with relative proportion 0.3 and the other with 0.7 reflecting the relative frequency of the two components.

Estimation

In order to use mixture models, the following choices need to be made by the modeler or by the mixture model software, based on the characteristics of a particular problem domain and its datasets:

1. The type of the component distributions (e.g., Gaussian, multinomial etc.)
2. The number of component distributions, k
3. The parameters for the component distributions (e.g., a one-dimensional Gaussian has a mean and standard deviation as parameters, a higher-dimensional Gaussian has a mean vector and covariance matrix as parameters)
4. Mixing weights, w_i
5. (Optional) component labels, c_j for each datum x_j, where $j = 1 \ldots n$ and n is the number of data

The fifth item above, component labels, are optional, because they are only used in latent class mixture model frameworks where a definite component membership is part of the model specification. Other mixture model frameworks use probabilistic membership of each datum to each component distribution and so do not need explicit component labels.

The most common way of fitting distribution parameters and mixture weights is to use the expectation-maximization (EM) algorithm to find the maximum likelihood estimates. The EM algorithm is an iterative algorithm that, starting with initial guesses of parameter values, computes the mixing weights (the expectation step). The next step is to then compute the parameter values based on these weights (the maximization step). The Expectation and Maximization steps iterate and convergence is assured (Redner & Walker, 2004). However, there is no guarantee that a global optimum has been found and so a number of random restarts may be required to find what other optima exist (Xu & Jordan, 1996).

As an alternative to random restarts, a good search strategy can be used to modify the current best solution, perhaps by choosing to split, merge, delete, or add, component distributions at random. This can also be a way to explore mixture models with different number of components (Figueiredo & Jain, 2002).

Since mixture models are a probabilistic model class, besides EM, other methods such as Bayesian methods or methods for graphical models can be used. These include Markov Chain Monte Carlo inference and Variational learning (Bishop, 2006).

Choosing the Number of Components

The number of components in a mixture model is often unknown when used for clustering real-world data. There have been many methods for choosing the number of components. The global maximum for maximum likelihood chooses a component for every data item, which is usually undesirable. Criteria based on information theory or Bayesian model selection choose reasonable numbers of components in many domains (McLachlan & Peel, 2000, Chap. 6, 5). There is no universally accepted method, because there is no universally accepted optimality criteria for clustering or density estimation. Use of an infinite mixture model, by using an infinite number of components, is one way to avoid the number of components problem (Rasmussen, 2000).

Types of Component Distributions

Besides Gaussian, other distributions can be used such as Poisson (for count data), von Mises (for data involving directions or angles), and Weibull. Heavy-tailed distributions require particular care, because standard estimation may not work when mean or variance is infinite (Dasgupta, Hopcroft, Kleinberg, & Sandler, 2005).

Another commonly needed variation is a mixture model to handle a mix of continuous and categorical features (McLachlan & Peel, 2000). For example, a binomial distribution can be used to model male/female

gender proportions and Gaussian to model length for data relating to a seal species sample.

A further extension is to allow components to depend on covariates, leading to mixtures of regression models (McLachlan & Peel, 2000). This leads to models such as mixtures of experts and hierarchical mixtures of experts (Bishop, 2006; McLachlan & Peel, 2000), which are flexible models for nonlinear regression. The combination of mixture models with Hidden Markov models allows the modeling of dependent data (McLachlan & Peel, 2000).

Large Datasets

The EM algorithm can be modified to find mixture models for very large datasets (Bradley, Reina, & Fayyad, 2000). The modification allows for a single scan of the data and involves identifying compressible regions of the data.

Theory

A key issue for mixture models is learnability (Chaudri, 2010). The more the component distributions overlap, the harder they are to learn. Higher-dimensional data also makes learning harder. Sometimes, these problems can be overcome by increasing the data quantity, but, in extremely hard cases, this will not work (Srebo, Shakhnarovich, & Roweis, 2006; Xu & Jordan, 1996).

Another issue is the relationship between adequate sample size and the number of components. A pragmatic policy is to set minimum mixing weights for component distributions. For example, for a dataset of size 100, if mixing weights are required to be greater than 0.1, this implies a maximum of ten components are possible to be learnt from the data with these parameter settings.

Applications

Mixture model software is often available in the clustering or density estimation parts of general statistical and data mining software. More specialized mixture modeling software for clustering data have included Autoclass (Autoclass, 2010), Snob (Snob, 2010), and mclust (Mclust, 2010).

Cross References

►Density-Based Clustering
►Density Estimation
►Gaussian Distribution
►Graphical Models
►Learning Graphical Models
►Markov Chain Monte Carlo
►Model-Based Clustering
►Unsupervised Learning

Recommended Reading

Autoclass, http://ti.arc.nasa.gov/project/autoclass/. Accessed 22 March 2010.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.

Bradley, P. S., Reina, C. A., & Fayyad, U. M. (2000). Clustering very large databases using EM mixture models. Fifteenth International Conference on Pattern Recognition, (ICPR-2000), (Vol. 2., p. 2076), Barcelona, Spain.

Chaudri, K. (2009). Learning mixture models. http://themachinelearningforum.org/index.php/overviews/34-colt-overviews/53-learning-mixture-models.html. Accessed 21 March 2010.

Dasgupta, A., Hopcroft, J., Kleinberg, J., & Sandler, M. (2005). On learning mixtures of heavy-tailed distributions. In: Proceedings of the 46th Annual IEEE symposium on foundations of computer science, (FOCS '05), Pittsburgh, Pennsylvania, USA.

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern classification* (2nd ed.). New York: Wiley-Interscience.

Figueiredo, M. A. T., & Jain, A. T. (2002). Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*, 381–396.

Lindsey, B. G. (1996). *Mixture models: Theory, geometry and applications*. Hayward, CA: IMS.

McLachlan, G. J., & Peel, D. (2000). *Finite mixture models*. New York: Wiley.

Mclust, http://www.stat.washington.edu/mclust/. Accessed 22 March 2010.

Rasmussen, C. E. (2000). The infinite Gaussian mixture model, NIPS 12 (pp. 554–560). Cambridge, MA: MIT Press.

Redner, R. A., & Walker, H. F. (2004). Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review*, 26, 195–239.

Snob, http://www.datamining.monash.edu.au/software/snob/. Accessed 22 March 2010.

Srebo, N., Shakhnarovich, G., & Roweis, S. (2006). An investigation of computational and informational limits in Gaussian mixture modeling. In Proceedings of the 23rd international conference on machine learning (ICML 2006), Pittsburgh, Pennsylvania.

Xu, L., & Jordan, M. I. (1996). On convergence properties of the EM algorithm for Gaussian mixtures. *Neural Computation*, *8*, 129–151.

Mixture Modeling

►Mixture Model

Mode Analysis

►Density-Based Clustering

Model Evaluation

Geoffrey I. Webb
Monash University, Victoria, Australia

Model evaluation is the process of assessing a property or properties of a ►model.

Motivation and Background

It is often valuable to assess the efficacy of a model that has been learned. Such assessment is frequently relative—an evaluation of which of several alternative models is best suited to a specific application.

Processes and Techniques

There are many metrics by which a model may be assessed. The relative importance of each metric varies from application to application.

The primary considerations often relate to predictive efficacy—how useful will the predictions be in the particular context it is to be deployed. Measures relating to predictive efficacy include ►Accuracy, ►Lift, ►Mean Absolute Error, ►Mean Squared Error, ►Negative Predictive Value, ►Positive Predictive Value, ►Precision, ►Recall, ►Sensitivity, ►Specificity, and various metrics based on ►ROC analysis.

Computational issues may also be important, such as a model's size or its execution time.

In many applications one of the most important considerations is the ease with which the model can be understood by the users or how consistent it is with the users' prior beliefs and understanding of the application domain.

When assessing the predictive efficacy of a model learned from data, to obtain a reliable estimate of its likely performance on new data, it is essential that it is not assessed by considering its performance on the data from which it was learned. A learning algorithm must interpolate appropriate predictions for regions of the ►instance space that are not included in the training data. It is probable that the inferred model will be more accurate for those regions represented in the training data than for those that are not, and hence predictions are likely to be less accurate for instances that were not included in the training data. Estimates that have been computed on the training data are called ►*resubstitution estimates*. For example, the error of a model on the training data from which it was learned is called resubstitution error.

Algorithm evaluation techniques such as ►cross-validation, ►holdout evaluation, and ►bootstrap sampling are designed to provide more reliable estimates of the accuracy of the models learned by an algorithm than would be obtained by assessing them on the training data.

Cross References

►Algorithm Evaluation
►Overfitting
►ROC Analysis

Recommended Reading

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. New York: Springer.
Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.
Witten, I. H., Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.

Model Selection

Model selection is the process of choosing an appropriate mathematical model from a class of models.

Model Space

►Hypothesis Space

Model Trees

Luís Torgo
University of Porto,
Porto, Portugal

Synonyms

Functional trees; Linear regression trees; Piecewise linear models

Definition

Model trees are supervised learning methods that obtain a type of tree-based ▶Regression model, similar to ▶Regression Trees, with the particularity of having functional models in the leaves instead of constants. These methods address multiple regression problems. In these problems we are usually given a training sample of n observations of a target continuous variable Y and of a vector of k predictor variables, $\mathbf{x} = X_1, \cdots, X_k$. Model trees provide an approximation of an unknown regression function $Y = f(\mathbf{x}) + \varepsilon$ with $Y \in R$ and ε, a normally distributed noise component with mean 0 and σ^2 variance. The leaves of these trees usually contain ▶linear regression models, although some works also consider other types of models.

Motivation and Background

Model trees are motivated by the purpose of overcoming some of the known limitations of regression trees caused by their piecewise constant approximation. In effect, by using constants at the leaves, regression trees provide a coarse grained function approximation leading to poor accuracy in some domains. Model trees try to overcome this by using more complex models on the leaves. Trees with linear models in the leaves were first considered in Breiman and Meisel (1976) and Friedman (1979). Torgo (1997) has extended the notion of model trees to other types of models in the tree leaves, namely, kernel regression, later extended to other types of local regression models (Torgo, 1999, 2000). The added complexity of the models used in the leaves increases the computational complexity of model trees when compared to regression trees, and also decreases their interpretability. In this context, several works Chaudhuri, Huang, Loh, & Yao (1994); Dobra & Gehrke (2002); Loh (2002); Malerba, Appice, Ceci, & Monopoli (2002); Natarajan & Pednault (2002); Torgo (2002); Malerba, Esposito, Ceci, & Appice (2004); Potts & Sammut (2005); Vogel, Asparouhov, & Scheffer (2007) have focused on obtaining model trees in a computationally efficient form.

Structure of Learning System

Approaches to model trees can be distinguished along two dimensions: the criterion used to select the best splits at each node, that is, the criterion guiding the partitioning obtained by the tree; and the type of models used in the leaves. The choices along the first dimension are mainly driven by considerations of computational efficiency. In effect, the selection of the best split node involves evaluating many candidate splits. The evaluation of a binary split (the most common splits in tree-based models) consists in calculating the error reduction produced by the split, that is,

$$\Delta(s,t) = Err(t) - \left(\frac{n_{t_L}}{n_t} \times Err(t_L) + \frac{n_{t_R}}{n_t} \times Err(t_R) \right) \quad (1)$$

where t is a tree node with sub-nodes t_L and t_R originated by the split test s, while n_t, n_{t_L}, n_{t_R} are the cardinalities of the respective sets of observations on each of these nodes, and $Err()$ is a function that estimates the error on a node being defined as,

$$Err(t) = \frac{1}{n_t} \sum_{\langle \mathbf{x}_i, y_i \rangle \in D_t} (y_i - g(D_t))^2 \quad (2)$$

where D_t is the sample of cases in node t, n_t is the cardinality of this set, and $g(D_t)$ is a function of the cases in node t.

In standard regression trees the function $g()$ is the average of the target variable Y, that is, $\frac{1}{n_t} \sum_{\langle \mathbf{x}_i, y_i \rangle \in D_t} y_i$. This corresponds to assuming a constant model on each leaf of the tree. The evaluation of each candidate split requires obtaining the models at the respective left and right branches (Eq. 1). If this model is an average, rather efficient incremental algorithms can be used to evaluate all candidate splits. On the contrary, if $g()$ is a ▶linear regression model or even other more complex models, this evaluation is not so simple and it is computationally very demanding, as a result of which systems that use this strategy (Karalic, 1992) become impractical for

large problems. In this context, several authors have adopted the alternative of growing the trees assuming constant values in the leaves and then fitting the complex models on each of the obtained leaves (e.g., Quinlan, 1992; Torgo, 1997, 1999, 2000). This only requires fitting as many models as there are leaves in the final tree. The main drawback of this approach is that the splits for the tree nodes are selected assuming the leaves will have averages instead of the models that in effect will be used. This may lead to splits that are suboptimal for the models that will fit on each leaf (Malerba et al., 2002, 2004). Several authors have tried to maintain the consistency of the split selection step with the models used in the leaves by proposing efficient algorithms for evaluating the different splits. In Malerba et al. (2002, 2004) linear models are obtained in a stepwise manner during tree growth. In Chaudhuri et al. (1994), Loh (2002), and Dobra and Gehrke (2002) the computational complexity is reduced by transforming the original regression problem into a classification problem. In effect, the best split is chosen by looking at the distribution of the sign of the residuals of a linear model fitted locally. In Natarajan and Pednault (2002); Torgo (2002); Vogel et al. (2007) the problem is addressed by proposing more efficient algorithms to evaluate all candidate splits. Finally, Potts and Sammut (2005) proposes an incremental algorithm to obtain model trees that fights the complexity of this task by imposing a limit on the number of splits that are considered for each node.

The most common form of model used in leaves is ►linear regression. Still, there are systems considering kernel models (Torgo, 1997), local linear models (Torgo, 1999), and partial linear models (Torgo, 2000). These alternatives provide smoother function approximation, although with increased computational costs and less interpretable models.

►Pruning in model trees does not bring any additional challenges when compared to standard regression trees and so similar methods are used for this over-fitting avoidance task. The same occurs with the use of model trees for obtaining predictions for new test cases. Each case is "dropped-down" the tree from the root node, following the branches according to the logical tests in the nodes, till a leaf is reached. The model in this leaf is used to obtain the prediction for the test case.

Cross References

►Random Forests
►Regression
►Regression Trees
►Supervised Learning
►Training Sample

Recommended Reading

Breiman, L., & Meisel, W. S. (1976). General estimates of the intrinsic variability of data in nonlinear regression models. *Journal of the American Statistical Association, 71*, 301–307.

Chaudhuri, P., Huang, M., Loh, W., & Yao, R. (1994). Piecewise-polynomial regression trees. *Statistica Sinica, 4*, 143–167.

Dobra, A., & Gehrke, J. E. (2002). Secret: A scalable linear regression tree algorithm. In *Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining*. Edmonton, Alberta, Canada: ACM.

Friedman, J. (1979). A tree-structured approach to nonparametric multiple regression. In T. Gasser & M. Rosenblatt (Eds.), *Smoothing techniques for curve estimation. Lecture notes in mathematics* (Vol. 757, pp. 5–22). Berlin/Heidelberg: Springer.

Karalic, A. (1992). Employing linear regression in regression tree leaves. In *Proceedings of ECAI-92*. New York, NY, USA: John Wiley & Sons, Inc.

Loh, W. (2002). Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica, 12*, 361–386.

Malerba, D., Appice, A., Ceci, M., & Monopoli, M. (2002). Trading-off local versus global effects of regression nodes in model trees. In *ISMIS '02: Proceedings of the 13th international symposium on foundations of intelligent systems* (pp. 393–402). Springer.

Malerba, D., Esposito, F., Ceci, M., & Appice, A. (2004). Top-down induction of model trees with regression and splitting nodes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 26*(5), 612–625.

Natarajan, R., & Pednault, E. (2002). Segmented regression estimators for massive data sets. In *Proceedings of the second SIAM international conference on data mining (SDM'02)*. Arlington, VA, USA: SIAM.

Potts, D., & Sammut, C. (2005). Incremental learning of linear model trees. *Machine Learning, 61*(1–3), 5–48.

Quinlan, J. (1992). Learning with continuous classes. In Adams & Sterling (Ed.), *Proceedings of AI'92* (pp. 343–348). World Scientific.

Torgo, L. (1997). Functional models for regression tree leaves. In D. Fisher (Ed.), *Proceedings of the 14th international conference on machine learning*. San Francisco, CA, USA: Morgan Kaufmann.

Torgo, L. (1999). *Inductive learning of tree-based regression models*. PhD thesis, Faculty of Sciences, University of Porto.

Torgo, L. (2000). Partial linear trees. In P. Langley (Ed.), *Proceedings of the 17th international conference on machine learning (ICML 2000)* (pp. 1007–1014). San Francisco, CA, USA: Morgan Kaufmann.

Torgo, L. (2002). Computationally efficient linear regression trees. In K. Jajuga, A. Sokolowski, & H. Bock (Eds.), *Classification, clustering and data analysis: Recent advances and applications (Proceedings of IFCS 2002). Studies in classification, data analysis, and knowledge organization* (pp. 409–415). Heidelberg/New York: Springer.

Vogel, D., Asparouhov, O., & Scheffer, T. (2007). Scalable look-ahead linear regression trees. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 757–764). ACM.

Model-Based Clustering

Arindam Banerjee, Hanhuai Shan
University of Minnesota, Minneapolis, MN, USA

Definition

Model-based clustering is a statistical approach to data clustering. The observed (multivariate) data is assumed to have been generated from a finite mixture of component models. Each component model is a probability distribution, typically a parametric multivariate distribution. For example, in a multivariate Gaussian mixture model, each component is a multivariate Gaussian distribution. The component responsible for generating a particular observation determines the cluster to which the observation belongs. However, the component generating each observation as well as the parameters for each of the component distributions are unknown. The key learning task is to determine the component responsible for generating each observation, which in turn gives the clustering of the data. Ideally, observations generated from the same component are inferred to belong to the same cluster. In addition to inferring the component assignment of observations, most popular learning approaches also estimate the parameters of each component in the process. The strength and popularity of the methods stem from the fact that they are applicable for a wide variety of data types, such as multivariate, categorical, sequential, etc., as long as suitable component generative models can be constructed. Such methods have found applications in several domains such as text clustering, image processing, computational biology, and climate sciences.

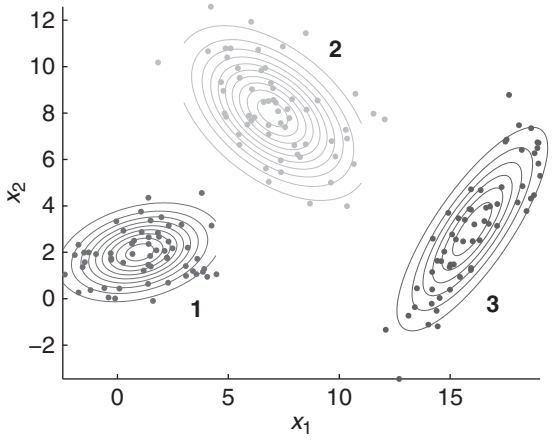
Structure of Learning System

Generative Model

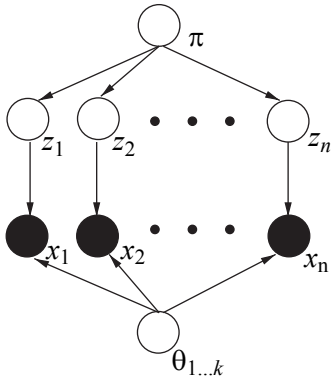
Let $X = \{x_1, \ldots, x_n\}$ be a dataset on which a k-clustering is to be performed. Let $p(x|\theta_1), \ldots, p(x|\theta_k)$ be k distributions which form the components of the mixture model from which the observed data is assumed to have been generated, and let $\pi = (\pi_1, \ldots, \pi_k)$ denote a prior distribution over the components. Then $\Theta = (\pi, \theta)$ constitutes the (unknown) parameters of the generative mixture model, where $\theta = \{\theta_1, \ldots, \theta_k\}$ and $\pi = \{\pi_1, \ldots, \pi_k\}$.

Given the model, an observation is assumed to be generated by the following two-step process: (1) randomly pick a component following the discrete distribution π over the components, i.e., the hth component is chosen with the probability of π_h; (2) the observation is sampled from the component distribution, e.g., if the hth component was chosen, we draw a sample $x \sim p(x|\theta_h)$. Each observation is assumed to be statistically independent so that they are all generated independently following the same two-step process.

Figure 1 gives an example of data drawn from a mixture of three ($k = 3$) 2-dimentional multivariate Gaussians. In the example, the discrete distribution over the component Gaussians is given by $\pi = (0.2, 0.3, 0.5)$. The parameter set $\theta_h, h = 1, 2, 3$ for any individual multivariate Gaussian consists of the mean vector μ_h and the covariance matrix Σ_h. For the example, we have $\mu_1 = [1, 2]$, $\mu_2 = [7, 8]$, $\mu_3 = [16, 3]$, and $\Sigma_1 = \begin{bmatrix} 3 & 0.5196 \\ 0.5196 & 1 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 4 & -1.7321 \\ -1.7321 & 3 \end{bmatrix}$, $\Sigma_3 = \begin{bmatrix} 3 & 3.0984 \\ 3.0984 & 5 \end{bmatrix}$.



Model-Based Clustering. Figure 1. Three 2-dimensional Gaussians



Model-Based Clustering. Figure 2. Bayesian network for a finite mixture model

The generative process could be represented as a Bayesian network as shown in Fig. 2, where the arrows denote the dependencies among variables/parameters. In the Bayesian network, (π, θ) are the parameters of the mixture model, x_i are the observations and z_i are the latent variables corresponding to the component which generates $x_i, i = 1, \ldots, n$. To generate an observation x_i, the model first samples a latent variable z_i from the discrete distribution π, and then samples the observation x_i from component distribution $p(x|\theta_{z_i})$.

Learning

Given a set of observations $X = \{x_1, \ldots, x_n\}$ assumed to have been generated from a finite mixture model, the learning task is to infer the latent variables z_i for each observation as well as estimate the model parameters $\Theta = (\pi, \theta)$. In the Gaussian mixture model example, the goal would be to infer the component responsible for generating each observation and estimate the mean and covariance for each component Gaussian as well as the discrete distribution π over the three Gaussians. After learning model parameters, the posterior probability $p(h|x_i, \Theta)$ of each observation x_i belonging to each component Gaussian gives a (soft) clustering for the observation.

The most popular approach for learning mixture models is based on maximum likelihood estimation (MLE) of the model parameters. In particular, given the set of observations X, one estimates the set of model parameters which maximizes the (log-)likelihood of observing the entire dataset X. For the finite mixture model, the likelihood of observing any data point x_i is given by

$$p(x_i|\Theta) = \sum_{h=1}^{k} \pi_h p(x_i|\theta_h) \,. \tag{1}$$

Since the data points in X are assumed to be statistically independent, the log-likelihood. (In practice, one typically focuses on maximizing the log-likelihood $\log p(X|\Theta)$ instead of the likelihood $p(X|\Theta)$ due to both numerical stability and analytical tractability). of observing the entire dataset X is given by

$$\begin{aligned} \log p(X|\Theta) &= \log \left(\prod_{i=1}^{n} p(x_i|\pi, \theta) \right) \\ &= \sum_{i=1}^{n} \log \left(\sum_{h=1}^{k} \pi_h p(x_i|\theta_h) \right) \,. \end{aligned} \tag{2}$$

A direct application of MLE is difficult since the log-likelihood cannot be directly optimized with respect to the model parameters. The standard approach to work around this issue is to use the expectation maximization (EM) algorithm which entails maximizing a tractable lower bound to the log-likelihood $\log p(X|\Theta)$. To this end, a latent variable z_i is explicitly introduced for each x_i to inform the component that x_i is generated from. The joint distribution of (x_i, z_i) is $p(x_i, z_i|\pi, \theta) = \pi_{z_i} p(x_i|\theta_{z_i})$. Let $Z = \{z_1, \ldots, z_n\}$ denote the set of latent variables corresponding to $X = \{x_1, \ldots, x_n\}$. The joint log-likelihood of (X, Z) then becomes

$$\begin{aligned} \log p(X, Z|\Theta) &= \sum_{i=1}^{n} \log p(x_i, z_i|\Theta) \\ &= \sum_{i=1}^{n} (\log \pi_{z_i} + \log p(x_i|\theta_{z_i})) \,. \end{aligned} \tag{3}$$

For a given set Z, it is easy to directly optimize (3) with respect to the parameters $\Theta = (\pi, \theta)$. However, Z is actually a random vector whose exact value is unknown. Hence, the log-likelihood $\log p(X, Z|\Theta)$ is a random variable depending on the distribution of Z. As a result, EM focuses on optimizing the following lower bound based on the expectation of $\log p(X, Z|\Theta)$ where the expectation is taken with respect to some distribution $p(Z)$ over the latent variable set Z. In particular, for any distribution $q(Z)$, we consider the lower bound

$$L(q, \Theta) = E_{Z \sim q}[\log p(X, Z|\theta)] + H(q(Z)) \,, \tag{4}$$

where the expectation on the first term is with respect to the posterior distribution $q(Z)$ and $H(q(Z))$ denotes the Shannon entropy of the latent variable set $Z \sim q(Z)$. A direct calculation shows that the difference between the true log-likelihood in (2) and the lower bound in (4) is exactly the relative entropy between $q(Z)$ and the posterior distribution $p(Z|X,\Theta)$, i.e.,

$$\log p(X|\Theta) - L(q,\Theta) = KL(q(Z)||p(Z|X,\Theta)) \geq 0 \quad (5)$$

$$\Rightarrow \log p(X|\Theta) \geq L(q,\Theta)\,, \quad (6)$$

where $KL(||)$ denotes the KL-divergence or relative entropy. As a result, when $q(Z) = p(Z|X,\Theta)$, the lower bound is exactly equal to the log-likelihood $\log p(X|\Theta)$. EM algorithms for learning mixture models work by alternately optimizing the lower bound $L(q,\Theta)$ over q and Θ. Starting with an initial guess $\Theta^{(0)}$ of the parameters, in iteration t such algorithms perform the following two steps:

E-step Maximize $L(q,\Theta^{(t-1)})$ with respect to $q(Z)$ to obtain

$$q^{(t)}(Z) = \underset{q(Z)}{argmax}\, L(q(Z),\Theta^{(t-1)}) = p(Z|X,\Theta^{(t-1)})\,. \quad (7)$$

M-step Maximize $L(q^{(t)},\Theta)$ with respect to Θ, i.e.,

$$\Theta^{(t)} = \underset{\Theta}{argmax}\, L(q^{(t)}(Z),\Theta)\,, \quad (8)$$

which is equivalent to

$$\Theta^{(t)} = \underset{\Theta}{argmax} \sum_{i=1}^{n} E_{z_i}[\log p(x_i,z_i|\Theta)]$$

since the second term in (4) does not depend on Θ.

Model-based clustering of multivariate data is often performed by learning a Mixture of Gaussians (MoG) using the EM algorithm. In a MoG model, the parameters corresponding to each component are the mean and covariance for each Gaussian given by (μ_h,Σ_h), $h = 1,\ldots,k$. For a given dataset X, the EM algorithm for learning MoG starts with an initial guess $\Theta^{(0)}$ for the parameters where $\Theta^{(0)} = \{(\pi_h^{(0)},\mu_h^{(0)},\Sigma_h^{(0)}), h = 1,\ldots,k\}$. At iteration t, the following updates are done:

E-step Update distributions over latent variables $z_i, i = 1,\ldots,n$ as

$$q^{(t)}(z_i = h) = p(z_i = h|x_i,\Theta^{(t-1)}) = \frac{\pi_h^{(t-1)} p(x_i|\mu_h^{(t-1)},\Sigma_h^{(t-1)})}{\sum_{h'=1}^{k} \pi_{h'}^{(t-1)} p(x_i|\mu_h^{(t-1)},\Sigma_{h'}^{(t-1)})}\,. \quad (9)$$

M-step Optimizing the lower bound over $\{(\pi_h,\mu_h,\Sigma_h), h = 1,\ldots,k\}$ yields

$$\pi_h^{(t)} = \frac{1}{n}\sum_{i=1}^{n} p(h|x_i,\Theta^{(t-1)})\,, \quad (10)$$

$$\mu_h^{(t)} = \frac{\sum_{i=1}^{n} x_i p(h|x_i,\Theta^{(t-1)})}{n\pi_h^{(t)}}\,, \quad (11)$$

$$\Sigma_h^{(t)} = \frac{\sum_{i=1}^{n}(x_i-\mu_h^{(t)})(x_i-\mu_h^{(t)})^T p(h|x_i,\Theta^{(t-1)})}{n\pi_h^{(t)}}\,. \quad (12)$$

The iterations are guaranteed to lead to monotonically non decreasing improvements of the lower bound $L(q,\Theta)$. The iterations are typically run till a suitable convergence criterion is satisfied. On convergence, one gets the estimates $\Theta = \{(\pi_h,\mu_h,\Sigma_h), h = 1,\ldots,k\}$ of the component parameters as well as the soft clustering $p(h|x_i,\Theta)$ of individual data points. The alternating maximization algorithm outlined above can get stuck in a local minima or saddle point of the objective function. In general, the iterations are not guaranteed to converge to a global optima. In fact, different initializations $\Theta^{(0)}$ of parameters can yield different final results. In practice, one typically tries a set of different initializations and picks the best among them according to the final value of the lower bound obtained. Extensive empirical research has gone into devising good initialization schemes for EM algorithm in the context of learning mixture models.

Recent years have seen progress in the design and analysis of provably correct algorithms for learning mixture models for certain well behaved distributions, where the component distributions are assumed to be

separated from each other in a well-defined sense. Such algorithms typically involve projecting data to a suitable lower-dimensional space where the components separate out and the clustering becomes simpler. One family of algorithms rely on random projections and are applicable to variety of problems including that of learning mixture of Gaussians. More recent developments include algorithms based on spectral projections and are applicable to any log-concave distributions.

Related Work

Model-based clustering is intimately related to a wide variety of centroid-based partitional clustering algorithms. In particular, the popular kmeans clustering algorithm can be viewed as a special case of learning mixture of Gaussians with a specific covariance structure. Given a dataset X, the kmeans problem is to find a partitioning $C = \{C_h, h = 1, \ldots, k\}$ of X such that the following objective is minimized:

$$J(C) = \sum_{h=1}^{k} \sum_{x \in C_h} \|x - \mu_h\|^2 ,$$

where μ_h is the mean of the points in C_h. Starting from an initial guess at the cluster means, the kmeans algorithm alternates between assigning points to the nearest cluster and updating the cluster means till convergence. Consider the problem of learning a mixture of Gaussians on X such that each Gaussian has a fixed covariance matrix $\Sigma_h = \beta I$, where I is the identity matrix and $\beta > 0$ is a constant. Then, as $\beta \to 0$, maximizing the scaled lower bound $\beta L(q, \Theta)$ corresponding to the mixture modeling problem becomes equivalent to minimizing the kmeans objective. Further, the EM algorithm outlined above reduces to the popular kmeans algorithm. In fact, such a reduction holds for a much larger class of centroid-based clustering algorithms based on Bregman divergences, which are a general class of divergence measures derived from convex function and have popular divergences such as squared Euclidean distance and KL-divergence as special cases. Centroid-based clustering with Bregman divergences can be viewed as a special case of learning mixtures of exponential family distributions with a reduction similar to the one from mixture of Gaussians to kmeans. Further, non linear clustering algorithms such as kernel kmeans can be viewed as a special case of learning mixture of Gaussians in a Hilbert space.

Recent years have seen generalizations of mixture models to mixed membership models and their non parametric extensions. Latent Dirichlet allocation is an example of such a mixed membership model for topic modeling in text corpora. The key novelty of mixed membership models is that they allow a different component proportions π_x for each observation x instead of a fixed proportion π as in mixture models. The added flexibility yields superior performance in certain problem domains.

Recommended Reading

Banerjee, A., Merugu, S., Dhillon, I., & Ghosh, J. (2005). Clustering with Bregman divergences. *Journal of Machine Learning Research, 6*, 1705–1749.

Bilmes, J. (1997). A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-02, University of Berkeley.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research, 3*, 993–1022.

Dasgupta, S. (1999). *Learning mixtures of Gaussians*. IEEE Symposium on foundations of Computer Science (FOCS). Washington, DC: IEEE Press.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological), 39*(1), 1–38.

Kannan, R., Salmasian, H., & Vempala, S. (2005). *The spectral method for general mixture models*. Conference on Learning Theory (COLT).

McLachlan, G. J., & Krishnan, T. (1996). *The EM algorithm and extensions*. NewYork: Wiley-Interscience.

McLachlan, G. J., & Peel, D. (2000). *Finite mixture models. Wiley series in probability and mathematical statistics: Applied probability and statistics section*. New York: Wiley.

Neal, R. M., & Hinton, G. E. (1998). A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan (Ed.), *Learning in graphical models* (pp. 355–368). Cambridge, MA: MIT Press.

Redner, R., & Walker, H. (1984). Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review, 26*(2), 195–239.

Model-Based Control

►Internal Model Control

Model-Based Reinforcement Learning

Soumya Ray[1], Prasad Tadepalli[2]
[1]Case Western Reserve University, Cleveland, OH, USA
[2]Oregon State University, Corvallis, OR, USA

Synonyms

Indirect reinforcement learning

Definition

Model-based Reinforcement Learning refers to learning optimal behavior indirectly by learning a model of the environment by taking actions and observing the outcomes that include the next state and the immediate reward. The models predict the outcomes of actions and are used in lieu of or in addition to interaction with the environment to learn optimal policies.

Motivation and Background

►Reinforcement Learning (RL) refers to learning to behave optimally in a stochastic environment by taking actions and receiving rewards (Sutton & Barto, 1998). The environment is assumed Markovian in that there is a fixed probability of the next state given the current state and the agent's action. The agent also receives an immediate reward based on the current state and the action. Models of the next-state distribution and the immediate rewards are referred to as "action models" and, in general, are not known to the learner. The agent's goal is to take actions, observe the outcomes including rewards and next states, and learn a policy or a mapping from states to actions that optimizes some performance measure. Typically the performance measure is the expected total reward in episodic domains, and the expected average reward per time step or expected discounted total reward in infinite-horizon domains.

The theory of ►Markov Decision Processes (MDPs) implies that under fairly general conditions, there is a stationary policy, i.e., a time-invariant mapping from states to actions, which maximizes each of the above reward measures. Moreover, there are MDP solution algorithms, e.g., value iteration and policy iteration (Puterman, 1994), which can be used to solve the MDP exactly given the action models. Assuming that the number of states is not exceedingly high, this suggests a straight-forward approach for model-based reinforcement learning. The models can be learned by interacting with the environment by taking actions, observing the resulting states and rewards, and estimating the parameters of the action models through maximum likelihood methods. Once the models are estimated to a desired accuracy, the MDP solution algorithms can be run to learn the optimal policy.

One weakness of the above approach is that it seems to suggest that a fairly accurate model needs to be learned over the entire domain to learn a good policy. Intuitively it seems that we should be able to get by without learning highly accurate models for suboptimal actions. A related problem is that the method does not suggest how best to explore the domain, i.e., which states to visit and which actions to execute to quickly learn an optimal policy. A third issue is one of scaling these methods, including model learning, to very large state spaces with billions of states.

The remaining sections outline some of the approaches explored in the literature to solve these problems.

Theory and Methods

Systems that solve MDPs using value-based methods can take advantage of models in at least two ways. First, with an accurate model, they can use offline learning algorithms that directly solve the modeled MDPs. Second, in an online setting, they can use the estimated models to guide exploration and action selection. Algorithms have been developed that exploit MDP models in each of these ways. We describe some such algorithms below.

Common approaches to solving MDPs given a model are value or policy iteration (Kaelbling, Littman, & Moore, 1996; Sutton & Barto, 1998). In these approaches, the algorithms start with a randomly initialized value function or policy. In value iteration, the algorithm loops through the state space, updating the value estimates of each state using Bellman backups, until convergence. In policy iteration, the algorithm calculates the value of the current policy and then loops through the state space, updating the current policy to be greedy with respect to the

backed up values. This is repeated until the policy converges.

When the model is unknown but being estimated as learning progresses, we could use value or policy iteration in the inner loop: after updating our current model estimate using an observed sample from the MDP, we could solve the updated MDP offline and take an action based on the solution. However, this is computationally very expensive. To gain efficiency, algorithms such as ▶Adaptive Real-time Dynamic Programming (ARTDP) (Barto, Bradtke, & Singh, 1995) and DYNA (Sutton, 1990) perform one or more Bellman updates using the action models after each real-world action and corresponding update to either a state-based or state-action-based value function. Other approaches, such as prioritized sweeping (Moore & Atkeson, 1993) and Queue-Dyna (Peng & Williams, 1993), have considered the problem of intelligently choosing which states to update after each iteration.

A different approach to discovering the optimal policy is to use algorithms that calculate the gradient of the utility measure with respect to some adjustable policy parameters. The standard policy gradient approaches that estimate the gradient from immediate rewards suffer from high variance due to the stochasticity of the domain and the policy. Wang and Dietterich propose a model-based policy gradient algorithm that alleviates this problem by learning a partial model of the domain (Wang & Dietterich, 2003). The partial model is solved to yield the value function of the current policy and the expected number of visits to each state, which are then used to derive the gradient of the policy in closed form. The authors observe that their approach converges in many fewer exploratory steps compared with model-free policy gradient algorithms in a number of domains including a real-world resource-controlled scheduling problem.

One of the many challenges in model-based reinforcement learning is that of efficient exploration of the MDP to learn the dynamics and the rewards. In the "Explicit Explore and Exploit" or E^3 algorithm, the agent explicitly decides between exploiting the known part of the MDP and optimally trying to reach the unknown part of the MDP (exploration) (Kearns & Singh, 2002). During exploration, it uses the idea of "balanced wandering," where the least executed action in the current state is preferred until all actions are executed a certain number of times. In contrast, the R-Max algorithm implicitly chooses between exploration and exploitation by using the principle of "optimism under uncertainty" (Brafman & Tennenholtz, 2002). The idea here is to initialize the model parameters optimistically so that all unexplored actions in all states are assumed to reach a fictitious state that yields maximum possible reward from then on regardless of which action is taken. Both these algorithms are guaranteed to find models whose approximate policies are close to the optimal with high probability in time polynomial in the size and mixing time of the MDP.

Since a table-based representation of the model is impractical in large state spaces, efficient model-based learning depends on compact parameterization of the models. Dynamic Bayesian networks offer an elegant way to represent action models compactly by exploiting conditional independence relationships, and have been shown to lead to fast convergence of models (Tadepalli & Ok, 1998). In some cases, choosing an appropriate prior distribution over model parameters can be important and lead to faster learning. In recent work, the acquisition of a model prior has been investigated in a multi-task setting (Wilson, Fern, Ray, & Tadepalli, 2007). In this work, the authors use a hierarchical Bayesian model to represent classes of MDPs. Given observations from a new MDP, the algorithm uses the model to infer an appropriate class (creating a new class if none seem appropriate). It then uses the distributions governing the inferred class as a prior to guide exploration in the new MDP. This approach is able to significantly speed up the rate of convergence to optimal policy as more environments are seen.

In recent work, researchers have explored the possibility of using approximate models coupled with policy gradient approaches to solve hard control problems (Abbeel, Quigley, & Ag, 2006). In this work, the approximate model is used to calculate gradient directions for the policy parameters. When searching for an improved policy, however, the real environment is used to calculate the utility of each intermediate policy. Observations from the environment are also used to update the approximate model. The authors show that their approach improves upon model-based algorithms which only used the approximate model while learning.

Applications

In this section, we describe some domains where model-based reinforcement learning has been applied.

Model-based approaches have been commonly used in RL systems that play two-player games (Baxter, Tridgell, & Weaver, 1998; Tesauro, 1995). In such systems, the model corresponds to legal moves in the game. Such models are easy to acquire and can be used to perform lookahead search on the game tree. For example, the TD-LEAF(λ) system (Baxter et al., 1998) uses the values at the leaves of an expanded game tree at some depth to update the estimate of the value of the current state. After playing a few hundred chess games, this algorithm was able to reach the play level of a US Master.

Model-based reinforcement learning has been used in a spoken dialog system (Singh, Kearns, Litman, & Walker, 1999). In this application, a dialog is modeled as a turn-based process, where at each step the system speaks a phrase and records certain observations about the response and possibly receives a reward. The system estimates a model from the observations and rewards and uses value iteration to compute optimal policies for the estimated MDP. The authors show empirically that, among other things, the system finds sensible policies and is able to model situations that involve "distress features" that indicate the dialog is in trouble.

It was shown that in complex real-world control tasks such as pendulum swing-up task on a real anthropomorphic robot arm, model-based learning is very effective in learning from demonstrations (Atkeson & Schaal, 1997). A model is learned from the human demonstration of pendulum swing-up, and an optimal policy is computed using a standard approach in control theory called linear quadratic regulation. Direct imitation of the human policy would not work in this case due to the small differences in the tasks and the imperfections of the robot controller. On the other hand, model-based learning was able to learn successfully from short demonstrations of pendulum swing up. However, on a more difficult swing-up task that includes pumping, model-based learning by itself was inadequate due to the inaccuracies in the model. They obtained better results by combining model-based learning with learning appropriate task parameters such as the desired pendulum target angle at an intermediate stage where the pendulum was at its highest point.

In more recent work, model-based RL has been used to learn to fly a remote-controlled helicopter (Abbeel, Coates, Quigley, & Ng, 2007). Again, the use of model-free approaches is very difficult, because almost any random exploratory action results in an undesirable outcome (i.e., a crash). To learn a model, the system bootstraps from a trajectory that is observed by watching an expert human fly the desired maneuvers. In each step, the system learns a model with the observed trajectory and finds a controller that works in simulation with the model. This controller is then tried with the real helicopter. If it fails to work well, the model is refined with the new observations and the process is repeated. Using this approach, the system is able to learn a controller that can repeatedly perform complex aerobatic maneuvers, such as flips and rolls.

Model-based RL has also been applied to other domains, such as robot juggling (Schaal & Atkeson, 1994) and job-shop scheduling (Zhang & Dietterich, 1995). Some work has also been done that compares model-free and model-based RL methods (Atkeson & Santamaria, 1997). From their experiments, the authors conclude that, for systems with reasonably simple dynamics, model-based RL is more data efficient, finds better policies, and handles changing goals better than model-free methods. On the other hand, model-based methods are subject to errors due to inaccurate model representations.

Future Directions

Representing and learning richer action models for stochastic domains that involve relations, numeric quantities, and parallel, hierarchical, and durative actions is a challenging open problem. Efficient derivation of optimal policies from such rich representations of action models is another problem that is partially explored in ►symbolic dynamic programming. Constructing good policy languages appropriate for a given action model or class of models might be useful to accelerate learning near-optimal policies for MDPs.

Cross References

►Adaptive Real-Time Dynamic Programming
►Autonomous Helicopter Flight Using Reinforcement Learning
►Bayesian Reinforcement Learning

►Efficient Exploration in Reinforcement Learning
►Symbolic Dynamic Programming

Recommended Reading

Abbeel, P., Coates, A., Quigley, M., & Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems* (Vol. 19, pp. 1–8). Cambridge, MA: MIT Press.

Abbeel, P., Quigley, M., & Ng, A. Y. (2006). Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on machine learning* (pp. 1–8). ACM Press, New York, USA.

Atkeson, C. G., & Santamaria, J. C. (1997). A comparison of direct and model-based reinforcement learning. In *Proceedings of the international conference on robotics and automation* (pp. 20–25). IEEE Press.

Atkeson, C. G., & Schaal, S. (1997). Robot learning from demonstration. In *Proceedings of the fourteenth international conference on machine learning* (Vol. 4, pp. 12–20). San Francisco: Morgan Kaufmann.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 72*(1), 81–138.

Baxter, J., Tridgell, A., & Weaver, L. (1998). TDLeaf(λ): Combining temporal difference learning with game-tree search. In *Proceedings of the ninth Australian conference on neural networks (ACNN'98)* (pp. 168–172).

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research, 2*, 213–231.

Kaelbling, L. P., Littman, M. L., & Moore, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning, 49*(2/3), 209–232.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning, 13*, 103–130.

Peng, J., & Williams, R. J. (1993). Efficient learning and planning within the dyna framework. *Adaptive Behavior, 1*(4), 437–454.

Puterman, M. L. (1994). *Markov decision processes: Discrete dynamic stochastic programming*. New York: Wiley.

Schaal, S., & Atkeson, C. G. (1994). Robot juggling: Implementation of memory-based learning. *IEEE Control Systems Magazine, 14*(1), 57–71.

Singh, S., Kearns, M., Litman, D., & Walker, M. (1999) Reinforcement learning for spoken dialogue systems. In *Advances in neural information processing systems* (Vol. 11, pp. 956–962). MIT Press.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning* (pp. 216–224). San Francisco: Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Tadepalli, P., & Ok, D. (1998). Model-based average-reward reinforcement learning. *Artificial Intelligence, 100*, 177–224.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM, 38*(3), 58–68.

Wang, X., & Dietterich, T. G. (2003). Model-based policy gradient reinforcement learning. In *Proceedings of the 20th international conference on machine learning* (pp. 776–783). AAAI Press.

Wilson, A., Fern, A., Ray, S., & Tadepalli, P. (2007). Multi-task reinforcement learning: A hierarchical Bayesian approach. In *Proceedings of the 24th international conference on machine learning* (pp. 1015–1022). Madison, WI: Omnipress.

Zhang, W., & Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the international joint conference on artificial intelligence* (pp. 1114–1120). Morgan Kaufman.

Modularity Detection

►Group Detection

MOO

►Multi-Objective Optimization

Morphosyntactic Disambiguation

►POS Tagging

Most General Hypothesis

Synonyms

Maximally general hypothesis

Definition

A hypothesis, h, is a most general hypothesis if it covers none of the negative examples and there is no other hypothesis h' that covers no negative examples, such that h is strictly more specific than h'.

Cross References

►Learning as Search

Most Similar Point

►Nearest Neighbor

Most Specific Hypothesis

Synonyms
Maximally specific hypothesis

Definition
A hypothesis, h, is a most specific hypothesis if it covers none of the negative examples and there is no other hypothesis h' that covers no negative examples, such that h is strictly more general than h'.

Cross References
►Learning as Search

Multi-Agent Learning I: Problem Definition

Yoav Shoham, Rob Powers
Stanford University,
Stanford, CA, USA

Definition
Multi-agent learning (MAL) refers to settings in which multiple agents learn simultaneously. Usually defined in a game theoretic setting, specifically in repeated games or stochastic games, the key feature that distinguishes multi-agent learning from single-agent learning is that in the former the learning of one agent impacts the learning of others. As a result, neither the problem definition for multi-agent learning nor the algorithms offered follow in a straightforward way from the single-agent case. In this first of two entries on the subject we focus on the problem definition.

Background
The topic of multi-agent learning (MAL henceforth) has a long history in game theory, almost as long as the history of game theory itself (Another more recent term for the area within game theory is *interactive learning*). In artificial intelligence (AI) the history of *single*-agent learning is of course as rich if not richer; one need not look further than this Encyclopedia for evidence. And while it is only in recent years that AI has branched into the multi-agent aspects of learning, it has done so with something of a vengeance. If in 2003 one could describe the AI literature on MAL by enumerating the relevant articles, today this is no longer possible. The leading conferences routinely feature articles on MAL, as do the journals (We acknowledge a simplification of history here. There is definitely MAL work in AI that predates the last few years, though the relative deluge is indeed recent. Similarly, we focus on AI since this is where most of the action is these days, but there are also other areas in computer science that feature MAL material; we mean to include that literature here as well).

While the AI literature maintains a certain flavor that distinguishes it from the game theoretic literature, the commonalities are greater than the differences. Indeed, alongside the area of mechanism design, and perhaps the computational questions surrounding solution concepts such as the Nash equilibrium, MAL is today arguably one of the most fertile interaction grounds between computer science and game theory. The key aspect of MAL, which ties the work together, and which distinguishes it from single-agent learning, is the fact that in MAL one cannot separate the process of learning from the process of teaching. The learning of one agent causes it to change its behavior; this causes other agents to adapt their behavior, which in turn causes the first agent to keep adapting too. Such reciprocal – or interactive – learning calls not only for different types of learning algorithms, but also for different yardsticks by which to evaluate learning. For this reason, the literature on MAL can be confusing. Not only do the learning techniques vary, but the goal of learning and the evaluation measures are diverse, and often left only implicit.

We will couch our discussion in the formal setting of *stochastic games* (a.k.a. *Markov games*). Most of the MAL literature adopts this setting, and indeed most of it focuses on the even more narrow class of *repeated games*. Furthermore, stochastic games also generalize *Markov decision problems* (MDPs), the setting from

which much of the relevant learning literature in AI originates. These are defined as follows.

A stochastic game can be represented as a tuple: $(N, S, \vec{A}, \vec{R}, T)$. N is a set of agents indexed $1, \ldots, n$. S is a set of n-agent stage games. $\vec{A} = A_1, \ldots, A_n$, with A_i the set of actions (or pure strategies) of agent i (note that we assume the agent has the same strategy space in all games; this is a notational convenience, but not a substantive restriction). $\vec{R} = R_1, \ldots, R_n$, with $R_i : S \times \vec{A} \to \mathcal{R}$ giving the immediate reward function of agent i for stage game S. $T : S \times \vec{A} \to \Pi(S)$ is a stochastic transition function, specifying the probability of the next stage game to be played based on the game just played and the actions taken in it.

We also need to define a way for each agent to aggregate the set of immediate rewards received in each state. For finitely repeated games we can simply use the sum or average, while for infinite games the most common approaches are to use either the limit average or the sum of discounted awards $\sum_{t=1}^{\infty} \delta^t r_t$, where r_t is the reward received at time t.

A repeated game is a stochastic game with only one stage game, while an MDP is a stochastic game with only one agent. (Note: While most of the MAL literature lives happily in this setting, we would be remiss not to acknowledge the literature that does not. Certainly, one could discuss learning in the context of extensive-form games of incomplete and/or imperfect information. Even farther afield, interesting studies of learning exist in large population games and evolutionary models, particularly *replicator dynamics (RD)* and *evolutionary stable strategies (ESS)*.)

What is there to learn in stochastic games? Here we need to be explicit about some aspects of stochastic games that were glossed over so far. Do the agents know the stochastic game, including the stage games and the transition probabilities? If not, do they at least know the specific game being played at each stage, or only the actions available to them? What do they see after each stage game has been played – only their own rewards, or also the actions played by the other agent(s)? Do they perhaps magically see the other agent(s)' mixed strategy in the stage game? And so on.

In general, games may be known or not, play may be observable or not, and so on. We will focus on known, fully observable games, where the other agent's strategy (or agents' strategies) is not known a priori (though in some cases there is a prior distribution over it). In our restricted setting there are two possible things to learn. First, the agent can learn the opponent's (or opponents') strategy (or strategies), so that the agent can then devise a best (or at least a good) response. Alternatively, the agent can learn a strategy of his own that does well against the opponents, without explicitly learning the opponent's strategy. The first is sometimes called *model-based learning*, and the second *model-free learning*.

In broader settings there is more to learn. In particular, with unknown games, one can learn the game itself. Some will argue that the restricted setting is not a true learning setting, but (a) much of the current work on MAL, particularly in game theory, takes place in this setting, and (b) the foundational issues we wish to tackle surface already here. In particular, our comments are intended to also apply to the work in the AI literature on games with unknown payoffs, work which builds on the success of learning in unknown MDPs. We will have more to say about the nature of "learning" in the setting of stochastic games in the following sections.

Problem Definition

When one examines the MAL literature one can identify several distinct agendas at play, which are often left implicit and conflated. A prerequisite for success in the field is to be very explicit about the problem being addressed. Here we list five distinct coherent goals of MAL research. They each have a clear motivation and a success criterion. They can be caricatured as follows:

1. Computational
2. Descriptive
3. Normative
4. Prescriptive, cooperative
5. Prescriptive, non-cooperative

The first agenda is computational in nature. It views learning algorithms as an iterative way to compute properties of the game, such as solution concepts. As an example, fictitious play was originally proposed as a way of computing a sample Nash equilibrium for zero-sum games, and replicator dynamics has been proposed

for computing a sample Nash equilibrium in symmetric games. These tend not to be the most efficient computation methods, but they do sometimes constitute quick-and-dirty methods that can easily be understood and implemented.

The second agenda is descriptive – it asks how natural agents learn in the context of other learners. The goal here is to investigate formal models of learning that agree with people's behavior (typically, in laboratory experiments), or possibly with the behaviors of other agents (e.g., animals or organizations). This problem is clearly an important one, and when taken seriously calls for strong justification of the learning dynamics being studied. One approach is to apply the experimental methodology of the social sciences.

The centrality of equilibria in game theory underlies the third agenda we identify in MAL, which for lack of a better term we called normative, and which focuses on determining which sets of learning rules are in equilibrium with each other. More precisely, we ask which repeated-game strategies are in equilibrium; it just so happens that in repeated games, most strategies embody a learning rule of some sort. For example, we can ask whether fictitious play and Q-learning, appropriately initialized, are in equilibrium with each other in a repeated Prisoner's Dilemma game.

The last two agendas are prescriptive; they ask how agents *should* learn. The first of these involves distributed control in dynamic systems. There is sometimes a need or desire to decentralize the control of a system operating in a dynamic environment, and in this case the local controllers must adapt to each other's choices. This direction, which is most naturally modeled as a repeated or stochastic common-payoff (or "team") game. Proposed approaches can be evaluated based on the value achieved by the joint policy and the resources required, whether in terms of computation, communication, or time required to learn the policy. In this case there is rarely a role for equilibrium analysis; the agents have no freedom to deviate from the prescribed algorithm.

In our final agenda, termed "prescriptive, noncooperative," we ask how an agent should act to obtain high reward in the repeated (and more generally, stochastic) game. It thus retains the design stance of AI, asking how to design an optimal (or at least effective) agent for a given environment. It just so happens that this environment is characterized by the types of agents inhabiting it, agents who may do some learning of their own. The objective of this agenda is to identify effective strategies for environments of interest. An effective strategy is one that achieves a high reward in its environment, where one of the main characteristics of this environment is the selected class of possible opponents. This class of opponents should itself be motivated as being reasonable and containing opponents of interest. Convergence to an equilibrium is not a goal in and of itself.

Recommended Reading

Requisite background in game theory can be obtained from the many introductory texts, and most compactly from Leyton-Brown and Shoham (2008). Game theoretic work on multi-agent learning is covered in Fudenberg and Levine (1998) and Young (2004). An expanded discussion of the problems addressed under the header of MAL can be found in Shoham et al. (2007), and the responses to it in Vohra and Wellman (2007). Discussion of MAL algorithms, both traditional and more novel ones, can be found in the above references, as well as in Greenwald and Littman (2007).

Fudenberg, D., & Levine, D. (1998). *The theory of learning in games*. Cambridge: MIT Press.

Greenwald, A., & Littman, M. L. (Eds.). (2007). Special issue on learning and computational game theory. *Machine Learning 67*(1–2).

Leyton-Brown, K., & Shoham, Y. (2008). *Essentials of game theory*. San Rafael, CA: Morgan and Claypool.

Shoham, Y., Powers, W. R., & Grenager, T. (2007). If multiagent learning is the answer, what is the question? *Artificial Intelligence, 171*(1), 365–377. Special issue on foundations of multiagent learning.

Vohra, R., & Wellman, M. P. (Eds.). (2007). Special issue on foundations of multiagent learning. *Artificial Intelligence, 171*(1).

Young, H. P. (2004). *Strategic learning and its limits*. Oxford: Oxford University Press.

Multi-Agent Learning II: Algorithms

Yoav Shoham, Rob Powers
Stanford University, Stanford, CA, USA

Definition

Multi-agent learning (MAL) refers to settings in which multiple agents learn simultaneously. Usually defined in a game theoretic setting, specifically in repeated games or stochastic games, the key feature that distinguishes

MAL from single-agent learning is that in the former the learning of one agent impacts the learning of others. As a result, neither the problem definition for multi-agent learning nor the algorithms offered follow in a straightforward way from the single-agent case. In this second of two entries on the subject we focus on algorithms.

Some MAL Techniques

We will discuss three classes of techniques – one representative of work in game theory, one more typical of work in artificial intelligence (AI), and one that seems to have drawn equal attention from both communities.

Model-Based Approaches

The first approach to learning we discuss, which is common in the game theory literature, is the model-based one. It adopts the following general scheme:

1. Start with some model of the opponent's strategy.
2. Compute and play the best response.
3. Observe the opponent's play and update your model of his/her strategy.
4. Go to step 2.

Among the earliest, and probably the best-known, instance of this scheme is *fictitious play*. The model is simply a count of the plays by the opponent in the past. The opponent is assumed to be playing a stationary strategy, and the observed frequencies are taken to represent the opponent's mixed strategy. Thus after five repetitions of the Rochambeau game (R) in which the opponent played (R, S, P, R, P), the current model of his/her mixed strategy is $R = 0.4, P = 0.4, S = 0.2$.

There exist many variants of the general scheme, for example, those in which one does not play the exact best response in step 2. This is typically accomplished by assigning a probability of playing each pure strategy, assigning the best response the highest probability, but allowing some chance of playing any of the strategies. A number of proposals have been made of different ways to assign these probabilities such as *smooth fictitious play* and *exponential fictitious play*.

A more sophisticated version of the same scheme is seen in *rational learning*. The model is a distribution over the repeated-game strategies. One starts with some prior distribution; for example, in a repeated Rochambeau game, the prior could state that with probability 0.5 the opponent repeatedly plays the equilibrium strategy of the stage game, and, for all $k > 1$, with probability 2^{-k} she plays R k times and then reverts to the repeated equilibrium strategy. After each play, the model is updated to be the posterior obtained by Bayesian conditioning of the previous model. For instance, in our example, after the first non-R play of the opponent, the posterior places probability 1 on the repeated equilibrium play.

Model-Free Approaches

An entirely different approach that has been commonly pursued in the AI literature is the model-free one, which avoids building an explicit model of the opponent's strategy. Instead, over time one learns how well one's own various possible actions fare. This work takes place under the general heading of *reinforcement learning* (we note that the term is used somewhat differently in the game theory literature), and most approaches have their roots in the Bellman equations. We start our discussion with the familiar single-agent *Q-learning* algorithm for computing an optimal policy in an unknown Markov Decision Problem (MDP).

$$Q(s,a) \leftarrow (1-\alpha_t)Q(s,a) + \alpha_t[R(s,a) + \gamma V(s')]$$

$$V(s) \leftarrow \max_{a \in A} Q(s,a).$$

As is well known, with certain assumptions about the way in which actions are selected at each state over time and constraints on the learning rate schedule, α_t, Q-learning can be shown to converge to the optimal value function V^*.

The Q-learning algorithm can be extended to the multi-agent stochastic game setting by having each agent simply ignore the other agents and pretend that the environment is passive:

$$Q_i(s,a_i) \leftarrow (1-\alpha_t)Q_i(s,a_i) + \alpha_t[R_i(s,\vec{a}) + \gamma V_i(s')]$$

$$V_i(s) \leftarrow \max_{a_i \in A_i} Q_i(s,a_i).$$

Several authors have tested variations of the basic Q-learning algorithm for MAL. However, this approach ignores the multi-agent nature of the setting entirely. The Q-values are updated without regard for the actions selected by the other agents. While this can be justified when the opponents' distributions of actions are

stationary, it can fail when an opponent may adapt its choice of actions based on the past history of the game.

A first step in addressing this problem is to define the Q-values as a function of all the agents' actions:

$$Q_i(s,\vec{a}) \leftarrow (1-\alpha)Q_i(s,\vec{a}) + \alpha[R_i(s,\vec{a}) + \gamma V_i(s')].$$

We are, however, left with the question of how to update V, given the more complex nature of the Q-values.

For (by definition, two-player) zero-sum Stochastic Games (SGs), the *minimax-Q* learning algorithm updates V with the minimax of the Q-values:

$$V_1(s) \leftarrow \max_{P_1 \in \Pi(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} P_1(a_1) Q_1(s,(a_1,a_2)).$$

Later work proposed other update rules for the Q and V functions focusing on the special case of common-payoff (or "team") games. A stage game is common-payoff if at each outcome all agents receive the same payoff. The payoff is, in general, different in different outcomes, and thus the agents' problem is that of coordination; indeed, these are also called *games of pure coordination*.

The work on zero-sum and common-payoff games continues to be refined and extended; much of this work has concentrated on provably optimal tradeoffs between exploration and exploitation in unknown, zero-sum games. Other work attempted to extend the "Bellman heritage" to general-sum games (as opposed to zero-sum or common-payoff games), but the results here have been less conclusive.

Regret Minimization Approaches

Our third and final example of prior work in MAL is no-regret learning. It is an interesting example for two reasons. First, it has some unique properties that distinguish it from the work above. Second, both the AI and game theory communities appear to have converged on it independently. The basic idea goes back to early work on how to evaluate the success of learning rules in the mid-1950s, and has since been extended and rediscovered numerous times over the years under the names of universal consistency, no-regret learning, and the Bayes' envelope. The following algorithm is a representative of this body of work. We start by defining the *regret*, $r_i^t(a_j, s_i)$ of agent i for playing the sequence of actions s_i instead of playing action a_j, given that the opponents played the sequence s_{-i}.

$$r_i^t(a_j, s_i | s_{-i}) = \sum_{k=1}^{t} R\left(a_j, s_{-i}^k\right) - R\left(s_i^k, s_{-i}^k\right).$$

The agent then selects each of its actions with probability proportional to $\max\left(r_i^t(a_j, s_i), 0\right)$ at each time step $t+1$.

Some Typical Results

One sees at least three kinds of results in the literature regarding the learning algorithms presented above, and others similar to them. These are:

1. Convergence of the strategy profile to an (e.g., Nash) equilibrium of the stage game in self-play (i.e., when all agents adopt the learning procedure under consideration).
2. Successful learning of an opponent's strategy (or opponents' strategies).
3. Obtaining payoffs that exceed a specified threshold.

Each of these types comes in many flavors; here are some examples. The first type is perhaps the most common in the literature, in both game theory and AI. For example, while fictitious play does not in general converge to a Nash equilibrium of the stage game, the distribution of its play can be shown to converge to an equilibrium in zero-sum games, 2×2 games with generic payoffs, or games that can be solved by iterated elimination of strictly dominated strategies. Similarly in AI, minimax-Q learning is proven to converge in the limit to the correct Q-values for any zero-sum game, guaranteeing convergence to a Nash equilibrium in self-play. This result makes the standard assumptions of infinite exploration and the conditions on learning rates used in proofs of convergence for single-agent Q-learning.

Rational learning exemplifies results of the second type. The convergence shown is to correct beliefs about the opponent's repeated game strategy; thus it follows that, since each agent adopts a best response to their beliefs about the other agent, in the limit the agents will converge to a Nash equilibrium of the repeated game. This is an impressive result, but it is limited by two factors: the convergence depends on a very strong

assumption of absolute continuity; and the beliefs converged to are correct only with respect to the aspects of history that are observable given the strategies of the agents. This is an involved topic, and the reader is referred to the literature for more details.

The literature on no-regret learning provides an example of the third type of result, and has perhaps been the most explicit about criteria for evaluating learning rules. For example, one pair of criteria that have been suggested are as follows. The first criterion is that the learning rule should be "safe," which is defined as the requirement that the learning rule must guarantee at least the minimax payoff of the game. (The minimax payoff is the maximum expected value a player can guarantee against any possible opponent.) The second criterion is that the rule should be "consistent." In order to be "consistent," the learning rule must guarantee that it does at least as well as the best response to the empirical distribution of play when playing against an opponent whose play is governed by independent draws from a fixed distribution. "Universal consistency" is then defined as the requirement that a learning rule does at least as well as the best response to the empirical distribution regardless of the actual strategy the opponent is employing (this implies both safety and consistency). The requirement of "universal consistency" is in fact equivalent to requiring that an algorithm exhibits *no-regret*, generally defined as follows, against all opponents.

$$\forall \epsilon > 0, \left(lim_{t \to \inf} \left[\frac{1}{t} \max_{a_j \in A_i} r_i^t(a_j, s_i | s_{-i}) \right] < \epsilon \right)$$

In both game theory and artificial intelligence, a large number of algorithms have been shown to satisfy universal consistency or no-regret requirements.

Recommended Reading

Requisite background in game theory can be obtained from the many introductory texts, and most compactly from Leyton-Brown and Shoham (2008). Game theoretic work on multiagent learning is covered in Fudenberg and Levine (1998) and Young (2004). An expanded discussion of the problems addressed under the header of MAL can be found in Shoham, Powers, and Grenager (2007), and the responses to it in Vohra and Wellman (2007). Discussion of MAL algorithms, both traditional and more novel ones, can be found in the above references, as well as in Greenwald and Littman (2007).

Fudenberg, D., & Levine, D. (1998). *The theory of learning in games*. Cambridge: MIT Press.

Greenwald, A., & Littman, M. L. (Eds.). (2007). Special issue on learning and computational game theory. *Machine Learning, 67*(1–2).

Leyton-Brown, K., & Shoham, Y. (2008). *Essentials of game theory*. San Rafael, CA: Morgan and Claypool.

Shoham, Y., Powers, W. R., & Grenager, T. (2007). If multiagent learning is the answer, what is the question? *Artificial Intelligence, 171*(1), 365–377. Special issue on foundations of multiagent learning.

Vohra, R., & Wellman, M. P. (Eds.). (2007). Special issue on foundations of multiagent learning. *Artificial Intelligence, 171*(1).

Young, H. P. (2004). *Strategic learning and its limits*. Oxford: Oxford University Press.

Multi-Armed Bandit

►*k*-Armed Bandit

Multi-Armed Bandit Problem

►*k*-Armed Bandit

MultiBoosting

Geoffrey I. Webb
Monash University,
Victoria, Australia

Definition

MultiBoosting (Webb, 2000) is an approach to ►multi-strategy ensemble learning that combines features of ►AdaBoost and ►Bagging. The insight underlying MultiBoosting is that the primary effect of AdaBoost is ►bias reduction, while the primary effect of bagging is ►variance reduction. By combining the two techniques, it is possible to obtain both bias and variance reduction, the cumulative effect often being a greater reduction in error than can be obtained with the equivalent amount of computation by either AdaBoost or Bagging alone. Viewed from another perspective, as the size of an ensemble formed by either AdaBoost or Bagging is increased, each successive addition to the ensemble has decreasing effect. Thus, if the benefit of the first few applications of AdaBoost can be combined with

the benefit of the first few applications of Bagging, the combined benefit may be greater than simply increasing the number of applications of one or the other.

Algorithm

MultiBoosting operates by dividing the ensemble of classifiers that is to be created into a number of subcommittees. Each of these subcommittees is formed by Wagging (Baner & Kohavi, 1999), a variant of Bagging that utilizes weighted instances and, hence, is more readily integrated with AdaBoost. The ensemble is formed by applying AdaBoost to these subcommittees. The resulting algorithm is presented in Table 1. The learned ensemble classifier is C, and the tth member of the ensemble is C_t. Each S_t is a vector of n weighted training objects whose weights always sum to n. The weights change from turn to turn (the turns indicated by the subscript t). The base training algorithm *BaseLearn* should more heavily penalize errors on training instances with higher weights. ϵ_t is the weighted error

MultiBoosting. Table 1 MultiBoost Algorithm

MultiBoost

input:

- S_0, a sequence of m labeled examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ with labels $y_i \in Y$.
- base learning algorithm *BaseLearn*.
- integer T specifying the number of iterations.
- vector of integers I_i specifying the iteration at which each subcommittee $i \geq 1$ should terminate.

1. $S_1 = S_0$ with instance weights assigned to be 1.
2. set $k = 1$.
3. For $t = 1$ to T
4. If $I_k = t$ then
5. reweight S_t.
6. increment k.
7. $C_t = BaseLearn(S')$.
8. $\epsilon_t = \dfrac{\sum_{x_j \in S_t : C_t(x_j) \neq y_j} weight(x_j)}{m}$.
9. if $\epsilon_t > 0.5$ then
10. reweight S_t.
11. increment k.
12. go to 7.
13. otherwise if $\epsilon_t = 0$ then
14. set β_t to 10^{-10}.
15. reweight S_t.
16. increment k.
17. otherwise,
18. $\beta_t = \dfrac{\epsilon_t}{(1 - \epsilon_t)}$.
19. $S_{t+1} = S_t$.
20. For each $x_j \in S_{t+1}$,
21. divide $weight(x_j)$ by $2\epsilon_t$ if $C_t(x_j) \neq y_j$ and $2(1 - \epsilon_t)$ otherwise.
22. if $weight(x_j) < 10^{-8}$, set $weight(x_j)$ to 10^{-8}.

Output the final classifier: $C^*(x) = \underset{y \in Y}{argmax} \sum_{t : C_t(x) = y} log \dfrac{1}{\beta_t}$.

of C_t on S_i. β_t is a weight assigned to the tth classifier, C_t. The operation rewieght S_t sets the weights of the objects in S_t to random values drawn from the continuous Poisson distribution and then standardizes them to sum to n. The code set with a grey background is the code added to AdaBoost in order to create MultiBoost.

Cross References

►AdaBoost
►Bagging
►Ensemble Learning
►Multistrategy Ensemble Learning

Recommended Reading

Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning, 36*(1), 105–139.

Webb, G. I. (2000). MultiBoosting: A technique for combining boosting and wagging. *Machine Learning, 40*(2), 159–196.

Multi-Criteria Optimization

►Multi-Objective Optimization

Multi-Instance Learning

Soumya Ray,[1] Stephen Scott,[2] Hendrik Blockeel[3]
[1]Case Western Reserve University, Cleveland, OH, USA
[2]University of Nebraska, Lincoln, NE, USA
[3]K. U. Leuven, Heverlee, Belgium

Synonyms

Multiple-instance learning

Definition

Multiple-Instance (MI) learning is an extension of the standard supervised learning setting. In standard supervised learning, the input consists of a set of labeled instances each described by an attribute vector. The learner then induces a concept that relates the label of an instance to its attributes. In MI learning, the input consists of labeled examples (called "bags") consisting of *multisets* of instances, each described by an attribute vector, and there are constraints that relate the label of each bag to the unknown labels of each instance. The MI learner then induces a concept that relates the label of a bag to the attributes describing the instances in it. This setting contains supervised learning as a special case: if each bag contains exactly one instance, it reduces to a standard supervised learning problem.

Motivation and Background

The MI setting was introduced by Dietterich, Lathrop, and Lozano-Perez (1997) in the context of drug activity prediction. Drugs are typically molecules that fulfill some desired function by binding to a target. If we wish to learn the characteristics responsible for binding, a possible representation of the problem is to represent each molecule as a set of low energy shapes or *conformations*, and describe each conformation using a set of attributes. Each such bag of conformations is given a label corresponding to whether the molecule is active or inactive. To learn a classification model, an algorithm assumes that every instance in a bag labeled negative is actually negative, whereas at least one instance in a bag labeled positive is actually positive with respect to the underlying concept.

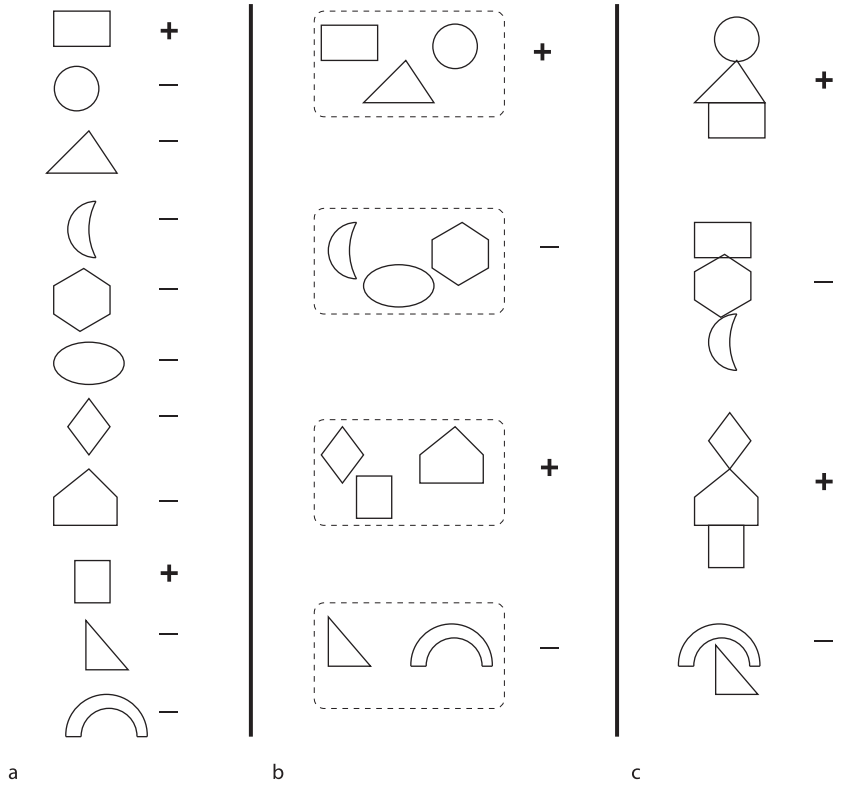
From a theoretical viewpoint, MI learning occupies an intermediate position between standard propositional supervised learning and first-order relational learning. Supervised learning is a special case of MI learning, while MI learning is a special case of first-order learning. It has been argued that the MI setting is a key transition between standard supervised and relational learning DeRaedt (1998). At the same time, theoretical results exist that show that, under certain assumptions, certain concept classes that are probably approximately correct (PAC)-learnable (see PAC Learning) in a supervised setting remain PAC-learnable in an MI setting. Thus, the MI setting is able to leverage some of the rich representational power of relational learners while not sacrificing the efficiency of propositional learners. Figure 1 illustrates the relationships between standard supervised learning, MI learning, and relational learning.

Since its introduction, a wide variety of tasks have been formulated as MI learning problems. Many

new algorithms have been developed, and well-known supervised learning algorithms extended, to learn MI concepts. A great deal of work has also been done to understand what kinds of concepts can and cannot be learned efficiently in this setting. In the following sections, we discuss the theory, methods, and applications of MI learning in more detail.

Structure of the Problem

The general MI classification task in shown in Fig. 2. The MI regression task is defined analogously by substituting a real-valued response for the classification label. In this case, the constraint used by the learning algorithm is that the response of any bag is equal to the



Multi-Instance Learning. Figure 1. The relationship between supervised, multiple-instance (MI), and relational learning. (a) In supervised learning, each example (geometric figure) is labeled. A possible concept that explains the example labels shown is "the figure is a rectangle." (b) In MI learning, bags of examples are labeled. A possible concept that explains the bag labels shown is "the bag contains at least one figure that is a rectangle." (c) In relational learning, objects of arbitrary structure are labeled. A possible concept that explains the object labels shown is "the object is a stack of three figures and the bottom figure is a rectangle"

Given: A set of bags $\{B_1, ...B_n\}$ each with label $\ell_i \in \{0, 1\}$. Each B_i is a multiset of n_i instances, $B_i = \{B_{i1}, \ldots, B_{in_i}\}$.

Constraints: There exists a concept c such that:

- For every B_i with $\ell_i = 1$, $c(B_{ij}) = 1$ for at least one j, and
- For every B_i with $\ell_i = 0$, $c(B_{ij}) = 0$ for all j.

Do: Learn a concept that maps a bag B_i to its label ℓ_i.

Multi-Instance Learning. Figure 2. Statement of the multiple-instance classification problem

response of at least one of the instances in it, for example, it could be equal to the largest response over all the instances.

Notice the following problem characteristics:

- The number of instances in each bag can vary independently of other bags. This implies in particular that an MI algorithm must be able to handle bags with as few as one instance (this is a supervised learning setting) to bags with large numbers of instances.
- The number of instances in any positive bag that are "truly positive" could be many more than one – in fact, the definition does not rule out the case where *all* instances in a positive bag are "truly positive."
- The problem definition does not specify how the instances in any bag are related to each other.

Theory and Methods

In this section we discuss some of the key algorithms and theoretical results in MI learning. We first discuss the methods and results for MI classification. Then we discuss the work on MI regression.

Multiple-Instance Classification

Axis-Parallel Rectangles (APRs) are a concept class that early work in MI classification focused on. These generative concepts specify upper and lower bounds for all numeric attributes describing each instance. An APR is said to "cover" an instance if the instance lies within it. An APR covers a bag if it covers at least one instance within it. The learning algorithm tries to find an APR such that it covers all positive bags and does not cover any negative bags.

An algorithm called "iterated-discrimination" was proposed by Dietterich et al. (1997) to learn APRs from MI data. This algorithm has two phases. In the first phase, it iteratively chooses a set of "relevant" attributes and grows an APR using this set. This phase results in the construction of a very "tight" APR that covers just positive bags. In the second phase, the algorithm expands this APR so that with high probability a new positive instance will fall within the APR. The key steps of the algorithm are outlined below. Note that initially, all attributes are considered to be "relevant."

The algorithm starts by choosing a random instance in a positive bag. Let us call this instance I_1. The smallest APR covering this instance is a point. The algorithm then expands this APR by finding the smallest APR that covers any instance from a yet uncovered positive bag; call the newly covered instance I_2. This process is continued, identifying new instances $I_3, \ldots, I_k$, until all positive bags are covered. At each step, the APR is "backfitted" in a way that is reminiscent of the later Expectation-Maximization (EM) approaches: each earlier choice is revisited, and I_j is replaced with an instance from the same bag that minimizes the current APR (which may or may not be the same as the one that minimized it at step j).

This process yields an APR that imposes maximally tight bounds on all attributes and covers all positive bags. Based on this APR, a new set of "relevant" attributes is selected as follows. An attribute's relevance is determined by how strongly it discriminates against negative instances, i.e., given the current APR bounds, how many negative instances the attribute excludes. Features are then chosen iteratively and greedily according to how relevant they are until all negative instances have been excluded. This yields a subset of (presumably relevant) attributes. The APR growth procedure in the previous paragraph is then repeated, with the size of an APR redefined as its size along relevant attributes only. The APR growth and attribute selection phases are repeated until the process converges.

The APR thus constructed may still be too tight, as it fits narrowly around the positive bags in the dataset. In the second phase of the algorithm, the APR bounds are further expanded using a kernel density estimate approach. Here, a probability distribution is constructed for each relevant attribute using Gaussian distributions centered at each instance in a positive bag. Then, the bounds on that attribute are adjusted so that with high probability, any positive instance will lie within the expanded APR.

Theoretical analyses of APR concepts have been performed along with the empirical approach, using Valiant's "probably approximately correct" (PAC) learning model (Valiant, 1984). In early work (Long & Tan, 1998), it was shown that if each instance was drawn according to a fixed, unknown product distribution over the rational numbers, independently from every other instance, then an algorithm could PAC-learn APRs. Later, this result was improved in two ways (Auer, Long, & Srinivasan, 1998). First, the restriction that the

individual instances in each bag come from a product distribution was removed. Instead, each instance is generated by an arbitrary probability distribution (though each instance in a bag is still generated independently and identically distributed (iid) according to that one distribution). Second, the time and sample complexities for PAC-learning APRs were improved. Specifically, the algorithm described in this work PAC-learns APRs in

$$O\left(\frac{d^3n^2}{\epsilon^2}\log\frac{nd\log(1/\delta)}{\epsilon}\log\frac{d}{\delta}\right)$$

using

$$O\left(\frac{d^2n^2}{\epsilon^2}\log\frac{d}{\delta}\right)$$

time-labeled training bags. Here, d is the dimension of each instance, n is the (largest) number of instances per training bag, and ϵ and δ are parameters to the algorithm. A variant of this algorithm was empirically evaluated and found to be successful (Auer, 1997).

Diverse Density (Maron, 1998; Maron & Lozano-Pérez, 1998) is a probabilistic generative framework for MI classification. The idea behind this framework is that, given a set of positive and negative bags, we wish to learn a concept that is "close" to at least one instance from each positive bag, while remaining "far" from every instance in every negative bag. Thus, the concept must describe a region of instance space that is "dense" in instances from positive bags, and is also "diverse" in that it describes every positive bag. More formally, let

$$DD(t) = \frac{1}{Z}\left(\prod_i \Pr(t|B_i^+)\prod_i \Pr(t|B_i^-)\right),$$

where t is a candidate concept, B_i^+ represents the ith positive bag, and B_i^- represents the ith negative bag. We seek a concept that maximizes $DD(t)$. The concept generates the instances of a bag, rather than the bag itself. To score a concept with respect to a bag, we combine t's probabilities for instances using a function based on noisy-OR Pearl (1998):

$$\Pr(t|B_i^+) \propto (1 - \prod_j (1 - \Pr(B_{ij}^+ \in t))) \quad (1)$$

$$\Pr(t|B_i^-) \propto \prod_j (1 - \Pr(B_{ij}^- \in t)) \quad (2)$$

Here, the instances B_{ij}^+ and B_{ij}^- belonging to t are the "causes" of the "event" that "t is the target." The concept class investigated by Maron (1998) is the class of generative Gaussian models, which are parameterized by the mean μ and a "scale" $s = \frac{1}{2\sigma^2}$:

$$\Pr(B_{ij} \in t) \propto e^{-\sum_k (s_k (B_{ijk} - \mu_k)^2)},$$

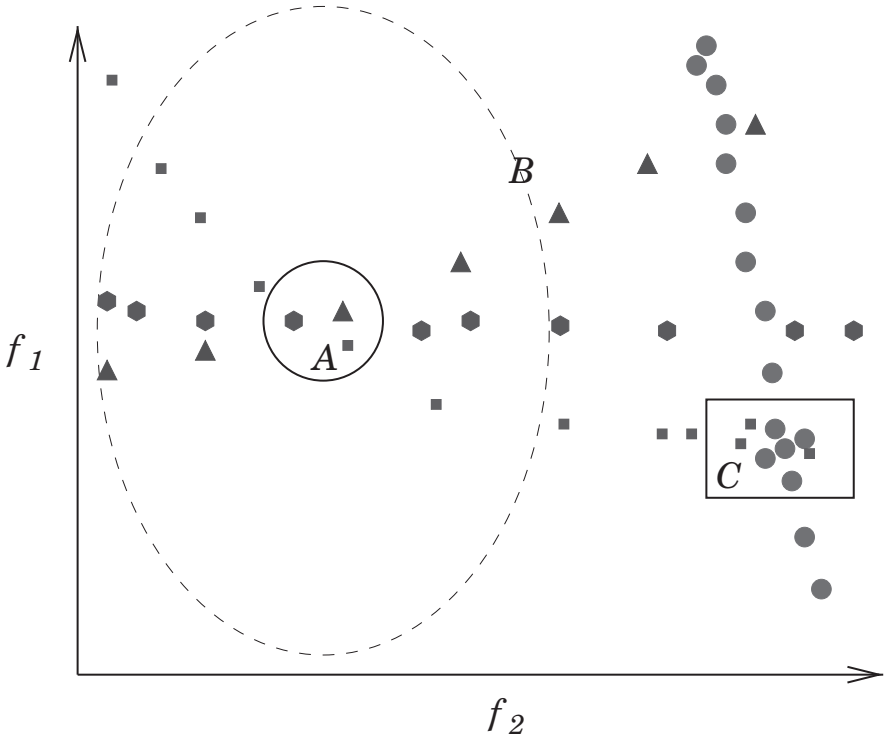
where k ranges over attributes. Figure 3 illustrates a concept that Diverse Density might learn when applied to an MI dataset.

Diverse Density with k disjuncts is a variant of Diverse Density that has also been investigated (Maron, 1998). This is a class of disjunctive Gaussian concepts, where the probability of an instance belonging to a concept is given by the maximum probability of belonging to any of the disjuncts.

EM-DD (Zhang & Goldman, 2001) is an example of a class of algorithms that try to identify the "cause" of a bag's label using EM. These algorithms sometimes assume that there is a single instance in each bag that is responsible for the bag's label (though variants using "soft EM" are possible). The key idea behind this approach is as follows: from each positive bag, we take a random instance and assume that this instance is the relevant one. We learn a hypothesis from these relevant instances and all negative bags. Next, for each positive bag, we replace the current relevant instance by the instance most consistent with the learned hypothesis (which will initially not be the chosen instance in general). We then relearn the hypothesis with these new instances. This process is continued until the set of chosen instances does not change (or alternatively, the objective function of the classifier reaches a fixed point). This procedure has the advantage of being computationally efficient, since the learning algorithm only uses one instance from each positive bag. This approach has also been used in MI regression described later.

"Upgraded" supervised learning algorithms can be used in a MI setting by suitably modifying their objective functions. Below, we summarize some of the algorithms that have been derived in this way.

1. ▶*Decision Tree induction* algorithms have been adapted to the MI setting (Blockeel, Page, & Srinivasan, 2005). The standard algorithm measures



Multi-Instance Learning. Figure 3. An illustration of the concept that Diverse Density searches for on a simple MI dataset with three positive bags and one negative bag, where each instance (represented by the geometric figures) is described by two attributes, f_1 and f_2. Each type of figure represents one bag, i.e., all triangles belong to one bag, all circles belong to a second bag, and so forth. The bag containing the red circles is negative, while the other bags are positive. Region *C* is a region of high density, because several instances belong to that region. Region *A* is a region of high "Diverse Density," because several instances *from different positive bags* belong to that region, and no instances from negative bags are nearby. Region *B* shows a concept that might be learned if the learning algorithm assumed that all instances in every positive bag are positive. Figure adapted from Maron (1998)

the quality of a split on an attribute by considering the class label distribution in the child nodes produced. In the MI case, this distribution is uncertain, because the true instance labels in positive bags are unknown. However, some rules have been identified that lead to empirically good MI trees: (1) use an asymmetric heuristic that favors early creation of pure positive (rather than negative) leaves, (2) once a positive leaf has been created, remove all other instances of the bags covered by this leaf; (3) abandon the depth-first or breadth-first order in which nodes are usually split, adopting a best-first strategy instead (indeed, because of (2), the result of tree learning is now sensitive to the order in which the nodes are split).

2. ►*Artificial Neural Networks* have been adapted to the MI setting by representing the bag classifier as a network that combines several copies of a smaller network, which represents the instance classifier, with a smooth approximation of the *max* combining function (Ramon & DeRaedt, 2000). Weight update rules for a backpropagation algorithm working on this network have been derived. Later work on MI neural networks has been performed independently by others (Zhou & Zhang, 2002).
3. ►*Logistic Regression* has been adapted to the MI setting by using it as an instance-based classifier and combining the instance-level probabilities using functions like softmax (Ray & Craven, 2005) and

arithmetic and geometric averages (Xu & Frank, 2004).

4. The ►*k-Nearest Neighbor* algorithm has been adapted to the MI setting by using set-based distance metrics, such as variants based on the Hausdorff distance. However, this alone does not solve the problem – it is possible for a positive bag to be mistakenly classified negative if it contains a "true negative" instance that happens to be much closer to negative instances in other negative bags. To solve this, a "Citation-kNN" (Wang & Zucker, 2000) approach has been proposed that also considers, for each bag B, the labels of those bags for which B is a nearest neighbor.
5. ►*Support Vector Machines* have been adapted to the MI setting in several ways. In one method, the constraints in the quadratic program for SVMs is modified to account for the fact that certain instance labels are unknown but have constraints relating them (Andrews, Tsochantaridis, & Hofmann, 2003). In another method, new kernels are designed for MI data by modifying standard supervised SVM kernels (Gartner, Flach, Kowalczyk, & Smola, 2002) or designing new kernels (Tao, Scott, & Vinodchandran, 2004). The modification allows these MI kernels to distinguish between positive and negative bags if the supervised kernel could distinguish between ("true") positive and negative instances.
6. ►*Rule learning algorithms* have been adapted to the MI setting in two ways. One method has investigated upgrading a supervised rule-learner, the RIPPER system (Cohen, 1995), to the MI setting by modifying its objective function to account for bags and addressing several issues that resulted. Another method has investigated using general purpose relational algorithms, such as FOIL (Quinlan, 1990) and TILDE (Blockeel & De Raedt, 1998), and providing them with an appropriate ►inductive bias so that they learn the MI concepts. Further, it has been observed that techniques from MI learning can also be used inside relational learning algorithms (Alphonse & Matwin, 2002).

A large-scale empirical analysis of several such propositional supervised learning algorithms and their MI counterparts has been performed (Ray & Craven, 2005). This analysis concludes that (1) no single MI algorithm works well across all problems. Thus, different inductive biases are suited to different problems, (2) some MI algorithms consistently perform better than their supervised counterparts but others do not (hence for these biases there seems room for improvement), and (3) assigning a larger weight to false positives than to false negatives is a simple but effective method to adapt supervised learning algorithms to the MI setting. It was also observed that the advantages of MI learners may be more pronounced if they would be evaluated on the task of labeling individual instances rather than bags.

Along with "upgrading" supervised learning algorithms, a *theoretical analysis of supervised learners* learning with MI data has been carried out (Blum & Kalai, 1998). In particular, the MI problem has been related to the problem of learning in the presence of classification noise (i.e., each training example's label is flipped with some probability $< 1/2$). This implies that any concept class that is PAC-learnable in the presence of such noise is also learnable in the MI learning model when each instance of a bag is drawn iid. Since many concept classes are learnable under this noise assumption (using e.g., *statistical queries* Kearns, 1998), Blum and Kalai's result implies PAC learnability of many concept classes. Further, they improved on previous learnability results (Auer et al., 1998) by reducing the number of training bags required for PAC learning by about a factor of n with only an increase in time complexity of about $\log n/\epsilon$.

Besides these positive results, a *negative learnability result* describing when it is hard to learn concepts from MI data is also known (Auer et al., 1998). Specifically, if the instances of each bag are allowed collectively to be generated according to an arbitrary distribution, learning from MI examples is as hard as PAC-learning disjunctive normal form (DNF) formulas from single-instance examples, which is an open problem in learning theory that is believed to be hard. Further, it has been showed that if an efficient algorithm exists for the non-iid case that outputs as its hypothesis an axis-parallel rectangle, then NP = RP (Randomized Polynomial time, see e.g., Papadimitriou, 1994), which is very unlikely.

Learning from structured MI data has received some attention (McGovern & Jensen, 2003). In this work,

each instance is a graph, and a bag is a set of graphs (e.g., a bag could consist of certain subgraphs of a larger graph). To learn the concepts in this structured space, the authors use a modified form of the Diverse Density algorithm discussed above. As before, the concept being searched for is a point (which corresponds to a graph in this case). The main modification is the use of the size of the maximal common subgraph to estimate the probability of a concept – i.e., the probability of a concept given a bag is estimated as proportional to the size of the maximal common subgraph between the concept and any instance in the bag.

Multiple-Instance Regression

Regression problems in an MI setting have received less attention than the classification problem. Two key directions have been explored in this setting. One direction extends the well-known standard ►linear regression method to the MI setting. The other direction considers extending various MI classification methods to a regression setting.

In *MI Linear Regression* (Ray & Page, 2001) (referred to as multiple-instance regression in the cited work), it is assumed that the hypothesis underlying the data is a linear model with Gaussian noise on the value of the dependent variable (which is the response). Further, it is assumed that it is sufficient to model one instance from each bag, i.e., that there is some *primary* instance which is responsible for the real-valued label. Ideally, one would like to find a hyperplane that minimizes the squared error with respect to these primary instances. However, these instances are unknown during training. The authors conjecture that, given enough data, a good approximation to the ideal is given by the "best-fit" hyperplane, defined as the hyperplane that minimizes the training set squared error by fitting one instance from each bag such that the response of the fitted instance most closely matches the bag response. This conjecture will be true if the nonprimary instances are not a better fit to a hyperplane than the primary instances. However, exactly finding the "best-fit" hyperplane is intractable. It is shown that the decision problem "Is there a hyperplane which perfectly fits one instance from each bag?" is *NP*-complete for arbitrary numbers of bags, attributes, and at most three instances per bag. Thus, the authors propose an approximation algorithm which iterates between choosing instances and learning linear regression models that best fit them, similar to the EM-DD algorithm described earlier.

Another direction has explored *extending MI classification algorithms* to the regression setting. This approach (Dooly, Zhang, Goldman, & Amar, 2002) uses algorithms like Citation-kNN and Diverse Density to learn real-valued concepts. To predict a real value, the approach uses the average of the nearest neighbor responses or interprets the Gaussian "probability" as a real number for Diverse Density.

Recent work has analyzed the Diverse Density-based regression in the *online* model (Angluin, 1988; Littlestone, 1988) (see ►online learning). In the online model, learning proceeds in *trials*, where in each trial a single example is selected adversarially and given to the learner for classification. After the learner predicts a label, the true label is revealed and the learner incurs a *loss* based on whether its prediction was correct. The goal of the online learner is to minimize the loss over all trials. Online learning is harder than PAC learning in that there are some PAC-learnable concept classes that are not online learnable.

In the regression setting above (Dooly, Goldman, & Kwek, 2006), there is a point concept, and the label of each bag is a function of the distance between the concept and the point in the bag closest to the target. It is shown that similar to Auer et al.'s lower bound, learning in this setting using labeled bags alone is as hard as learning DNF. They then define an *MI membership query* (MI-MQ) in which an adversary defines a bag $B = \{p_1, \ldots, p_n\}$ and the learner is allowed to ask an oracle for the label of bag $B + \vec{v} = \{p_1 + \vec{v}, \ldots, p_n + \vec{v}\}$ for any d-dimensional vector $\vec{v}$. Their algorithm then uses this MI-MQ oracle to online learn a real-valued MI concept in time $O(dn^2)$.

Applications

In this section, we describe domains where MI learning problems have been formulated.

Drug activity was the motivating application for the MI representation (Dietterich et al., 1997). Drugs are typically molecules that fulfill some desired function by binding to a target. In this domain, we wish to predict how strongly a given molecule will bind to a target. Each molecule is a three-dimensional entity and

takes on multiple shapes or *conformations* in solution. We know that for every molecule showing activity, at least one of its low energy conformations possesses the right shape for interacting with the target. Similarly, if the molecule does not show drug-like activity, none of its conformations possess the right shape for interaction. Thus, each molecule is represented as a bag, where each instance is a low energy conformation of the molecule. A well-known example from this domain is the MUSK dataset. The positive class in this data consists of molecules that smell "musky." This dataset has two variants, MUSK1 and MUSK2, both with similar numbers of bags, with MUSK2 having many more instances per bag.

Content-Based Image Retrieval is another domain where the MI representation has been used (Maron & Lozano-Pérez, 1998; Zhang, Yu, Goldman, & Fritts, 2002). In this domain, the task is to find images that contain objects of interest, such as tigers, in a database of images. An image is represented by a bag. An instance in a bag corresponds to a segment in the image, obtained by some segmentation technique. The underlying assumption is that the object of interest is contained in (at least) one segment of the image. For example, if we are trying to find images of mountains in a database, it is reasonable to expect most images of mountains to have certain distinctive segments characteristic of mountains. An MI learning algorithm should be able to use the segmented images to learn a concept that represents the shape of a mountain and use the learned concept to collect images of mountains from the database.

The *identification of protein families* has been framed as an MI problem (Tao et al., 2004). The objective in that work is to classify given protein sequences according to whether they belong to the family of thioredoxin-fold proteins. The given proteins are first aligned with respect to a motif that is known to be conserved in the members of the family. Each aligned protein is represented by a bag. A bag is labeled positive if the protein belongs to the family, and negative otherwise. An instance in a bag corresponds to a position in a fixed length sequence around the conserved motif. Each position is described by a vector of attributes; each attribute describes the properties of the amino acid at that position, and is smoothed using the same properties from its neighbors.

Text Categorization is another domain that has used the MI representation (Andrews et al., 2003; Ray & Craven 2005). In this domain, the task is to classify a document as belonging to a certain category or not. Often, whether the document belongs to the specified category is the function of a few passages in the document. These passages are however not labeled with the category information. Thus, a document could be represented as a set of passages. We assume that each positive document (i.e., that belongs to the specified category) has at least one passage that contains words that indicate category membership. On the other hand, a negative document (that does not belong to the category) has no passage that contain words indicating category membership. This formulation has been used to classify whether MEDLINE documents should be annotated with specific MeSH terms (Andrews et al.) and to determine if specific documents should be annotated with terms from the Gene Ontology (Ray & Craven, 2005).

Time-series data from the hard drives have been used to define an MI problem (Murray, Hughes, & Kreutz-Delgado, 2005). The task here is to distinguish drives that fail from others. Each hard drive is a bag. Each instance in the bag is a fixed-size window over timepoints when the drive's state was measured using certain attributes. In the training set, each drive is labeled according to whether it failed during a window of observation. An interesting aspect to prediction in this setting is that it is done online, i.e., the algorithm learns a classifier for instances, which is applied to each instance as it becomes available in time. The authors learn a naïve Bayes model using an EM-based approach to solve this problem.

Discovering useful subgoals in reinforcement learning has been formulated as an MI problem (McGovern & Barto, 2001). Imagine that a robot has to get from one room to another by passing through a connecting door. If the robot knew of the existence of the door, it could decompose the problem into two simpler subproblems to be solved separately: getting from the initial location in the first room to the door, and then getting from the door to its destination. How could the robot discover such a "useful subgoal?" One approach formulates this as an MI problem. Each trajectory of the robot, where the robot starts at the source and then moves for some number of time steps, is considered to be a bag. An

instance in a bag is a state of the world, that records observations such as, "is the robot's current location a door?" Trajectories that reach the destination are positive, while those that do not are negative. Given this data, we can learn a classifier that predicts which states are more likely to be seen on successful trajectories than on unsuccessful ones. These states are taken to be useful subgoals. In the previous example, the MI algorithm could learn that the state "location is a door" is a useful subgoal, since it appears on all successful trajectories, but infrequently on unsuccessful ones.

Future Directions

MI learning remains an active research area. One direction that is being explored relaxes the "Constraints" in Fig. 2 in different ways (Tao et al., 2004; Weidmann, Frank, & Pfahringer 2003). For example, one could consider constraints where at least a certain number (or fraction) of instances have to be positive for a bag to be labeled positive. Similarly, it may be the case that a bag is labeled positive only if it does not contain a specific instance. Such relaxations are often studied as "generalized multiple-instance learning."

One such generalization of MI learning has been formally studied under the name "geometric patterns." In this setting, the target concept consists of a collection of APRs, and a bag is labeled positive if and only if (1) each of its points lies in a target APR, and (2) every target APR contains a point. Noise-tolerant PAC algorithms (Goldman & Scott, 1999) and online algorithms (Goldman, Kwek, & Scott, 2001) have been presented for such concept classes. These algorithms make no assumptions on the distribution used to generate the bags (e.g., instances might not be generated by an iid process). This does not violate Auer et al.'s lower bound since these algorithms do not scale with the dimension of the input space.

Another recent direction explores the connections between MI and semi-supervised learnings. Semi-supervised learning generally refers to learning from a setting where some instance labels are unknown. MI learning can be viewed as one example of this setting. Exploiting this connection between MI learning and other methods for semi-supervised learning, recent work (Rahmani & Goldman, 2006) proposes an approach where an MI problem is transformed into a semi-supervised learning problem. An advantage of the approach is that it automatically also takes into account unlabeled bags.

Cross References

- ▶Artificial Neural Network
- ▶Attribute
- ▶Classification
- ▶Data Set
- ▶Decision Trees
- ▶Expectation-Maximization
- ▶First-Order Rule
- ▶Gaussian Distribution
- ▶Inductive Logic Programming
- ▶Kernel Methods
- ▶Linear Regression
- ▶Nearest Neighbor
- ▶Noise
- ▶On-Line Learning
- ▶PAC Learning
- ▶Relational Learning
- ▶Supervised Learning

Recommended Reading

Alphonse, E., & Matwin, S. (2002). Feature subset selection and inductive logic programming. In Proceedings of the 19th International Conference on Machine Learning (pp. 11–18). Morgan Kaufmann, San Francisco, USA.

Andrews, S., Tsochantaridis, I., & Hofmann, T. (2003). Support vector machines for multiple-instance learning. In S. Becker, S. Thrun, & K. Obermayer, (Eds.), *Advances in neural information processing systems*. (Vol. 15, pp. 561–568). Cambridge, MA: MIT Press.

Angluin, D. (1988). Queries and concept learning. *Machine Learning, 2*(4), 319–342.

Auer, P. (1997). On learning from multi-instance examples: Empirical evaluation of a theoretical approach. In *Proceeding of 14th international conference on machine learning* (pp. 21–29). San Francisco: Morgan Kaufmann.

Auer, P., Long, P. M., & Srinivasan, A. (1998). Approximating hyper-rectangles: Learning and pseudorandom sets. *Journal of Computer and System Sciences, 57*(3), 376–388.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence, 101*(1–2), 285–297.

Blockeel, H., Page, D., & Srinivasan, A. (2005). Multi-instance tree learning. In *Proceedings of 22nd international conference on machine learning* (pp. 57–64). Bonn, Germany.

Blum, A., & Kalai, A. (1998). A note on learning from multiple-instance examples. *Machine Learning Journal, 30*(1), 23–29.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the 12th international conference on machine learning*. San Francisco: Morgan Kaufmann.

DeRaedt, L. (1998). Attribute-value learning versus inductive logic programming: The missing links. In *Proceedings of the eighth international conference on inductive logic programming* (pp. 1–8). New York: Springer.

Dietterich, T., Lathrop, R., & Lozano-Perez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence, 89*(1–2), 31–71.

Dooly, D. R., Goldman, S. A., & Kwek, S. S. (2006). Real-valued multiple-instance learning with queries. *Journal of Computer and System Sciences, 72*(1), 1–15.

Dooly, D. R., Zhang, Q., Goldman, S. A., & Amar, R. A. (2002). Multiple-instance learning of real-valued data. *Journal of Machine Learning Research, 3*, 651–678.

Gartner, T., Flach, P. A., Kowalczyk, A., & Smola, A. J. (2002). Multi-instance kernels. In C. Sammut, & A. Hoffmann, (Eds.), *Proceedings of the 19th international conference on machine learning* (pp. 179–186). San Francisco: Morgan Kaufmann.

Goldman, S. A., Kwek, S. K., & Scott, S. D. (2001). Agnostic learning of geometric patterns. *Journal of Computer and System Sciences, 6*(1), 123–151.

Goldman, S. A., & Scott, S. D. (1999). A theoretical and empirical study of a noise-tolerant algorithm to learn geometric patterns. *Machine Learning, 37*(1), 5–49.

Kearns, M. (1998). Efficient noise-tolerant learning from statistical queries. *Journal of the ACM, 45*(6), 983–1006.

Long, P. M., & Tan, L. (1998). PAC learning axis-aligned rectangles with respect to product distributions from multiple-instance examples. *Machine Learning, 30*(1), 7–21.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning, 2*(4), 285–318.

Maron, O. (1998). Learning from ambiguity. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA.

Maron, O., & Lozano-Pérez, T. (1998). A framework for multiple-instance learning. In M. I. Jordan, M. J. Kearns, & S. A. Solla, (Eds.), *Advances in neural information processing systems* (Vol. 10, pp. 570–576). Cambridge, MA: MIT Press.

McGovern, A., & Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the 18th international conference on machine learning* (pp. 361–368). San Francisco: Morgan Kaufmann.

McGovern, A., & Jensen, D. (2003). Identifying predictive structures in relational data using multiple instance learning. In *Proceedings of the 20th international conference on machine learning* (pp. 528–535). Menlo Park, USA: AAAI Press.

Murray, J. F., Hughes, G. F., & Kreutz-Delgado, K. (2005). Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research, 6*, 783–816.

Papadimitriou, C. (1994). *Computational complexity*. Boston, MA: Addison-Wesley.

Pearl, J. (1998). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Rahmani, R., & Goldman, S. A. (2006). MISSL: Multiple-instance semi-supervised learning. In *Proceedings of the 23rd international conference on machine learning* (pp. 705–712). New York, USA: ACM Press.

Ramon, J., & DeRaedt, L. (2000). Multi instance neural networks. In *Proceedings of ICML-2000 workshop on attribute-value and relational learning.*

Ray, S., & Craven, M. (2005). Supervised versus multiple-instance learning: An empirical comparison. In *Proceedings of the 22nd international conference on machine learning* (pp. 697–704). New York: ACM Press.

Ray, S., & Page, D. (2001). Multiple instance regression. In *Proceedings of the 18th international conference on machine learning*. Williamstown, MA: Morgan Kaufmann.

Tao, Q., Scott, S. D., & Vinodchandran, N. V. (2004). SVM-based generalized multiple-instance learning via approximate box counting. In *Proceedings of the 21st international conference on machine learning* (pp. 779–806). San Francisco: Morgan Kaufmann.

Valiant, L. G. (1984). A theory of the learnable. *Communiactions of the ACM, 27*(11), 1134–1142.

Wang, J., & Zucker, J. D. (2000). Solving the multiple-instance problem: A lazy learning approach. In *Proceedings of the 17th international conference on machine learning* (pp. 1119–1125). San Francisco: Morgan Kaufmann.

Weidmann, N., Frank, E., & Pfahringer, B. (2003). A two-level learning method for generalized multi-instance problems. In *Proceedings of the European conference on machine learning* (pp. 468–479). Berlin/Heidelberg: Springer.

Xu, X., & Frank, E. (2004). Logistic regression and boosting for labeled bags of instances. In *Proceedings of the Pacific-Asia conference on knowledge discovery and data mining* (pp. 272–281). Sydney, Australia.

Zhang, Q., & Goldman, S. (2001). EM-DD: An improved multiple-instance learning technique. In *Advances in Neural Information Processing Systems* (pp. 1073–1080). MIT Press.

Zhang, Q., Yu, W., Goldman, S., & Fritts, J. (2002). Content-based image retrieval using multiple-instance learning. In *Proceedings of the 19th international conference on machine learning* (pp. 682–689). San Francisco: Morgan Kaufmann.

Zhou, Z. H., & Zhang, M. L. (2002). Neural networks for multi-instance learning. Technical Report, Nanjing University, Nanjing, China.

Multi-Objective Optimization

Synonyms

MOO; Multi-criteria optimization; Vector optimization

Definition

Multi-criteria optimization is concerned with the optimization of a vector of objectives, which can be the subject of a number of constraints or bounds. The goal of multi-objective optimization is usually to find or to approximate the set of Pareto-optimal solutions. A solution is Pareto-optimal if it cannot be improved in one objective without getting worse in another one.

Multiple Classifier Systems

►Ensemble Learning

Multiple-Instance Learning

►Multi-Instance Learning

Multi-Relational Data Mining

Luc De Raedt
Katholieke Universiteit Leuven,
Heverlee, Belgium

Synonyms

Inductive logic programming; Relational learning; Statistical relational learning

Definition

Multi-relational data mining is the subfield of knowledge discovery that is concerned with the mining of multiple tables or relations in a database. This allows it to cope with structured data in the form of complex data that cannot easily be represented using a single table, or an ►attribute as is common in machine learning.

Relevant techniques of multi-relational data mining include those from relational learning, statistical relational learning, and inductive logic programming.

Cross References

►Inductive Logic Programming

Recommended Reading

Dzeroski, S., & Lavrac, N. (Eds.). (2001). *Relational data mining*. Berlin: Springer.

Multistrategy Ensemble Learning

Definition

Every ►ensemble learning strategy might be expected to have unique effects on the base learner. Combining multiple ensemble learning algorithms might hence be expected to provide benefit. For example, ►MultiBoosting combines ►AdaBoost and a variant of ►Bagging, obtaining most of AdaBoost's ►bias reduction coupled with most of Bagging's ►variance reduction. Similarly, ►Random Forests combines Bagging's variance reduction with ►Random Subspaces' bias reduction.

Cross References

►Ensemble Learning
►MultiBoosting
►Random Forests

Recommended Reading

Webb, G. I., & Zheng, Z. (2004). Multistrategy ensemble learning: Reducing error by combining ensemble learning techniques. *IEEE Transactions on Knowledge and Data Engineering, 16*(8), 980–991.

Must-Link Constraint

A pairwise constraint between two items indicating that they should be placed into the same cluster in the final partition.

N

Naïve Bayes

Geoffrey I. Webb
Monash University, Melbourne, Victoria

Synonyms

Idiot's bayes; Simple bayes

Definition

Naïve Bayes is a simple learning algorithm that utilizes ▶Bayes rule together with a strong assumption that the attributes are conditionally independent, given the class. While this independence assumption is often violated in practice, naïve Bayes nonetheless often delivers competitive classification accuracy. Coupled with its computational efficiency and many other desirable features, this leads to naïve Bayes being widely applied in practice.

Motivation and Background

Naïve Bayes provides a mechanism for using the information in sample data to estimate the posterior probability $P(y \mid \mathbf{x})$ of each class y, given an object $\mathbf{x}$. Once we have such estimates, we can use them for ▶classification or other decision support applications.

Naïve Bayes' many desirable properties include:

- *Computational efficiency*: ▶Training time is linear with respect to both the number of ▶training examples and the number of ▶attributes, and ▶classification time is linear with respect to the number of attributes and unaffected by the number of training examples.
- *Low variance*: Because naïve Bayes does not utilize search, it has low ▶variance, albeit at the cost of high ▶bias.
- *Incremental learning*: Naïve Bayes operates from estimates of low order probabilities that are derived from the training data. These can readily be updated as new training data are acquired.
- *Direct prediction of posterior probabilities*.
- *Robustness in the face of noise*: Naïve Bayes always uses all attributes for all predictions and hence is relatively insensitive to ▶noise in the examples to be classified. Because it uses probabilities, it is also relatively insensitive to noise in the ▶training data.
- *Robustness in the face of missing values*: Because naïve Bayes always uses all attributes for all predictions, if one attribute value is missing, information from other attributes is still used, resulting in graceful degradation in performance. It is also relatively insensitive to ▶missing attribute values in the ▶training data due to its probabilistic framework.

Structure of Learning System

Naïve Bayes is based on ▶Bayes rule

$$P(y \mid \mathbf{x}) = P(y)P(\mathbf{x} \mid y)/P(\mathbf{x}) \quad (1)$$

together with an assumption that the attributes are conditionally independent given the class. For ▶attribute-value data, this assumption entitles

$$P(\mathbf{x} \mid y) = \prod_{i=1}^{n} P(x_i \mid y) \quad (2)$$

where x_i is the value of the ith attribute in $\mathbf{x}$, and n is the number of attributes.

$$P(\mathbf{x}) = \prod_{i=1}^{k} P(c_i)P(\mathbf{x} \mid c_i) \quad (3)$$

where k is the number of classes and c_i is the ith class. Thus, (1) can be calculated by normalizing the numerators of the right-hand-side of the equation.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

For ►categorical attributes, the required probabilities $P(y)$ and $P(x_i \mid y)$ are normally derived from frequency counts stored in arrays whose values are calculated by a single pass through the training data at training time. These arrays can be updated as new data are acquired, supporting ►incremental learning. Probability estimates are usually derived from the frequency counts using smoothing functions such as the ►Laplace estimate or an ►m-estimate.

For ►numeric attributes, either the data are discretized (see ►discretization), or probability density estimation is employed.

In ►document classification, two variants of naïve Bayes are often employed (McCallum and Nigam, 1998). The *multivariate Bernoulli model* utilizes naïve Bayes as described above, with each word in a corpus represented by a binary variable that is true if and only if the word is present in a document. However, only the words that are present in a document are considered when calculating the probabilities for that document.

In contrast, the *multinomial model* uses information about the number of times a word appears in a document. It treats each occurrence of a word in a document as a separate event. These events are assumed independent of each other. Hence the probability of a document given a class is the product of the probabilities of each word event given the class.

Cross References

►Bayes Rule
►Bayesian Methods
►Bayesian Networks
►Semi-Naïve Bayesian Learning

Recommended Reading

Lewis, D. (1998) Naive Bayes at forty: the independence assumption in information retrieval. In *Machine Learning: ECML-98, Proceedings of the 10th European Conference on Machine Learning, Chemnitz, Germany* (pp. 4–15). Berlin: Springer.

McCallum, A., & Nigam, K. (1998). A comparison of event models for Naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization* (pp. 41–48). CA: AAAI Press.

NC-Learning

►Negative Correlation Learning

NCL

►Negative Correlation Learning

Nearest Neighbor

Eamonn Keogh
University California-Riverside

Synonyms

Closest point; Most similar point

Definition

In a data collection M, the *nearest neighbor* to a data object q is the data object M_i, which minimizes dist (q, M_i), where dist is a *distance measure* defined for the objects in question. Note that the fact that the object M_i is the nearest neighbor to q does not imply that q is the nearest neighbor to M_i.

Motivation and Background

Nearest neighbors are useful in many machine learning and data mining tasks, such as classification, anomaly detection, and motif discovery and in more general tasks such as spell checking, vector quantization, plagiarism detection, web search, and recommender systems.

The naive method to find the nearest neighbor to a point q requires a linear scan of all objects in M. Since this may be unacceptably slow for large datasets and/or computationally demanding distance measures, there is a huge amount of literature on speeding up nearest neighbor searches (query-by-content). The fastest methods depend on the distance measure used, whether the data is disk resident or in main memory, and the structure of the data itself. Many methods are based on the R-tree (Guttman, 1984) or one of its variants (Manolopoulos, Nanopoulos, Papadopoulos, and Theodoridis, 2005). However, in recent years there has been an increased awareness that for many applications

approximate nearest neighbors may suffice. This has led to the development of techniques like *locality sensitive hashing*, which finds high-quality approximate nearest neighbors in constant time.

The definition of nearest neighbor allows for the definition of one of the simplest classification schemes, the *nearest neighbor classifier*.

The major database (SIGMOD, VLDB, and PODS) and data mining (SIGKDD, ICDM, and SDM) conferences typically feature several papers on novel distance measures and techniques for speeding up nearest neighbor search. Pavel et al.'s book provides an excellent overview on the state-of-the-art techniques in nearest neighbor searching.

Recommended Reading

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on management of data* (pp. 47–57). New York: ACM. ISBN 0-89791-128-8

Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A. N., & Theodoridis, Y. (2005). R-*trees: Theory and applications*. Berlin: Springer.

Zezula, P., Amato, G., Dohnal, V., & Batko, M. (2005). Similarity search: The metric space approach. In *Advances in database systems* (Vol. 32, p. 220). New York: Springer. ISBN 0-387-29146-6

Nearest Neighbor Methods

►Instance-Based Learning

Negative Correlation Learning

Synonyms

NC-learning; NCL

Definition

Negative correlation learning (Liu & Yao, 1999) is an ►ensemble learning technique. It can be used for regression or classification problems, though with classification problems the models must be capable of producing posterior probabilities. The model outputs are combined with a uniformly weighted average. The squared error is augmented with a penalty term which takes into account the diversity of the ensemble. The error for the *i*th model is,

$$E(f_i(x)) = \frac{1}{2}(f_i(x) - d)^2 - \lambda(f_i(x) - \bar{f}(x))^2. \quad (1)$$

The coefficient λ determines the balance between optimizing individual accuracy, and optimizing ensemble diversity. With $\lambda = 0$, the models are trained independently, with no emphasis on diversity. With $\lambda = 1$, the models are tightly coupled, and the ensemble is trained as a single unit. Theoretical studies (Brown, Wyatt, & Tino, 2006) have shown that NC works by directly optimizing the ►bias-variance-covariance trade-off, thus it explicitly *manages* the ensemble diversity. When the complexity of the individuals is sufficient to have high individual accuracy, NC provides little benefit. When the complexity is low, NC with a well-chosen λ can provide significant performance improvements. Thus the best situation to make use of the NC framework is with a large number of low accuracy models.

Recommended Reading

Brown, G., Wyatt, J. L., & Tino, P. (2006). Managing diversity in regression ensembles. *Journal of Machine Learning Research, 6*, 1621–1650.

Liu, Y., & Yao, X. (1999). Ensemble learning via negative correlation *Neural Networks, 12*(10), 1399–1404.

Negative Predictive Value

Negative Predictive Value (NPV) is defined as a ratio of true negatives to the total number of negatives predicted by a model. This is defined with reference to a special case of the ►confusion matrix with two classes – one designated the *positive* class and the other the *negative* class – as indicated in Table 1.

NPV can then be defined in terms of true negatives and false negatives as follows.

$$\mathrm{NPV} = \mathrm{TN}/(\mathrm{TN} + \mathrm{FN})$$

Negative Predictive Value. Table 1 The outcomes of classification into positive and negative classes

		Assigned Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Network Analysis

►LinkMining and Link Discovery

Network Clustering

►Graph Clustering

Networks with Kernel Functions

►Radial Basis Function Networks

Neural Networks

Neural networks are learning algorithms based on a loose analogy of how the human brain functions. Learning is achieved by adjusting the weights on the connections between nodes, which are analogous to synapses and neurons.

Cross References

►Radial Basis Function Networks

Neural Network Architecture

►Topology of a Neural Network

Neuro-Dynamic Programming

►Value Function Approximation

Neuroevolution

Risto Miikkulainen
The University of Texas at Austin
Austin, TX, USA

Synonyms

Evolving neural networks; Genetic neural networks

Definition

Neuroevolution is a method for modifying ►neural network weights, topologies, or ensembles in order to learn a specific task. Evolutionary computation (see ►Evolutionary Algorithms) is used to search for network parameters that maximize a fitness function that measures performance in the task. Compared to other neural network learning methods, neuroevolution is highly general, allowing learning without explicit targets, with non differentiable activation functions, and with recurrent networks. It can also be combined with standard neural network learning, e.g. to biological adaptation. Neuroevolution can also be seen as a policy search method for reinforcement-learning problems, where it is well suited to continuous domains and to domains where the state is only partially observable.

Motivation and Background

The primary motivation for neuroevolution is to be able to train neural networks in sequential decision tasks with sparse reinforcement information. Most neural network learning is concerned with supervised tasks, where the desired behavior is described in terms of a corpus of input to output examples. However, many learning tasks in the real world do not lend themselves to the supervised learning approach. For example, in game playing, vehicle control, and robotics, the optimal actions at each point in time are not always known; only after performing several actions, it is possible to get information about how well they worked, such as winning or losing the game. Neuroevolution makes it possible to find a neural network that optimizes behavior given only such sparse information about how well the networks are doing, without direct information about what exactly they should be doing.

The main benefit of neuroevolution compared with other reinforcement learning (RL) methods in such tasks is that it allows representing continuous state and action spaces and disambiguating hidden states naturally. Network activations are continuous, and the network generalizes well between continuous values, largely avoiding the state explosion problem that plagues many reinforcement-learning approaches. ►Recurrent networks can encode memories of past states and actions, making it possible to learn in ►partially observable Markov decision process (POMDP) environments that are difficult for many RL approaches.

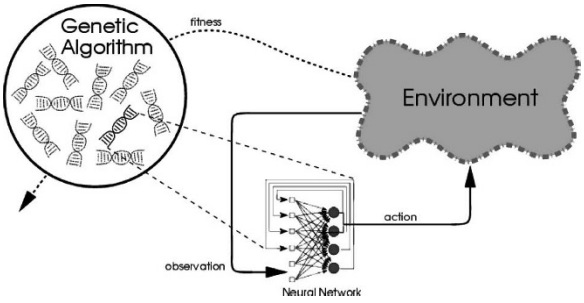
Compared to other neural network learning methods, neuroevolution is highly general. As long as the performance of the networks can be evaluated over time, and the behavior of the network can be modified through evolution, it can be applied to a wide range of network architectures, including those with non differentiable activation functions and recurrent and higher-order connections. While most neural learning algorithms focus on modifying only the weights, neuroevolution can be used to optimize other aspects of the networks as well, including activation functions and network topologies.

Third, neuroevolution allows combining evolution over a population of solutions with lifetime learning in individual solutions: the evolved networks can each learn further through, e.g., backpropagation or Hebbian learning. The approach is therefore well suited for understanding biological adaptation and building artificial life systems.

Structure of the Learning System

Basic methods

In neuroevolution, a population of genetic encodings of neural networks is evolved to find a network that solves the given task. Most neuroevolution methods follow the usual generate-and-test loop of evolutionary algorithms (Fig. 1). Each encoding in the population (a genotype) is chosen in turn and decoded into the corresponding neural network (a phenotype). This network is then employed in the task and its performance measured over time, obtaining a fitness value for the corresponding genotype. After all members of the population have been evaluated in this manner, genetic operators are used to create the next generation of the population. Those encodings with the highest fitness are mutated and crossed over with each other, and the resulting offspring replaces the genotypes with the lowest fitness in the population. The process therefore constitutes an intelligent parallel search towards better genotypes and continues until a network with a sufficiently high fitness is found.



Neuroevolution. Figure 1. Evolving neural networks. A population of genetic neural networks encodings (genotypes) is first created. At each iteration of evolution (generation), each genotype is decoded into a neural network (phenotype), which is evaluated in the task, resulting in a fitness value for the genotype. Crossover and mutation among the genotypes with the highest fitness is then used to generate the next generation

Several methods exist for evolving neural networks depending on how the networks are encoded. The most straightforward encoding, sometimes called conventional neuroevolution (CNE), is formed by concatenating the numerical values for the network weights (either binary or floating point; Floreano, Dürr, & Mattiussi, 2008; Schaffer, Whitley, & Eshelman, 1992; Yao, 1999). This encoding allows evolution to optimize the weights of a fixed neural network architecture, an approach that is easy to implement and is practical in many domains.

In more challenging domains, the CNE approach suffers from three problems. The method may cause the population to converge before a solution is found, making further progress difficult (i.e., premature convergence); similar networks, such as those where the order of nodes is different, may have different encodings and much effort is wasted in trying to optimize them in parallel (i.e., competing conventions); a large number

of parameters need to be optimized at once, which is difficult through evolution.

More sophisticated encodings have been devised to alleviate these problems. One approach is to run the evolution at the level of solution components instead of full solutions. That is, instead of a population of complete neural networks, a population of network fragments, neurons, or connection weights is evolved (Gomez, Schmidhuber, & Miikkulainen, 2008; Moriarty, Schultz, & Grefenstette, 1999; Potter & Jong, 2000). Each individual is evaluated as part of a full network, and its fitness reflects how well it cooperates with other individuals in forming a full network. Specifications for how to combine the components into a full network can be evolved separately, or the combination can be based on designated roles for subpopulations. In this manner, the complex problem of finding a solution network is broken into several smaller subproblems; evolution is forced to maintain diverse solutions, and competing conventions and the number of parameters is drastically reduced.

Another approach is to evolve the network topology, in addition to the weights. The idea is that topology can have a large effect on function, and evolving appropriate topologies can achieve good performance faster than evolving weights only (Angeline, Saunders, Pollack, & An, 1994; Floreano et al., 2008; Stanley & Miikkulainen, 2004; Yao, 1999). Since topologies are explicitly specified, competing conventions are largely avoided. It is also possible to start evolution with simple solutions and gradually make them more complex, a process that takes place in biology and is a powerful approach in machine learning in general. Speciation according to the topology can be used to avoid premature convergence, and to protect novel topological solutions until their weights have been sufficiently optimized.

All of the above methods map the genetic encoding directly to the corresponding neural network, i.e., each part of the encoding corresponds to a part of the network, and vice versa. Indirect encoding, in contrast, specifies a process through which the network is constructed, such as cell division or generation through a grammar (Floreano et al., 2008; Gruau, Whitley, & Adding, 1993; Stanley & Miikkulainen, 2003; Yao, 1999). Such an encoding can be highly compact and also take advantage of modular solutions. The same structures can be repeated with minor modifications, as they often are in biology. It is, however, difficult to optimize solutions produced by indirect encoding, and realizing its full potential is still future work.

The fifth approach is to evolve an ensemble of neural networks to solve the task together, instead of a single network (Liu, Yao, & Higuchi, 2000). This approach takes advantage of the diversity in the population. Different networks learn different parts or aspects of the training data, and together the whole ensemble can perform better than a single network. Diversity can be created through speciation and negative correlation, encouraging useful specializations to emerge. The approach can be used to design ensembles for classification problems, but it can also be extended to control tasks.

Extensions

The basic mechanisms of neuroevolution can be augmented in several ways, making the process more efficient and extending it to various applications. One of the most basic ones is incremental evolution or shaping. Evolution is started on a simple task and once that is mastered, the solutions are evolved further on a more challenging task, and through a series of such transfer steps, eventually on the actual goal task itself (Gomez et al., 2008). Shaping can be done by changing the environment, such as increasing the speed of the opponents, or by changing the fitness function, e.g., by rewarding gradually more complex behaviors. It is often possible to solve challenging tasks by approaching them incrementally even when they cannot be solved directly.

Many extensions to evolutionary computation methods apply particularly well to neuroevolution. For instance, intelligent mutation techniques such as those employed in evolutionary strategies are effective because the weights often have suitable correlations (Igel, 2003). Networks can also be evolved through coevolution (Chellapilla & Fogel, 1999; Stanley & Miikkulainen, 2004). A coevolutionary arms race can be established, e.g., based on complexification of network topology: as the network becomes gradually more complex, evolution is likely to elaborate on existing behaviors instead of replacing them.

On the other hand, several extensions utilize the special properties of the neural network phenotype. For

instance, neuron activation functions, initial states, and learning rules can be evolved to fit the task (Floreano et al., 2008; Yao, 1999; Schaffer et al., 1992). Most significantly, evolution can be combined with other neural network learning methods (Floreano et al., 2008). In such approaches, evolution usually provides the initial network, which then adapts further during its evaluation in the task. The adaptation can take place through Hebbian learning, thereby strengthening those existing behaviors that are invoked often during evaluation. Alternatively, supervised learning such as backpropagation can be used, provided targets are available. Even if the optimal behaviors are not known, such training can be useful. Networks can be trained to imitate the most successful individuals in the population, or part of the network can be trained in a related task such as predicting the next inputs, or evaluating the utility of actions based on values obtained through Q-learning. The weight changes may be encoded back into the genotype, implementing Lamarckian evolution; alternatively, they may affect selection through the Baldwin effect, i.e., networks that learn well will be selected for reproduction even if the weight changes themselves are not inherited (Ackley & Littman, 1992; Bryant & Miikkulainen, 2007; Gruau et al., 1993).

There are also several ways to bias and direct the learning system using human knowledge. For instance, human-coded rules can be encoded in partial network structures and incorporated into the evolving networks as structural mutations. Such knowledge can be used to implement initial behaviors in the population, or it can serve as an advice during evolution (Miikkulainen, Bryant, Cornelius, Karpov, Stanley, & Yong, 2006). In cases where rule-based knowledge is not available, it may still be possible to obtain examples of human behavior. Such examples can then be incorporated into evolution, either as components of fitness or by explicitly training the evolved solutions towards human behavior through, e.g., backpropagation (Bryant & Miikkulainen, 2007). Similarly, knowledge about the task and its components can be utilized in designing effective shaping strategies. In this manner, human expertise can be used to bootstrap and guide evolution in difficult tasks, as well as direct it towards the desired kinds of solutions.

Applications

Neuroevolution methods are powerful especially in continuous domains of reinforcement learning, and those that have partially observable states. For instance, in the benchmark task of balancing the inverted pendulum without velocity information (making the problem partially observable), the advanced methods have been shown to find solutions two orders of magnitude faster than value-function-based reinforcement-learning methods (measured by number of evaluations; Gomez et al., 2008). They can also solve harder versions of the problem, such as balancing two poles simultaneously.

The method is powerful enough to make many real-world applications of reinforcement learning possible. The most obvious area is adaptive, nonlinear control of physical devices. For instance, neural network controllers have been evolved to drive mobile robots, automobiles, and even rockets (Gomez & Miikkulainen, 2003; Nolfi & Floreano, 2000; Togelius & Lucas, 2006). The control approach have been extended to optimize systems such as chemical processes, manufacturing systems, and computer systems. A crucial limitation with current approaches is that the controllers usually need to be developed in simulation and transferred to the real system. Evolution is the strongest as an off-line learning method where it is free to explore potential solutions in parallel.

Evolution of neural networks is a natural tool for problems in artificial life. Because networks implement behaviors, it is possible to design neuroevolution experiments on how behaviors such as foraging, pursuit and evasion, hunting and herding, collaboration, and even communication may emerge in response to environmental pressure (Werner & Dyer, 1992). It is possible to analyze the evolved circuits and understand how they map to function, leading to insights into biological networks (Keinan, Sandbank, Hilgetag, Meilijson, & Ruppin, 2006). The evolutionary behavior approach is also useful for constructing characters in artificial environments, such as games and simulators. Non-player characters in current video games are usually scripted and limited; neuroevolution can be used to evolve complex behaviors for them, and even adapt them in real time (Miikkulainen et al., 2006).

Programs and Data

Software for, e.g., the NEAT method for evolving network weights and topologies, and the ESP method for evolving neurons to form networks is available at http://nn.cs.utexas.edu/keyword?neuroevolution.
The TEEM software for evolving neural networks for robotics experiments is available at http://teem.epfl.ch.
The OpenNERO software for evolving intelligent multiagent behavior in simulated environments is at http://nn.cs.utexas.edu/?opennero.

Cross References

►Evolutionary Algorithms
►Reinforcement Learning

Recommended Reading

Ackley, D., & Littman, M. (1992). Interactions between learning and evolution. In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen (Eds.), *Artificial life II* (pp. 487–509). Reading, MA: Addison-Wesley.

Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks, 5*, 54–65.

Bryant, B. D., & Miikkulainen, R. (2007). Acquiring visibly intelligent behavior with example-guided neuroevolution http://nn.cs.utexas.edu/keyword?bryant:aaai07. In *Proceedings of the twenty-second national conference on artificial intelligence* (pp. 801–808). Menlo Park, CA: AAAI Press.

Chellapilla, K., & Fogel, D. B. (1999). Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE, 87*, 1471–1496.

Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: From architectures to learning. *Evolutionary Intelligence, 1*, 47–62.

Gomez, F., & Miikkulainen, R. (2003). Active guidance for a finless rocket using neuroevolution http://nn.cs.utexas.edu/keyword?gomez:gecco03. In *Proceedings of the genetic and evolutionary computation conference* (pp. 2084–2095). San Francisco: Morgan Kaufmann.

Gomez, F., Schmidhuber, J., & Miikkulainen, R. (2008). Accelerated neural evolution through cooperatively coevolved synapses http://nn.cs.utexas.edu/keyword?gomez:jmlr08. *Journal of Machine Learning Research, 9*, 937–965.

Gruau, F., & Whitley, D. (1993). Adding learning to the cellular development of neural networks: Evolution and the Baldwin effect. *Evolutionary Computation, 1*, 213–233 .

Igel, C. (2003). Neuroevolution for reinforcement learning using evolution strategies http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/igel/NfRLUES.pdf. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, & T. Gedeon, (Eds.), *Proceedings of the 2003 congress on evolutionary computation* (pp. 2588–2595). Piscataway, NJ: IEEE Press.

Keinan, A., Sandbank, B., Hilgetag, C. C., Meilijson, I., & Ruppin, E. (2006). Axiomatic scalable neurocontroller analysis via the Shapley value. *Artificial Life, 12*, 333–352.

Liu, Y., Yao, X., & Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation, 4*, 380–387.

Miikkulainen, R., Bryant, B. D., Cornelius, R., Karpov, I. V., Stanley, K. O., & Yong, C. H. (2006). Computational intelligence in games http://nn.cs.utexas.edu/keyword?miikkulainen:cigames06. In G. Y. Yen & D. B. Fogel (Eds.), *Computational intelligence: Principles and practice* (155–191). Piscataway, NJ: IEEE Computational Intelligence Society.

Moriarty, D. E., Schultz, A. C., & Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research, 11*, 199–229.

Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics*. Cambridge, MA: MIT Press.

Potter, M. A., & Jong, K. A. D. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=retrieve&db=pubmed&dopt=abstract&list_uids=10753229. *Evolutionary Computation, 8*, 1–29.

Schaffer, J. D., Whitley, D., & Eshelman, L. J. (1992). Combinations of genetic algorithms and neural networks: A survey of the state of the art. In D. Whitley & J. Schaffer (Eds.), *Proceedings of the international workshop on combinations of genetic algorithms and neural networks* (pp. 1–37). Los Alamitos, CA: IEEE Computer Society Press.

Stanley, K. O., & Miikkulainen, R. (2003). A taxonomy for artificial embryogeny http://nn.cs.utexas.edu/keyword?stanley:alife03. *Artificial Life, 9*, 93–130.

Stanley, K. O., & Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification http://nn.cs.utexas.edu/keyword?stanley:jair04. *Journal of Artificial Intelligence Research, 21*, 63–100.

Togelius, J., & Lucas, S. M. (2006). Evolving robust and specialized car racing skills http://algoval.essex.ac.uk/rep/games/Togelius2006Evolving.pdf. In *IEEE congress on evolutionary computation* (pp. 1187–1194). Piscataway, NJ: IEEE.

Werner, G. M., & Dyer, M. G. (1992). Evolution of communication in artificial organisms. In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen (Eds.) *Proceedings of the workshop on artificial life (ALIFE '90)* (pp. 659–687). Reading, MA: Addison-Wesley.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE, 87*(9), 1423–1447.

Neuron

Risto Miikkulainen
The University of Texas at Austin
Austin, TX, USA

Synonyms

Node; Unit

Definition

Neurons carry out the computational operations of a network; together with connections (see ►Topology

of a Neural Network, ►Weights), they constitute the neural network. Computational neurons are highly abstracted from their biological counterparts. In most cases, the neuron forms a weighted sum of a large number of inputs (activations of other neurons), applies a nonlinear transfer function to that sum, and broadcasts the resulting output activation to a large number of other neurons. Such activation models the firing rate of the biological neuron, and the nonlinearity is used to limit it to a certain range (e.g., 0/1 with a threshold, $(0\,..\,1)$ with a sigmoid, $(-1\,..\,1)$ with a hyperbolic tangent, or $(0\,..\,\infty)$ with an exponential function). Each neuron may also have a bias weight, i.e., a weight from a virtual neuron that is always maximally activated, which the learning algorithm can use to adjust the input sum quickly into the most effective range of the nonlinearity. Alternatively to firing rate neurons, the firing events (i.e., spikes or action potentials) of the neuron can be represented explicitly. In such an integrate-and-fire approach, each spike causes a change in the neuron's membrane potential that decays over time; an output spike is generated if the potential exceeds a threshold (see ►Biological Learning). In contrast, networks such as ►Self-Organizing Maps and ►Radial Basis Function Networks abstract the firing rate further into a measure of similarity (or distance) between the neuron's input weight vector and the vector of input activities. Learning in neural networks usually takes place by adjusting the weights on the input connections of the neuron, and can also include adjusting the parameters of the nonlinear transfer function, or the neuron's connectivity with other neurons. In this manner, the neuron converges information from other neurons, makes a simple decision based on it, broadcasts the result widely, and adapts.

Node

►Neuron

No-Free-Lunch Theorem

A theorem establishing that performance on test data cannot be deduced from performance on training data. It follows that the justification for any particular learning algorithm must be based on an assumption that nature is uniform in some way. Since different machine learning algorithms make such different assumptions, no-free-lunch theorems have been used to argue that it not possible to deduce that any algorithm is superior to any other from first principles. Thus "good" algorithms are those whose ►inductive bias matches the way the world happens to be.

Nogood Learning

Nogood learning is a ►deductive learning technique used for the purpose of ►intelligent backtracking in constraint satisfaction. The approach analyzes failures at backtracking points and derives sets of variable bindings, or *nogoods*, that will never lead to a solution. These nogood constraints can then be used to prune later search nodes.

Noise

The training data for a learning algorithm is said to be *noisy* if the data contain errors. Errors can be of two types:

- A *measurement error* occurs when some attribute values are incorrect or inaccurate. Note that measurement of physical properties by continuous values is always subject to some error.
- In supervised learning, *classification error* means that a training example has an incorrect class label.

In addition to errors, training examples may have ►missing attribute values. That is, the values of some attribute values are not recorded.

Noisy data can cause learning algorithms to fail to converge to a concept description or to build a concept description that has poor classification accuracy on unseen examples. This is often due to ►over fitting.

For methods to minimize the effects of noise, see ►Over Fitting.

Nominal Attribute

A **nominal attribute** assumes values that classify data into mutually exclusive (nonoverlapping), exhaustive, unordered categories. See ►Attribute and ►Measurement Scales.

Nonparametric Bayesian

►Gaussian Process

Nonparametric Cluster Analysis

►Density-Based Clustering

Non-Parametric Methods

►Instance-Based Learning

Nonstandard Criteria in Evolutionary Learning

Michele Sebag
Université Paris-Sud, Orsay, France

Introduction

Machine learning (ML), primarily concerned with extracting models or hypotheses from data, comes into three main flavors: ►supervised learning also known as ►classification or ►regression (Bishop, 2006; Duda et al., 2001; Han and Kamber, 2000), ►unsupervised learning also known as ►clustering (Ben-David et al., 2005), and ►reinforcement learning (Sutton and Barto, 1998).

All three types of problems can be viewed as optimization problems. The ML core task is to define a *learning criterion* (i.e., the function to be optimized) such that it enforces (i) the statistical relevance of the solution; (ii) the well-posedness of the underlying optimization problem. Since evolutionary computation (see ►Evolutionary Algorithms) makes it possible to handle ill-posed optimization problems, the field of evolutionary learning (Holland, 1986) has investigated quite a few nonstandard learning criteria and search spaces. Only supervised ML will be considered in the following. Unsupervised learning has hardly been touched upon in the evolutionary computation (EC) literature; regarding reinforcement learning, the interested reader is referred to the entries related to ►evolutionary robotics and control.

The entry will first briefly summarize the formal background of supervised ML and its two mainstream approaches for the last decade, namely support vector machines (SVMs) (Cristianini and Shawe-Taylor, 2000; Schölkopf et al., 1998; Vapnik, 1995) and ensemble learning (Breiman 1998; Dietterich, 2000; Schapire, 1990). Thereafter and without pretending to exhaustivity, this entry will illustrate some innovative variants of these approaches in the literature, building upon the evolutionary freedom of setting and tackling optimization problems.

Formal Background

Supervised learning exploits a dataset $\mathcal{E} = \{(\mathbf{x}_i, y_i), \mathbf{x}_i \in X, y_i \in Y, i = 1 \ldots n\}$, where X stands for the instance space (e.g., $\mathbb{R}^d$), Y is the label space, and $(\mathbf{x}_i, y_i)$ is a labeled example, as depicted in Table 1. Supervised learning is referred to as *classification* (respectively *regression*) when Y is a finite set (respectively when $Y = \mathbb{R}$).

The ML goal is to find a hypothesis or classifier $h : X \mapsto Y$ such that $h(x)$ is "sufficiently close" to the true label y of x for any x ranging in the instance domain. It is generally assumed that the available examples are independently and identically distributed (iid) after a probability distribution P_{XY} on $X \times Y$. Letting $\ell(y', y)$ denote the loss incurred by labeling $\mathbf{x}$ as y' instead of its true label y, the learning criterion is most naturally defined as the expectation of the loss, or *generalization*

Nonstandard Criteria in Evolutionary Learning. Table 1 Excerpt of a dataset in a failure identification problem (binary classification). Instance space *X* is the cross product of all attribute domains: for example, attribute *Temperature* ranges in R, attribute *Material* ranges in {*Ni*, *Fe*, ...}. Label space *Y* is binary

	Temperature	Material	Aging	Label
$\mathbf{x}_1$	118.2	Ni	No	Failure
$\mathbf{x}_2$	76.453	Fe	Yes	OK

error, to be minimized, where $\mathcal{H}$ denotes the hypothesis space:

$$\text{Find } h^* = arg\min\{\mathcal{F}(h) = \int \ell(h(x), y)dP(x, y),\ h \in \mathcal{H}\}$$

The generalization error however is not computable, since the joint distribution P_{XY} of instances and labels is unknown; only its approximation on the training set, referred to as *empirical error*, can be computed as follows:

$$\mathcal{F}_e(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(h(\mathbf{x}_i), y_i)$$

Using results from the theory of measure and integration, the generalization error is upper bounded by the empirical error, plus a term reflecting the number of examples and the regularity of the hypothesis class (Fig. 1).

Note that minimizing the empirical error alone leads to the infamous *overfitting* problem: while the predictive accuracy on the training set is excellent, the error on a (disjoint) test set is much higher. All learning criteria thus involve a trade-off between the empirical error and a so-called regularization term, providing good guarantees (upper bound) on the generalization error.

In practice, learning algorithms also involve hyperparameters (e.g., the weight of the regularization term). These are adjusted using cross-validation using a grid search (EC approaches have also been used to find optimal learning hyperparameters, ranging from the topology of neural nets [Miikkulainen et al., 2003], to the kernel parameters in SVM [Friedrichs and Igel, 2005; Mierswa, 2006].) The dataset is divided into K subsets with same class distribution; hypothesis h_i is learned from the training set made of all subsets except the i-th and the empirical error of h_i is measured on the ith subset. An approximation of the generalization error is provided by the average of the h_i errors when $i = 1 \ldots K$, referred to as cross-fold error, and the hyperparameter setting is empirically determined to minimize the cross-fold error.

Support Vector Machines

Considering a real-valued instance space ($X = \mathbb{R}^D$), a linear ▶support vector machine (SVM) (Boser et al., 1992) constructs the separating hyperplane (where $< a, b >$ stands for the dot product of vectors a and b):

$$h(\mathbf{x}) = < w, \mathbf{x} > + b$$

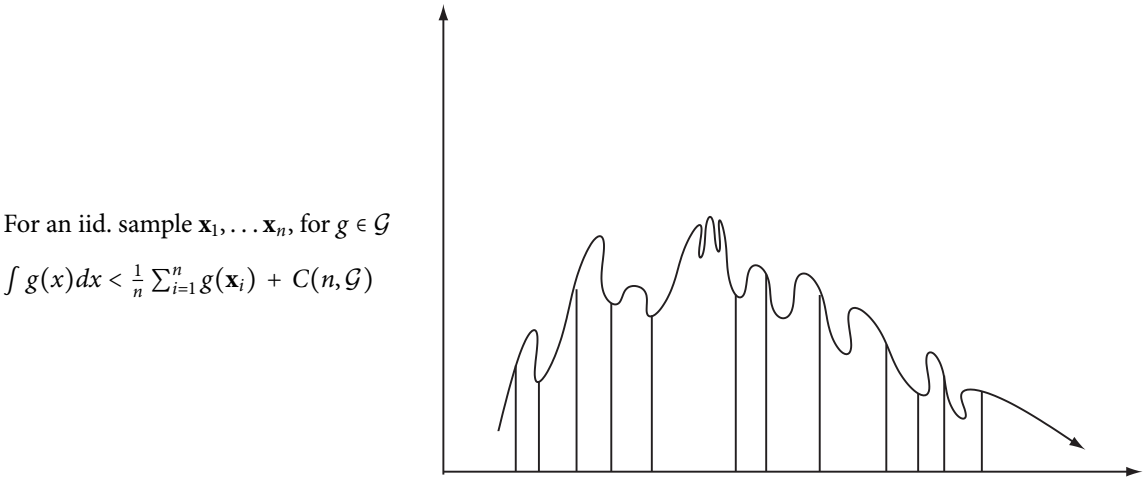
which maximizes the margin that is, the minimal distance between the examples and the hyperplane, when such separating hyperplanes exists (Fig. 2). A slightly more complex formulation, involving the so-called slack variables $\mathbf{x}i_i$, is defined to deal with noise (Cortes and Vapnik, 1995).

The function to be optimized, the L_2 norm of the hyperplane normal vector w, is quadratic; using Lagrange multipliers to account for the constraints gives rise to the so-called dual formulation. Let us call *support vectors* those examples for which the constraint is active (Lagrange multiplier $\alpha_i > 0$), then it becomes

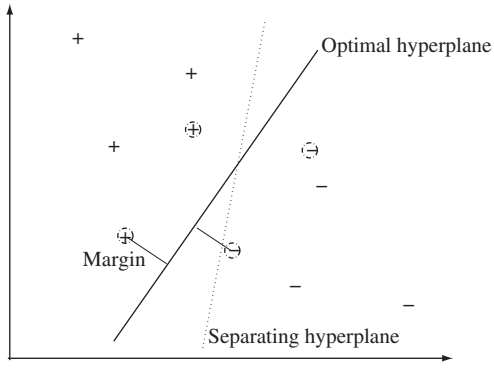
$$h(\mathbf{x}) = \sum y_i\alpha_i < \mathbf{x}_i, \mathbf{x} > + b \quad \text{with } \alpha_i > 0; \quad \sum \alpha_i y_i = 0$$

As will be seen in section "Evolutionary Regularization," this formulation defines a search space, which can be directly explored by EC (Mierswa, 2007).

Obviously however, linear classifiers are limited. The power of SVMs comes from the so-called kernel *trick*, naturally exporting the SVM approach to nonlinear hypothesis spaces. Let us map the instance space X onto some *feature space* X' via mapping Φ. If the scalar product on X' can be computed in X (e.g., $< \Phi(\mathbf{x}), \Phi(\mathbf{x}') > =_{def} K(x, x')$) then a linear classifier in X' (nonlinear with reference to X) is given as



Nonstandard Criteria in Evolutionary Learning. Figure 1. Bounding the integral from the empirical average depending on the uniform sample size and the class of functions $\mathcal{G}$ at hand



without noise

$$\text{Minimize } \tfrac{1}{2}||w||^2$$

$$\text{s.t. for } i = 1 \text{ to } n$$

$$y_i(< w, \mathbf{x}_i > + b) \geq 1$$

with noise

$$\text{Minimize } \tfrac{1}{2}||w||^2 + C\sum_{i=1}^{n} \mathbf{x}i_i$$

$$\text{s.t. for } i = 1 \text{ to } n$$

$$y_i(< w, \mathbf{x}_i > + b) \geq 1 - \mathbf{x}i_i; \quad \mathbf{x}i_i \geq 0$$

Nonstandard Criteria in Evolutionary Learning. Figure 2. Linear support vector machines. The optimal hyperplane is the one maximizing the minimal distance to the examples

$h(\mathbf{x}) = \sum_i y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$. The only requirement is to use a positive definite kernel (ensuring that the underlying optimization problem is well posed). Again, this requirement can be relaxed in the evolutionary learning framework (Mierswa, 2006).

Among the most widely used kernels are the Gaussian kernel $\left(K(x, x') = exp\left\{-\frac{||x-x'||^2}{\sigma^2}\right\}\right)$ and the polynomial kernel $(K(x, x') = (<x, x'> + c)^d)$. The kernel parameters σ, c, d, referred to as learning hyperparameters, have been tuned by some authors using EC, as well as the kernel itself (see among others (Friedrichs and Igel, 2005; Gagné et al., 2006; Mierswa, 2006)).

Ensemble methods

The other mainstream approach in supervised learning, ►ensemble learning (EL), relies on somewhat different principles. Schapire's seminal paper, *The strength of weak learnability*, exploring the relationship between *weak learnability* (ability of building a hypothesis slightly better than random guessing, whatever the distribution of the dataset is (C)) and *strong learnability* (ability of building a hypothesis with arbitrarily high predictive accuracy), established a major and counterintuitive result: strong and weak learnability are equivalent (Schapire, 1990). The idea behind the proof

is that combining many weak hypotheses learned under different distributions yields an arbitrarily accurate hypothesis. As the errors of the weak hypotheses should not concentrate in any particular region of the instance space (for condition C to hold), the law of large numbers states that averaging them leads to exponentially decrease the empirical error.

Two main EL approaches have been investigated in the literature. The first one, ▶bagging (Breiman, 1998), builds a large number of independent hypotheses; the source of variations is bootstrapping (uniformly selecting the training set with replacement from the initial dataset); or varying the parameters of the learning algorithm; or subsampling the features considered at each step of the learning process (Amit et al., 1997; Breiman, 2001). The final classifier is usually obtained by averaging these solutions.

The other EL approach, ▶boosting (Freund and Shapire, 1996), iteratively builds a sequence of hypotheses, where each h_i somehow is in charge of correcting the mistakes of $h_1, \ldots h_{i-1}$. Specifically, a distribution $\mathcal{W}_t$ is defined on the training set at step t, with $\mathcal{W}_0$ being the uniform distribution. At step t, the weight of every example misclassified by h_t is increased (multiplied by $exp\{-h_t(\mathbf{x}_i).h_i\}$; then a normalization step follows to ensure that $\mathcal{W}_{t+1}$ still sums to 1); hypothesis h_{t+1} will thus focus on the examples misclassified by h_t. Finally, the classifier is defined as the weighted vote of all h_t.

The intuition behind boosting is that not all examples are equal: some examples are more difficult than others (more hypotheses misclassify them) and the learning process should thus focus on these examples (with the caveat that a difficult example might be so because it is noisy). Interestingly, the intuition that examples are not equal has been formalized in terms of coevolution (When designing a program, the fitness of the candidate solutions is computed after some test cases; for the sake of accuracy and feasability, the difficulty and number of test cases must be commensurate with the competence of the current candidate solutions. Hillis defined a competitive coevolution setting between the program species and the test case species: while programs aim at solving test cases, test cases aim at defeating candidate programs. This major line of research however is outside the scope of evolutionary learning as it assumes that the whole distribution P_{XY} is known.) by D. Hillis in the early 1990s (Hillis, 1990).

Many empirical studies suggest that boosting is more effective than bagging (with some caveat in the case of noisy domains), thanks to the higher diversity of the boosting ensemble (Dietterich, 2000; Margineantu and Dietterich, 1997).

In the ensemble learning framework, the *margin* of an example $\mathbf{x}$ is defined as the difference between the (cumulated weight or number) of hypotheses labeling $\mathbf{x}$ as positive, and those labeling $\mathbf{x}$ as negative. Like in the SVM framework, the margin of an example reflects the confidence of its classification (how much this example should be perturbed for its label to be modified).

Learning Criteria

Learning criterion and *fitness function* will be used interchangeably in the following. Since Holland's seminal papers on evolutionary learning (Holland, 1986, 1975), the most used learning criterion is the predictive accuracy on the available dataset. After the early 1990s however, drawbacks related to either learning or evolutionary issues motivated the design of new fitness functions.

Evolutionary Regularization

In the ▶genetic programming field, the early use of more sophisticated learning criteria was motivated by the so-called bloat phenomenon (Banzhaf and Langdon, 2002; Poli, 2008), that is, the uncontrolled growth of the solution size as evolution goes on. Two main approaches have been considered. The first one boils down to regularization (section "Formal Background"): the fitness function is composed of the predictive accuracy plus an additional term meant to penalize large-sized solutions (Blickle, 1996). The tricky issue of course is how to adjust the weight of the penalization term; the statistical ML theory offers no principled solution to this issue (except in an asymptotic perspective, when the number of training examples goes to infinity (Gelly et al., 2006)); thus, the weight is adjusted empirically using cross-validation (section "Formal Background").

Another approach (Blickle, 1996) is based on the use of two fitness functions during the same evolution run, after the so-called behavioral memory paradigm (Schoenauer and Xanthakis, 1993). In a first phase, the population is evolved to maximize the predictive accuracy. In a second phase, the optimization goal becomes

to minimize the solution size *while preserving the predictive accuracy* reached in the former phase. As could have been expected, this second approach also depends upon the careful empirical adjustment of hyper-parameters (when to switch from one phase to another one).

Another approach is to consider regularized learning as a multi-objective optimization problem, avoiding the computationally heavy tuning of the regularization weight (Note however that in the case where the regularization involves the L_1 norm of the solution, the Pareto front can be analytically derived using the celebrated LASSO algorithm (Hastie et al., 2004; Tibshirani, 1996).). Mierswa (2007) applies multi-objective evolutionary optimization, specifically NSGA-II ([Deb et al., 2000]; see the Multi-Objective Evolutionary Optimization entry in this encyclopedia), to the simultaneous optimization of the margin and the error. The search space is nicely and elegantly derived from the dual form of SVMs (section "Support Vector Machines"): it consists of vectors $(\alpha_1, \ldots \alpha_n)$, where most α_i are zero and $\sum_i \alpha_i y_i = 0$. A customized mutation operator, similar in spirit to the sequential minimization optimization proposed by Platt [1999], enables to explore the solutions with few support vectors. The Pareto front shows the trade-off between the regularization term and the training error. At some point however, a hold-out (test set) needs be used to detect and avoid overfitting solutions, boiling down to cross-validation. Another multi-objective optimization learning is proposed by Suttorp and Igel (2006) (see section "AUC Area Under the Roc Curve").

Ensemble Learning and Boosting

Ensemble learning and evolutionary computation share two main original features. Firstly, both rely on a population of candidate solutions; secondly, the diversity of these solutions commands the effectiveness of the approach. It is no surprise therefore that evolutionary ensemble learning, tightly coupling EC and EL, has been intensively investigated in the last decade (Another exploitation of the hypotheses built along independent evolutionary learning runs concerns feature selection (Jong et al., 2004), which is outside the scope of this entry.)

A family of diversity-oriented learning criteria has been investigated by Xin Yao and collaborators, switching the optimization goal from "learning the best hypothesis" toward "learning the best ensemble" (Monirul Islam and Yao, 2008). The hypothesis space is that of neural networks (NNs). Nonparametric and parametric operators are used to simultaneously optimize the neural topology and the NN weights. Among parametric operators is the gradient-based backpropagation (BP) algorithm to locally optimize the weights (Rumelhart and McClelland, 1986), combined with simulated annealing to escape BP local minima.

Liu et al. (2000) enforce the diversity of the networks using a *negative correlation* learning criterion. Specifically, the BP algorithm is modified by replacing the error of the t-th NN on the i-th example with a weighted sum of this error and the error of the ensemble of the other NNs; denoting H_{-t} the ensemble made of all NNs but the tth one:

$$(h_t(\mathbf{x}_i) - y_i)^2 \rightarrow (1-\lambda)(h_t(\mathbf{x}_i) - y_i)^2 + \lambda(H_{-t}(\mathbf{x}_i) - y_i)^2)$$

Moreover, ensemble negative correlation–based learning exploits the fact that not all examples are equal, along the same line as boosting (section "Ensemble Methods"): to each training example is attached a weight, reflecting the number of hypotheses that misclassify it; finally the fitness associated to each network is the sum of the weights of all examples it correctly classifies. While this approach nicely suggests that ensemble learning is a multiple objective optimization (MOO) problem (minimize the error rate and maximize the diversity), it classically handles the MOO problem as a fixed weighted sum of the objectives (the value of parameter λ is fixed by the user).

The MOO perspective is further investigated by Chandra and Yao in the DIVACE system, enforcing the multilevel evolution of ensemble of classifiers (Chandra and Yao, 2006a,b). In (Chandra and Yao, 2006b), the top-level evolution simultaneously minimizes the error rate (accuracy) and maximizes the negative correlation (diversity). In (Chandra and Yao, 2006a), the negative correlation-inspired criterion is replaced by a *pairwise failure crediting*; the difference concerns the misclassification of examples that are correctly classified by other classifiers. Several heuristics have been investigated to construct the ensemble from the last population, based on averaging the hypothesis values, using the (weighted) vote of all hypotheses, or selecting a subset of hypotheses, for example, by clustering the final

hypothesis population after their phenotypic distance, and selecting a hypothesis in each cluster.

Gagné et al. (2007) tackle both the construction of a portfolio of classifiers, and the selection of a subset thereof, either from the final population only as in (Chandra and Yao, 2006a,b), or from all generations. In order to do so, a reference set of classifiers is used to define a dynamic optimization problem: the fitness of a candidate hypothesis reflects whether h improves on the reference set; in the meantime, the reference set is updated every generation. Specifically, noting w_i the fraction of reference classifiers misclassifying the i-th example, $\mathcal{F}(h)$ is set to the sum of w_i^γ, taken over all examples correctly classified by h. Parameter γ is used to mitigate the influence of noisy examples.

Boosting and Large-Scale Learning

Another key motivation for designing new learning criteria is to yield scalable learning algorithms, coping with giga or terabytes of data (see [Sonnenburg et al., 2008]).

Song et al. (2003, 2005) presented an elegant genetic programming approach to tackle the intrusion detection challenge (Lippmann et al., 2000); this challenge offers a 500,000 pattern training set, exceeding standard available RAM capacities. The proposed approach relies on the dynamic subset selection method first presented by Gathercole and Ross (1994). The whole dataset is equally and randomly divided into subsets $\mathcal{E}_i$ with same distribution as the whole dataset, where each $\mathcal{E}_i$ fits within the available RAM. Iteratively, some subset $\mathcal{E}_i$ is selected with uniform probability, and loaded in memory; it is used for a number of generations set to $G_{max} \times Err(i)$ where G_{max} is the user-supplied maximum number of generations, and $Err(i)$ is the minimum number of patterns in $\mathcal{E}_i$ misclassified the previous time $\mathcal{E}_i$ was considered. Within $\mathcal{E}_i$, a competition is initiated between training patterns to yield a frugal yet challenging assessment of the hypotheses. Specifically, every generation or so, a restricted subset is selected by tournament in $\mathcal{E}_i$, considering both the difficulty of the patterns (the difficulty of pattern $\mathbf{x}_j$ being the number of hypotheses misclassifying $\mathbf{x}_j$ last time $\mathbf{x}_j$ was selected) and its age (the number of generations since $\mathbf{x}_j$ was last selected). With some probability (30% in the experiments), the tournament returns the pattern with maximum age; otherwise, it returns the pattern with maximum difficulty.

The dynamic selection subset (DSS) heuristics can thus be viewed as a mixture of uniform sampling (modeled by the age-based selection) and boosting (corresponding to the difficulty-based selection). This mixed distribution gets the best of both worlds: it speeds up learning by putting the stress on the most challenging patterns, akin boosting; in the meanwhile, it prevents noisy examples from leading learning astray as the training set always includes a sufficient proportion of uniformly selected examples. The authors report that the approach yields accurate classifiers (though outperformed by the Challenge winning entry), while one trial takes 15 min on a modest laptop computer (1 GHz Pentium, 256 MB RAM).

Gagné et al., aiming at the scalable optimization of SVM kernels, proposed another use of dynamic selection subset in a coevolutionary perspective (Gagné et al., 2006). Specifically, any kernel induces a similarity on the training set

$$s(\mathbf{x}, \mathbf{x}') = 2K(\mathbf{x}, \mathbf{x}') - K(\mathbf{x}, \mathbf{x}) - K(\mathbf{x}', \mathbf{x}')$$

This similarity directly enables the classification of examples along the k-nearest neighbor approach (Duda et al., 2001) (see ▶Nearest Neighbor), labeling an example after the majority of its neighbors. Inspired from (Gilad-Bachrach et al., 2004), the margin of an example is defined as the rank of its closest neighbor in the same class, minus the rank of its closest neighbor in the other class (the closer a neighbor, the higher its rank is). The larger the margin of an example, the more confident one can be it will be correctly classified; the fitness of the kernel could thus be defined as the sum of the example margins. Computed naively however, this fitness would be quadratic in the size of the training set, hindering the scalability of the approach.

A three-species coevolutionary framework was thus defined. The first species is that of kernels; the second species includes the candidate neighbor instances, referred to as prototypes; the third species includes the training instances, referred to as test cases. Kernels and prototypes undergo a cooperative co-evolution: they

cooperate to yield the underlying metric (similarity) and the reference points (prototypes) enabling to classify all training instances. The test cases, in the meanwhile, undergo a competitive coevolution with the other two species: they present the learning process with more and more difficult training examples, aiming at a good coverage of the whole instance space. The approach reportedly yields accurate kernels at a moderate computational cost.

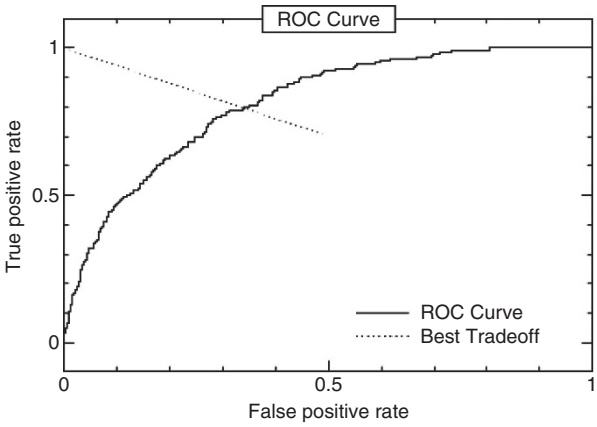
AUC: Area Under the ROC Curve

The misclassification rate criterion is notably ill-suited to problem domains with a minority class. If the goal is to discriminate a rare disease (< 1% of the training set) from a healthy state, the default hypothesis ("everyone is healthy" with 1% misclassified examples) can hardly be outperformed in terms of predictive accuracy. Standard heuristics accommodating ill-balanced problems involve the oversampling of the minority class, undersampling of the majority class, or cost-sensitive loss function (e.g., misclassifying a healthy person for an ill one costs 1, whereas the opposite costs 100) (Domingos, 1999).

Another principled approach is based on the so-called area under the receiver-operating characteristics curve (see ►ROC Analysis). Let us consider a continuous hypothesis h, mapping the instance space on the real-value space $\mathbb{R}$. For each threshold τ let the binary classifier h_τ be defined as instance x is positive iff $h(x) > \tau$. To each τ value can be associated the true positive (TP) rate (fraction of ill persons that are correctly classified) and the false positive (FP) rate (fraction of healty persons misclassified as ill ones). In the (FP,TP) plane, the curve drawn as τ varies defines the ROC curve (Fig. 3).

Noting that the ideal classifier lies in the upper left corner (0% false positive rate, 100% true positive rate), it comes naturally to optimize the area under the ROC curve. This criterion, also referred to as Wilcoxon rank test, has been intensively studied in both theoretical and algorithmic perspectives (see among many others (Cortes and Mohri, 2004; Ferri et al., 2002; Joachims, 2005; Rosset, 2004)).

The AUC criterion has been investigated in the EC literature since the 1990s (Fogel et al., 1998), for it defines a combinatorial optimization problem. Considering the search space of real-valued functions, mapping instance space X onto $\mathbb{R}$, the AUC (Wilcoxon) criterion is defined as



Nonstandard Criteria in Evolutionary Learning. Figure 3. The receiver operating characteristic (ROC) Curve depicts how the true positive (TP) rate increases vs the false positive (FP) rate. Random guessing corresponds to the diagonal line. The ROC curve is insensitive to ill-balanced distributions as TP and FP rates are normalized

$$\mathcal{F}(h) = Pr(h(\mathbf{x}) > h(\mathbf{x}')|y > y')$$

$$\mathcal{F}_e(h) \propto \#\{(\mathbf{x}_i, \mathbf{x}_j)\ s.t.\ h(\mathbf{x}_i) > h(\mathbf{x}_j), y_i = 1, y_j = 0\}$$

Specifically, hypothesis h is used to rank the instances; any ranking such that all positive instances are ranked before the negative ones gets the optimal AUC. The fitness criterion can be computed with complexity $\mathcal{O}(n \log n)$ where n stands for the number of training instances, by showing that

$$\mathcal{F}_e(h) \propto \sum_{i=1\ldots n, y_i=1} i \times rank(i)$$

Interestingly, the optimization of the AUC criterion can be dealt with in the SVM framework, as shown by Joachims (2005), replacing class constraints by inegality constraints (Fig. 2):

$$y_i(< w, \mathbf{x}_i > + b) \geq 1 \quad i = 1 \ldots n$$

$$\rightarrow \quad < w, \mathbf{x}_i - \mathbf{x}_j > \ \geq 1 \quad i, j = 1 \ldots n,\ s.t.\ y_i > y_j$$

In practice, the quadratic optimization process introduces gradually the violated constraints only, to avoid dealing with a quadratic number of constraints.

The flexibility of EC can still allow for more specific and application-driven interpretation of the AUC criterion. Typically in medical applications, the physician is most interested in the beginning of the AUC curve, trying to find a threshold τ retrieving a high fraction of ill patients for a very low false positive rate. The same situation occurs in customer relationship management, replacing positive cases by potential churners. The AUC criterion can be easily adapted to minimize the number of false positive within the top k-ranked individuals, as shown by Mozer et al., (2001).

In a statistical perspective however (and contrarily to a common practice in the ML and data mining communities), it has been argued that selecting a classifier based on its AUC was not appropriate (David J. Hand, 2009). The objection is that the AUC maximization yields the best hypothesis *under a uniform distribution of the misclassification costs*, whereas hypothesis h is used with a specific threshold τ, corresponding to a particular point of the ROC curve (Fig. 3).

Still, ROC curves convey very clear intuitions about the trade-off between TP and FP rates; analogous to a Pareto front, they enable one to select a posteriori the best trade-off according to a one's implicit preferences. An interesting approach along these lines has been investigated by Suttorp and Igel (2006) to learn SVMs, using a multi-objective optimization setting to simultaneously minimize the FP rate, and maximize the TP rate, and maximize the number of support vectors.

The last objective actually corresponds to a regularization term: the empirical error plus the number of support vectors upper-bounds the so-called leave-one-out error (when the number of folds in cross-fold validation is set to the number of examples), since the hypothesis is not modified when removing a non-support vectors. (see Zhang [2003] for more detail).

Conclusions

Unsurprisingly, the bottom line of evolutionary learning matches that of EC: any effort to customize the fitness function is highly rewarded; a good knowledge of the domain application enables to choose appropriate, frugal yet effective, search space and variation operators.

Another message concerns the validation of the proposed approaches. In early decades, hypotheses were assessed from their training error, with poor applicative relevance due to overfitting. Better practices are now widely used (e.g., training, validation, and test sets); as advocated by Dietterich (1998), good practices are based on cross-validation. Taking into account early remarks about the University of California Irvine (UCI) repository (Holte, 1993), experimental validation should consider actually challenging problems.

Due to space limitations, this entry has excluded some nice and elegant work at the crossroad of machine learning and evolutionary computation, among others, interactive optimization and modelisation of the user's preferences (Llorà et al., 2005), interactive feature construction (Krawiec and Bhanu, 2007; Venturini et al., 1997), or ML-based heuristics for noisy optimization (Heidrich-Meisner and Igel, 2009).

Recommended Reading

Amit, Y., Geman, D., & Wilder, K. (1997). Joint induction of shape features and tree classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 19*(11), 1300–1305.

Banzhaf, W., & Langdon, W. B. (2002). Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines, 3*(1), 81–91.

Ben-David, S., von Luxburg, U., Shawe-Taylor, J., & Tishby, N. (Eds.). (2005). *Theoretical foundations of clustering*. NIPS Workshop.

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer.

Blickle, T. (1996). Evolving compact solutions in genetic programming: a case study. In H.-M. Voigt et al. (Eds.), *Proceedings of the 4th international inference on parallel problem solving from nature. Lecture notes in computer science* (vol. 1141, pp. 564–573). Berlin: Springer.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the 5th annual ACM conference on Computational Learning Theory, COLT'92*, (pp. 144–152). Pittsburgh, PA.

Breiman, L. (1998). Arcing classifiers. *Annals of Statistics, 26*(3), 801–845.

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32.

Chandra, A., & Yao, X. (2006). Ensemble learning using multi-objective evolutionary algorithms. *Journal of Mathematical Modelling and Algorithms, 5*(4), 417–425.

Chandra, A., & Yao X. (2006). Evolving hybrid ensembles of learning machines for better generalisation. *Neurocomputing, 69*, 686–700.

Cortes, C., & Vapnik, V. N. (1995). Support-vector networks. *Machine Learning, 20*, 273–297.

Cortes, C., & Mohri, M. (2004). Confidence intervals for the area under the ROC curve. *Advances in Neural Information Processing Systems, NIPS*, 17.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge: Cambridge University Press.

David J. Hand. (2009). Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine Learning*, *77*(1), 103–123. http://dx.doi.org/10.1007/S10994-009-5119-5, DBLP, http://dblp.uni-trier.de

Deb, K., Agrawal, S., Pratab, A., & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. Schoenauer et al. (Eds.), *Proceedings of the parallel problem solving from nature VI conference*, Paris, France, pp. 849–858. Springer. Lecture Notes in Computer Science No. 1917.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation, 10*, 1895–1923.

Dietterich, T. (2000). Ensemble methods in machine learning. In J. Kittler & F. Roli (Eds.), *First International Workshop on Multiple Classifier Systems*, Springer, pp. 1–15.

Domingos, P. (1999). Meta-cost: a general method for making classifiers cost sensitive. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge discovery and data mining*, (pp. 155–164). San Diego, CA: ACM.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley.

Ferri, C., Flach, P. A., & Hernndez-Orallo, J. (2002). Learning decision trees using the area under the ROC curve. In C. Sammut & A. G. Hoffman (Eds.), *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, (pp. 179–186). Morgan Kaufmann.

Fogel, D. B., Wasson, E. C., Boughton, E. M., Porto, V. W., and Angeline, P. J. (1998). Linear and neural models for classifying breast cancer. *IEEE Transactions on Medical Imaging, 17*(3), 485–488.

Freund, Y., & Shapire, R. E. (1996). Experiments with a new boosting algorithm. In L. Saitta (Ed.), *Proceedings of the Thirteenth International Conference on Machine Learning (ICML 1996)*, (pp. 148–156). Bari: Morgan Kaufmann.

Friedrichs, F., & Igel, C. (2005). Evolutionary tuning of multiple SVM parameters. *Neurocomputing, 64*(C), 107–117.

Gagné, C., Schoenauer, M., Sebag, M., & Tomassini, M. (2006). Genetic programming for kernel-based learning with co-evolving subsets selection. In T. P. Runarsson, H.-G. Beyer, E. K. Burke, J. J. Merelo Guervós, L. Darrell Whitley, & X. Yao (Eds.), *Parallel problem solving from nature – PPSN IX, volume 4193 of Lecture Notes in Computer Science* (pp. 1008–1017). Springer.

Gagné, C., Sebag, M., Schoenauer, M., & Tomassini, M. (2007). Ensemble learning for free with evolutionary algorithms? In H. Lipson (Ed.), *Genetic and Evolutionary Computation Conference, GECCO 2007*, (pp. 1782–1789). ACM.

Gathercole, C., & Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In *Parallel problem solving from nature – PPSN III, volume 866 of lecture notes in computer science* (pp. 312–321). Springer.

Gelly, S., Teytaud, O., Bredeche, N., & Schoenauer, M. (2006). Universal consistency and bloat in GP: Some theoretical considerations about genetic programming from a statistical learning theory viewpoint. *Revue d'Intelligence Artificielle, 20*(6), 805–827.

Gilad-Bachrach, R., Navot, A., & Tishby, N. (2004). Margin based feature selection – theory and algorithms. *Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2009)*, ACM Press, p. 43.

Han, J., & Kamber, M. (2000). *Data mining: concepts and techniques*. New York: Morgan Kaufmann.

Hastie, T., Rosset, S., Tibshirani, R., & Zhu, J. (2004). The entire regularization path for the support vector machine. *Advances in Neural Information Processing Systems, NIPS* 17.

Heidrich-Meisner, V., & Igel, C. (2009). Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML 2009)*, ACM, pp. 401–408.

Hillis, W. D. (1990). Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D, 42*, 228–234.

Holland, J. (1986). Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: an artificial intelligence approach* (vol. 2, pp. 593–623). Morgan Kaufmann.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning, 11*, 63–90.

Monirul Islam, M., & Yao, X. Evolving artificial neural network ensembles. In J. Fulcher & L. C. Jain (Eds.), *Computational intelligence: a compendium, volume 115 of studies in computational intelligence* (pp. 851–880). Springer.

Joachims, T. (2005). A support vector method for multivariate performance measures. In L. De Raedt & S. Wrobel (Eds.), *Proceedings of the Twenty-second International Conference on Machine Learning (ICML 2009), volume 119 of ACM International Conference Proceeding Series* (pp. 377–384). ACM.

Jong, K., Marchiori, E., & Sebag, M. (2004). Ensemble learning with evolutionary computation: application to feature ranking. In X. Yao et al. (Eds.), *Parallel problem solving from nature – PPSN VIII, volume 3242 of lecture notes in computer science* (pp. 1133–1142). Springer.

Miikkulainen, R., Stanley, K. O., & Bryant, B. D. (2003). Evolving adaptive neural networks with and without adaptive synapses. *Evolutionary Computation, 4*, 2557–2564.

Krawiec, K., & Bhanu, B. (2007). Visual learning by evolutionary and coevolutionary feature synthesis. *IEEE Transactions on Evolutionary Computation, 11*(5), 635–650.

Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., & Das, K. (2000). Analysis and results of the 1999 DARPA on-line intrusion detection evaluation. In H. Debar, L. Mé, & S. F. Wu (Eds.), *Recent advances in intrusion detection, volume 1907 of lecture notes in computer science* (pp. 162–182). Springer.

Liu, Y., Yao, X., & Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation, 4*(4), 380–387.

Llorà, X., Sastry, K., Goldberg, D. E., Gupta, A., & Lakshmi, L. (2005). Combating user fatigue in igas: partial ordering, support vector machines, and synthetic fitness. In H.-G. Beyer & U.-M. O'Reilly (Eds.), *Genetic and Evolutionary Computation Conference (GECCO 05)*, ACM, pp. 1363–1370.

Margineantu, D., & Dietterich, T. G. (1997). Pruning adaptive boosting. *Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1996)*, Morgan Kaufmann, pp. 211–218.

Mierswa, I. Evolutionary learning with kernels: a generic solution for large margin problems. In M. Cattolico (Ed.), *Genetic and Evolutionary Computation Conference (GECCO 06)*, ACM, pp. 1553–1560.

Mierswa, I. (2007). Controlling overfitting with multi-objective support vector machines. In H. Lipson (Ed.), *Genetic and Evolutionary Computation Conference (GECCO 07)*, pp. 1830–1837.

Mozer, M. C., Dodier, R., Colagrosso, M. C., Guerra-Salcedo, C., & Wolniewicz, R. (2001). Prodding the ROC curve: constrained optimization of classifier performance. *Advances in Neural Information Processing Systems*, NIPS, MIT Press.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf et al. (Eds.), *Advances in kernel methods – support vector learning*. Morgan Kaufmann.

Poli, R. (2008). Genetic programming theory. In C. Ryan & M. Keijzer (Eds.), *Genetic and evolutionary computation conference, GECCO 2008*, (Companion), ACM, pp. 2559–2588.

Rosset, S. (2004). Model selection via the auc. *Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2009), volume 69 of ACM International Conference Proceeding Series*. ACM.

Rumelhart, D. E., & McClelland, J. L. (1990). *Parallel distributed processing*. Cambridge: MIT Press.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*, 197.

Schoenauer, M., & Xanthakis, S. Constrained GA optimization. In S. Forrest (Ed.), *Proceedings of the 5th International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 573–580.

Schölkopf, B., Burges, C., & Smola, A. (1998). *Advances in Kernel methods: support vector machines*. Cambridge: MIT Press.

Song, D., Heywood, M. I., & Nur Zincir-heywood, A. (2003). A linear genetic programming approach to intrusion detection. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Springer, pp. 2325–2336, LNCS 2724.

Song, D., Heywood, M. I., & Nur Zincir-Heywood, A. (2005). Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Transactions on Evolutionary Computation, 9*(3), 225–239.

Sonnenburg, S., Franc, V., Yom-Tov, E., & Sebag, M. (Eds.). (2008). *Large scale machine learning challenge*. ICML Workshop.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning*. Cambridge: MIT Press.

Suttorp, T., & Igel, C. (2006). Multi-objective optimization of support vector machines. In Y. Jin (Ed.), *Multi-objective Machine Learning, volume 16 of Studies in Computational Intelligence* (pp. 199–220). Springer.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Royal Statistical Society, B, 58*(1), 267–288.

Vapnik, V. N. (1995). *The nature of statistical learning*. New York: Springer.

Venturini, G., Slimane, M., Morin, F., & Asselin de Beauville, J. P. (1997). On using interactive genetic algorithms for knowledge discovery in databases. In Th. Bäck, (Ed.), *International Conference on Genetic Algorithms (ICGA)*, Morgan Kaufmann, pp. 696–703.

Zhang, T. (2003). Leave-one-out bounds for kernel methods. *Neural Computation, 15*(6), 1397–1437.

Nonstationary Kernels

►Local Distance Metric Adaptation

Nonstationary Kernels Supersmoothing

►Locally Weighted Regression for Control

Normal Distribution

►Gaussian Distribution

NP-Completeness

Definition

A *decision problem* consists in identifying symbol strings, presented as inputs, that have some specified property. The output consists in a yes/no or 0/1 answer. A decision problem belongs to the class P if there exists an algorithm, that is, a deterministic procedure, for deciding any instance of the problem in a length of time bounded by a polynomial function of the length of the input.

A decision problem is in the class NP if it is possible for every yes-instance of the problem to verify in polynomial time, after having been supplied with a polynomial-length *witness*, that the instance is indeed of the desired property.

An example is the problem to answer the question for two given numbers n and m whether n has a divisor d strictly between m and n. This problem is in NP: if the answer is positive, then such a divisor d will be a witness, since it can be easily checked that d lies between the required bounds, and that n is indeed divisible by d. However, it is not known whether this decision problem is in P or not, as it may not be easy to find a suitable divisor d, even if one exists.

The class of *NP-complete* decision problems contains such problems in NP for which if some algorithm decides it, then every problem in NP can be decided in polynomial time. A theorem of Stephen Cook and Leonid Levin states that such decision problems exist. Several decision problems of this class are problems on ►graphs.

Recommended Reading

Stephen Cook (1971). The complexity of theorem proving procedures. Proceedings of the third annual ACM symposium on theory of computing, 151–158.

Leonid Levin (1973). Universal'nye pereborne zadachi. *Problemy Peredachi Informatsii 9*(3): 265–266.

English translation, Universal Search Problems, in B. A. Trakhtenbrot (1984). A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms. *Annals of the History of Computing* 6(4): 384–400.

Numeric Attribute

Synonyms

Quantitative attribute

Definition

Numeric attributes are numerical in nature. Their values can be ranked in order and can be subjected to meaningful arithmetic operations. See ►Attribute and ►Measurement Scales.

O

Object

►Instance

Object Consolidation

►Entity Resolution

Object Space

►Example Space

Observation Language

Hendrik Blockeel
Katholieke Universiteit Leuven, Belgium
Leiden Institute of Advanced Computer Science
The Netherlands

Synonyms

Instance language

Definition

The *observation language* used by a machine learning system is the language in which the observations it learns from are described.

Motivation and Background

Most machine learning algorithms can be seen as a procedure for deriving one or more hypotheses from a set of observations. Both the input (the observations) and the output (the hypotheses) need to be described in some particular language and this language is called the observation language or the ►Hypothesis Language respectively. These terms are mostly used in the context of symbolic learning, where these languages are often more complex than in subsymbolic or statistical learning.

The following sections describe some of the key observation languages.

Attribute-Value Learning

Probably the most used setting in machine learning is the *attribute-value* setting (see ►Attribute-Value Learning). Here, an example (observation) is described by a fixed set of attributes, each of which is given a value from the domain of the attribute. Such an observation is often called a vector or, in relational database terminology, a tuple. The attributes are usually atomic (i.e., not decomposable in component values) and single-valued (i.e., an attribute has only one value, not a set of values). So we have an instance space (or space of observations)

$$\mathcal{O} = A_1 \times \cdots \times A_n,$$

elements of which are denoted using an observation language that typically has the same structure:

$$\mathcal{L}_O = \mathcal{L}_{A_1} \times \cdots \times \mathcal{L}_{A_n}$$

(the language contains tuples of objects that represent the attribute values).

The attribute-value framework easily allows for both supervised and unsupervised learning; in the supervised learning setting, the label of an instance is simply included as an attribute in the tuple, where as for unsupervised learning, it is excluded.

The attribute-value setting assumes that all instances can be represented using the same fixed set of attributes. When instances can be of different types or are variable-sized (e.g., when an instance is set-valued), this assumption may not hold, and more powerful languages may have to be used instead.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Learning from Graphs, Trees, or Sequences

We here consider the case in which a single instance is a graph, or a node in a graph. Note that trees and sequences are special cases of graphs.

A graph is defined as a pair (V, E), where V is a set of vertices and E a set of edges each edge being a pair of vertices. If the pair is ordered, the graph is directed; otherwise it is undirected. For simplicity, we restrict ourselves to undirected graphs.

A graph can, in practice, not be encoded in attribute-value format without the loss of information. That is, one could use a number of properties of graphs as attributes in the encoding, but several graphs may then still map onto the same representation, which implies loss of information. In theory, one could imagine defining a total order on (certain classes of) graphs and representing each graph by its rank in that order (which is a single numerical attribute), thus representing graphs as numbers without loss of information; but then it is not obvious how to map patterns in this numerical representation to patterns in the original representation. No such approaches have been proposed till now.

Describing the instance space is more difficult here than in the attribute value case. Consider a task of graph classification, where in observations are of the form (G, y) with G a graph and y a value for a target attribute Y. Then we can define the instance space as

$$\mathcal{O} = \{(V, E) | V \subseteq \mathbf{N} \wedge E \subseteq V^2\} \times Y,$$

where $\mathbf{N}$ is the set of all natural numbers. (For each graph, there exists a graph defined over $\mathbf{N}$ that is isomorphic with it, so $\mathcal{O}$ contains all possible graphs up to isomorphism.)

A straightforward observation language in the case of graph classification is then

$$\{(G, y) | G = (V, E) \wedge V \subseteq \mathcal{L}_V \wedge E \subseteq V^2 \wedge y \in Y\},$$

where $\mathcal{L}_V$ is some alphabet for representing nodes.

In learning from graphs, there are essentially two settings: those where a prediction is made for entire graphs, and those where a prediction is made for single nodes in a graph. In the first case, observations are of the form (G, y), where as, in the second case, they are of the form (G, v, y), where $G = (V, E)$ and $v \in V$. That is, a node is given together with the graph in which it occurs (its "environment"), and a prediction is to be made for this specific node, using the information about its environment.

In many cases, the set of observations one learns from is of the form (G, v_i, y_i), where each instance is a different node of exactly the same graph G. This is the case when, for instance, classifying web pages, we take the whole web as their environment.

In a labeled graph, labels are associated with each node or edge. Often these are assumed atomic, being elements of a finite alphabet or real numbers, but they can also be vectors of reals.

Relational Learning

In ▶relational learning, it is assumed that relationships may exist between different instances of the instance space, or an instance may internally consist of multiple objects among which relationships exist.

This essentially corresponds to learning from graphs, except that in a graph only one binary relation exists (the edges E), whereas here there may be multiple relations and they may be non binary. The expressiveness of the two settings is the same, however, as any relation can be represented using only binary relations.

In the attribute-value setting, one typically uses one table where each tuple represents all the relevant information for one observation. In the relational setting, there may be multiple tables, and information on a single instance is contained in multiple tuples, possibly belonging to multiple relations.

Example 1 *Assume we have a database about students, courses, and professors (see Fig. 1). We can define a single observation as all the information relevant to one student, that is: the name, year of entrance, etc. of the student and also the courses they take and the professors teaching these courses.*

Anne	1997
Bernard	1999
Celine	1996
Daniel	1999
Elisa	1997
Fabian	1999

Anne	Algebra	1998	A
Anne	Calculus	1998	B
Bernard	Databases	2000	A
Celine	Biology	1999	B
Celine	Databases	2000	B
Celine	Calculus	1998	A

Algebra
Biology
Calculus
Databases

Adams	Algebra	1998
Adams	Calculus	1999
Baeck	Biology	1999
Cools	Calculus	1998
Cools	Databases	1999

Adams
Baeck
Cools

Observation Language. Figure 1. A small database of students

The most obvious link to the graph representation is as follows: create one node for each tuple, labeled with that tuple, and create a link between two nodes if the corresponding tuples are connected by a foreign key relationship.

Defining a single observation as a set of tuples that are connected through foreign keys in the database corresponds to representing each observation (G, v, y) as (G', v, y), where G' is the connected component of G that contains v. The actual links are usually not explicitly written in this representation, as they are implicit: there is an edge between two tuples if they have the same value for a foreign key attribute.

Inductive Logic Programming

In ▶inductive logic programming, a language based on first order logic is used to represent the observations. Typically, an observation is then represented by a *ground fact*, which basically corresponds to a single tuple in a relational database. In some settings an observation is represented by an *interpretation*, a set of ground facts, which corresponds to the set of tuples mentioned in the previous subsection.

While the target variable can always be represented as an additional attribute, ILP systems often learn from examples and counterexamples of a concept. The target variable is then implicit: it is true or false depending on whether the example is in the positive or negative set, but it is not explicitly included in the fact.

Typical for the inductive logic programming setting is that the input of a system may contain, besides the observations, background knowledge about the application domain. The advantage of the ILP setting is that no separate language is needed for such background knowledge: the same first order logic-based language can be used for representing the observations as well as the background knowledge.

Example 2 *Take the following small dataset:*

```
sibling(bart,lisa).
sibling(lisa,bart).
:- sibling(bart, bart).
:- sibling(lisa, lisa).
father(homer, bart).
mother(marge, bart).
father(homer, lisa).
mother(marge, lisa).
```

There are positive and negative (preceded by `:-`*) examples of the Sibling relation. The following hypothesis might be learned:*

```
sibling(X,Y) :- father(Z,X),
  father(Z,Y), X ≠ Y.
sibling(X,Y) :- mother(Z,X),
  mother(Z,Y), X ≠ Y.
```

If the following clauses as included as background knowledge:

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

then the same ILP system might learn the following more compact definition:

```
sibling(X,Y) :- parent(Z,X),
parent(Z,Y), X ≠ Y.
```

Further Reading

Most of the literature on hypothesis and observation languages is found in the area of inductive logic programming. Excellent starting points to become familiar with this field are *Relational Data Mining* by Lavrač and Džeroski (2001) and *Logical and Relational Learning* by De Raedt (2008).

De Raedt (1998) compares a number of different observation and hypothesis languages with respect to their expressiveness, and indicates relationships between them.

Cross References

▶Hypothesis Language
▶Inductive Logic Programming
▶Relational Learning

Recommended Reading

De Raedt, L. (1998). Attribute-value learning versus inductive logic programming: the missing links (extended abstract). In D. Page (Ed.), *Proceedings of the eighth international conference on inductive logic programming. Lecture notes in artificial intelligence* (Vol. 1446, pp. 1–8). Berlin: Springer.

De Raedt, L. (2008). *Logical and relational learning*. Berlin: Springer.

Džeroski, S., & Lavrač, N. (Eds.). (2001). *Relational data mining*. Berlin: Springer. vfill

Occam's Razor

Geoffrey I. Webb
Monash University, Victoria 3800, Australia

Synonyms

Ockham's razor

Definition

Occam's Razor is the maxim that "entities are not to be multiplied beyond necessity," or as it is often interpreted in the modern context "of two hypotheses H and H', both of which explain E, the simpler is to be preferred" (Good, 1977).

Motivation and Background

Most attempts to learn a ►model from ►data confront the problem that there will be many models that are consistent with the data. In order to learn a single model, a choice must be made between the available models. The factors taken into account by a learner in choosing between models are called its ►inductive biases (Mitchell, 1980). A preference for simple models is a common inductive bias and is embodied in many learning techniques including ►pruning, ►minimum message length and ►minimum description length. ►Regularization is also sometimes viewed as an application of Occams' Razor.

Occam's Razor is an imperative, rather than a proposition. That is, it is neither true nor false. Rather, it is a call to act in a particular way without making any claim about the consequences of doing so. In machine learning the so-called *Occam thesis* is sometimes assumed, that

- given a choice between two plausible classifiers that perform identically on the training set, the simpler classifier is expected to classify correctly more objects outside the training set (Webb, 1996).

While there are many practical advantages in having an inductive bias toward simple models, there remains controversy as to whether the Occam thesis is true (Blumer, Ehrenfeucht, Haussler, & Warmuth, 1987; Domingos, 1999; Webb, 1996).

Recommended Reading

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). Occam's razor. *Information Processing Letters*, *24*(6), 377–380.

Domingos, P. (1999). The role of Occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, *3*(4), 409–425.

Good, I. J. (1977). Explicativity: A mathematical theory of explanation with statistical applications. *Proceedings of the Royal Society of London Series A*, *354*, 303–330.

Mitchell, T. M. (1980). *The need for biases in learning generalizations*. Tech. Rep. CBM-TR-117. Rutgers University, Department of Computer Science.

Webb, G. I. (1996). Further experimental evidence against the utility of Occams razor. *Journal of Artificial Intelligence Research*, *4*, 397–417; Menlo Park: AAAI Press.

Ockham's Razor

►Occam's Razor

Offline Learning

►Batch Learning

One-Step Reinforcement Learning

►Associative Reinforcement Learning

Online Learning

Peter Auer
University of Leoben,
Leoben, Austria

Synonyms

Mistake-bounded learning; Perceptron; Prediction with expert advice; Sequential prediction

Definition

In the online learning model the learner needs to make predictions about a sequence of instances, one after the other, and receives a reward or loss after each prediction. Typically, the learner receives a description of

the instance before making a prediction. The goal of the learner is to maximize the accumulated reward (or equivalently minimize the accumulated losses).

The online learning model is essentially a worst-case model of learning, as it makes no statistical assumptions on how the sequence of inputs and rewards is generated. In particular, it is not assumed that inputs and observations are generated by a probability distribution. In contrast, they might be generated by an adversary who tries to fool the learner.

To compensate for the adversarial nature of the model, in most cases the performance guarantees for online learning algorithms are relative to the performance of the best predictor from a certain class. Often these performance guarantees are quite strong, showing that the learner can do nearly as well as the best predictor from a large class of predictors.

Motivation and Background

Online learning is one of the main models of learning theory, complementing the statistical approach of the ►PAC learning model by making no statistical assumptions. The distinctive properties of the online learning model are:

- Learning proceeds in trials.
- There is no designated learning phase, but performance of the learner is evaluated for each trial.
- No assumptions on the generation of the inputs to the learner are made; they may depend even on previous predictions of the learner.
- In most cases no assumptions on the losses or rewards are made; they may be selected by an adversary.
- The sequential predictions model an interaction between the learner and its environment.
- Performance guarantees for learning algorithms are typically in terms of the performance of the best predictor from some given class, after some number of observations.

The first explicit models of online learning were proposed by Angluin (1988) and Littlestone (1988), but related work on repeated games by Hannan (1957) dates back to 1957. Littlestone proposed online learning as a sequence of trials, in each of which the learner receives some input, makes a prediction of the associated output, and receives the correct output. It was assumed that some function from a known class maps the inputs to correct outputs. The performance of the learner is measured by the number of mistakes made by a learner, before it converges to the correct predictor. Angluin's equivalence query model of learning is formulated differently but is essentially equivalent to Littlestone's model.

Many (including Littlestone (1991), Vovk (1990), and Littlestone and Warmuth (1994)) later removed the restriction that there must be a function that correctly predicts all the outputs. In their setting the learner competes with the best predictor from a given class. As the class of predictors can be seen as a set of experts advising the learner about the correct predictions, this led to the term "prediction with expert advice." A comprehensive treatment of binary predictions with expert advice can be found in Cesa-Bianchi et al. (1997). Relations of online learning to several other fields (e.g., compression, competitive analysis, game theory, and portfolio selection) are discussed in the excellent book on sequential prediction by Cesa-Bianchi and Lugosi (2006).

Structure of Learning System

The online learning model is formalized as follows. In each trial $t = 1, 2, \ldots$, the learner

1. Receives input $x_t \in X$
2. Makes prediction $y_t \in Y$
3. Receives response $z_t \in Z$
4. Incurs loss $\ell_t = \ell(y_t, z_t)$

where $\ell : Y \times Z \mapsto \mathbf{R}$ is some loss function. The performance of a learner up to trial T is measured by its accumulated loss $L_T = \sum_{t=1}^{T} \ell_t$.

Performance bounds for online learning algorithms are, typically, in respect to the performance of an optimal predictor (or expert) E^* from some class $\mathcal{E}$, $E^* \in \mathcal{E}$. A predictor E maps the past given by $(x_1, y_1, z_1), \ldots, (x_{t-1}, y_{t-1}, z_{t-1})$ and the current input x_t to a prediction y_t^E. As for the learner, the performance of a predictor is measured by its accumulated loss $L_T^E = \sum_{t=1}^{T} \ell_t^E$, where $\ell_t^E = \ell\left(y_t^E, z_t\right)$. Most bounds for the loss of online algorithms are of the form

$$L_T \le a \min_{E \in \mathcal{E}} L_T^E + b\mathcal{C}(\mathcal{E}),$$

where the constants a and b depend on the loss function and $\mathcal{C}(\mathcal{E})$ measures the complexity of the class of predictors. (e.g., the complexity $\mathcal{C}(\mathcal{E})$ could be $\log|\mathcal{E}|$ for a finite class $\mathcal{E}$.) Often it is possible to trade the constant a against the constant b such that bounds

$$L_T \le L_T^* + o\left(L_T^*\right)$$

can be achieved, where $L_T^* = \min_{E\in\mathcal{E}} L_T^E$ is the loss of the best predictor up to time T. These bounds are of particular interest as they show that the loss of the learning algorithm is only little larger than the loss of the best predictor. For such bounds the regret R_T of the learning algorithm,

$$R_T = L_T - L_T^*,$$

is the relevant quantity that measures the cost of not knowing the best predictor in advance. Again, it needs to be emphasized that these bounds hold for any sequence of inputs and responses without any additional assumptions. Such bounds are achieved by online learning algorithms that rely on the outputs of the predictors in $\mathcal{E}$ to form their own predictions.

The next section makes this general definition of online learning more concrete by presenting some important online learning algorithms, and it also discusses the related equivalence query model.

Theory/Solution

The Weighted Majority Algorithm

The weighted majority algorithm developed by Littlestone and Warmuth (1994) is one of the fundamental online learning algorithms, with many relatives using similar ideas. It will be presented for the basic scenario with a finite set of experts $\mathcal{E}$, binary predictions $y_t \in \{0,1\}$, binary responses $z_t \in \{0,1\}$, and the discrete loss which just counts mistakes, $\ell(y,z) = |y-z|$, such that $\ell(y,z) = 0$ if $y = z$ and $\ell(y,z) = 1$ if $y \neq z$. (We will use the terms experts and predictors interchangeably. In the literature finite sets of predictors are mostly called experts.)

The weighted majority algorithm maintains a weight w_t^E for each expert $E \in \mathcal{E}$ that are initialized as $w_1^E = 1$. The weights are used to combine the predictions y_t^E of the experts by a weighted majority vote: $y_t = 1$ if $\sum_E w_t^E y_t^E \ge \frac{1}{2}\sum_E w_t^E$, and $y_t = 0$ otherwise. After receiving the response z_t, the weights of experts that made incorrect predictions are reduced by multiplying with some constant $\beta < 1$, $w_{t+1}^E = \beta w_t^E$ if $y_t^E \neq z_t$, and $w_{t+1}^E = w_t^E$ if $y_t^E = z_t$. As a performance bound for the weighted majority algorithm one can achieve

$$L_T \le 2L_T^* + 2\sqrt{2L_T^*\log|\mathcal{E}|} + 4\log|\mathcal{E}|$$

with $L_T^* = \min_{E\in\mathcal{E}} L_T^E$ and an appropriate β. (Better constants on the square root and the logarithmic term are possible.)

While in this bound the loss of the deterministic weighted majority algorithm is twice the loss of the best expert, the randomized version of the weighted majority algorithm almost achieves the loss of the best expert. Instead of using a deterministic prediction, the randomized weighted majority algorithm tosses a coin and predicts $y_t = 1$ with probability $\sum_E w_t^E y_t^E / \sum_E w_t^E$.

Since a prediction of the randomized algorithm matches the prediction of the deterministic algorithm with probability at least 1/2, an incorrect prediction of the deterministic algorithm implies that the randomized algorithm will predict incorrectly with probability at least 1/2. Thus, the loss of the deterministic algorithm is at most twice the expected loss of the randomized algorithm. This can be used to transfer bounds for the randomized algorithm to the deterministic algorithm.

Below, the following bound will be proved on the expected loss of the randomized algorithm,

$$\mathbf{E}[L_T] \le \frac{\log(1/\beta)}{1-\beta} L_T^* + \frac{1}{1-\beta}\log|\mathcal{E}|. \tag{1}$$

Approximately optimizing for β yields $\beta = 1-\varepsilon$, where $\varepsilon = \min\left\{1/2, \sqrt{2(\log|\mathcal{E}|)/L_T^*}\right\}$, and

$$\mathbf{E}[L_T] \le L_T^* + \sqrt{2L_T^*\log|\mathcal{E}|} + 2\log|\mathcal{E}|. \tag{2}$$

The expectation in these bounds is only in respect to the randomization of the algorithm, no probabilistic assumptions on the experts or the sequence of responses are made. These bounds hold for any set of experts and any fixed sequence of responses. This type of bounds assumes that the sequence of inputs and responses does not depend on the randomization of the algorithm. If the inputs x_t and the responses z_t may depend on the past predictions of the algorithm, $y_1, \ldots, y_{t-1}$, then

L_T^* also becomes a random variable and the following bound is achieved:

$$\mathbf{E}[L_T] \le \mathbf{E}[L_T^*] + \sqrt{2\mathbf{E}[L_T^*]\log|\mathcal{E}|} + 2\log|\mathcal{E}|.$$

It can be even shown that the following similar bound holds with probability $1-\delta$ (in respect to the randomization of the algorithm):

$$L_T \le L_T^* + \sqrt{T\log(|\mathcal{E}|/\delta)}.$$

The proof of bound (1) shows many of the ideas used in the proofs for online learning algorithms. Key ingredients are a potential function and how the changes of the potential function relate to losses incurred by the learning algorithm. For the weighted majority algorithm a suitable potential function is the sum of the weights, $W_t = \sum_E w_t^E$. Then, since the losses are 0 or 1,

$$\frac{W_{t+1}}{W_t} = \frac{\sum_E w_{t+1}^E}{\sum_E w_t^E} = \frac{\sum_E \beta^{\ell_t^E} w_t^E}{\sum_E w_t^E} = \frac{\sum_E \left[1-(1-\beta)\ell_t^E\right] w_t^E}{\sum_E w_t^E}$$
$$= 1-(1-\beta)\frac{\sum_E \ell_t^E w_t^E}{\sum_E w_t^E}.$$

Since the probability that the randomized weighted majority algorithm makes a mistake is given by $\mathbf{E}[\ell_t] = \sum_E \ell_t^E w_t^E / \sum_E w_t^E$, we get by taking logarithms that

$$\log W_{t+1} - \log W_t = \log(1-(1-\beta)\mathbf{E}[\ell_t]) \le -(1-\beta)\mathbf{E}[\ell_t]$$

(since $\log(1-x) \le -x$ for $x \in (0,1)$). Summing over all trials $t = 1,\ldots,T$ we find

$$\log W_{T+1} - \log W_1 \le -(1-\beta)\mathbf{E}[L_t].$$

Since $W_1 = |\mathcal{E}|$ and $W_{T+1} = \sum_E w_{T+1}^E = \sum_E \beta^{L_T^E} \ge \beta^{L_T^*}$, rearranging the terms gives (1).

Extensions and Modifications of the Weighted Majority Algorithm Variants and improved versions of the weighed majority algorithm have been analyzed for various learning scenarios. An excellent coverage of the material can be found in Cesa-Bianchi and Lugosi (2006). This section mentions a few of them.

General loss functions. The analysis of the weighted majority algorithm can be generalized to any convex set of predictions Y and any set of outcomes Z, as long as the loss function $\ell(y,z)$ is bounded and convex in the first argument. Vovk (1998) analyzed for quite general Y, Z, and loss functions ℓ, which constants a and b allow a learning algorithm with loss bound

$$L_T \le aL_T^* + b\log|\mathcal{E}|.$$

Of particular interest is the smallest b for which a loss bound with $a = 1$ can be achieved.

Tracking the best expert and other structured experts. For a large number of experts, the loss bound of the weighted majority algorithm is still interesting since it scales only logarithmically with the number of experts. Nevertheless, the weighted majority algorithm and other online learning algorithms become computationally demanding as they need to keep track of the performance of all experts (computation time scales linearly with the number of experts). If the experts exhibit a suitable structure, then this computational burden can be avoided.

As an example, the problem of tracking the best expert is considered. Let $\mathcal{E}_0$ be a small set of base experts. The learning algorithm is required to compete with the best sequence of at most S experts from $\mathcal{E}_0$: the trials are divided into S periods, and in each period another expert might predict optimally. Thus, the minimal loss of a sequence of S experts is given by

$$L_{T,S}^* = \min_{0=T_0\le T_1\le T_2\le\cdots\le T_S=T} \sum_{i=1}^{S} \min_{E\in\mathcal{E}_0} \sum_{t=T_{i-1}+1}^{T_i} \ell_t^E,$$

where the trials are optimally divided into S periods $[T_{i-1}+1, T_i]$, and the best base expert is chosen for each period. Such sequences of base experts can be seen as experts themselves, but the number of such compound experts is $\binom{T-1}{S-1}|\mathcal{E}_0|^S$ and thus computationally prohibitive. Fortunately, a slightly modified weighted majority algorithm applied to the base experts, achieves almost the same performance as the weighted majority algorithm applied to the compound experts (Herbster & Warmuth, 1998). The modification of the weighted majority algorithm just lower bounds the relative weight of each base expert. This allows the relative weight of a base expert to grow large quickly if this expert predicts best in the current period. Hence, also the learning algorithm will predict almost optimally in each period.

Other examples of structured experts include tree experts and shortest path problems (see Cesa-Bianchi and Lugosi (2006) for further references).

The doubling trick. The optimal choice of β in the performance bound (1) requires knowledge about the loss of the best expert L_T^*. If such knowledge is not available, the doubling trick can be used. The idea is to start with an initial guess $\hat{L}^*$ and choose β according to this guess. When the loss of the best expert exceeds this guess, the guess is doubled, β is modified, and the learning algorithm is restarted. The bound (2) increases only slightly when L_T^* is not known and the doubling trick is used. It can be shown that still

$$\mathbf{E}[L_T] \le L_T^* + c_1\sqrt{L_T^* \log|\mathcal{E}|} + c_2 \log|\mathcal{E}|$$

for suitable constants c_1 and c_2. A thorough analysis of the doubling trick can be found in Cesa-Bianchi et al. (1997). Variations of the doubling trick can be used for many online learning algorithms to "guess" unknown quantities. A drawback of the doubling trick is that it restarts the learning algorithm and forgets about all previous trials. An alternative approach is an iterative adaptation of the parameter β, which can be shown to give better bounds than the doubling trick. The advantage of the doubling trick is that its analysis is quite simple.

Follow the perturbed leader. *Follow the perturbed leader* is a simple prediction strategy that was originally proposed by Hannan (1957). In each trial t, it generates identically distributed random values ψ_t^E for every expert E, adds these random values to the losses of the experts so far, and predicts with the expert that achieves the minimum sum,

$$\hat{E}_t = \arg\min_{E\in\mathcal{E}} L_{t-1}^E + \psi_t^E,$$
$$y_t = y_t^{\hat{E}_t}.$$

For suitably chosen distributions of the ψ_t^E, this simple prediction strategy achieves loss bounds similar to the more involved weighted majority like algorithms.

Prediction with limited feedback and the multiarmed bandit problem. In some online learning scenarios, the learner might not receive the original response z_t but only a projected version $\tilde{z}_t = \zeta(y_t, z_t)$ for some $\zeta : Y \times Z \to \tilde{Z}$. The value of the incurred loss $\ell(y_t, z_t)$ might be unknown to the learner. A general model for this situation is called *prediction with partial monitoring*. With suitable assumptions there are prediction strategies for partial monitoring that achieve a regret of order $O(T^{2/3})$.

A special case of partial monitoring is the multiarmed bandit problem. In the multiarmed bandit problem the learner chooses a prediction $y_t \in Y = \{1, \ldots, K\}$ and receives the loss of the chosen prediction $\ell_t(y_t) = \ell(y_t, z_t)$. The losses of the other predictions, $\ell_t(y)$, $y \ne y_t$, are not revealed to the learner. The goal of the learner is to compete with the loss of the single best prediction, $L_T^* = \min_{y\in Y} L_T^y$, $L_T^y = \sum_{t=1}^T \ell_t(y)$. The multiarmed bandit problem looks very much like the original online learning problem with the predictions $y \in Y$ as experts. But the main difference is that in the multiarmed bandit problem only the loss of the *chosen* expert/prediction is revealed, while in the original online learning problem the losses of *all* experts can be calculated by the learner. Therefore, algorithms for the multiarmed bandit problem estimate the unseen losses and use these estimates to make their predictions. Since accurate estimates need a sufficient amount of data, this leads to a trade-off between choosing the (apparently) best prediction to minimize loss, and choosing another prediction for which more data need to be collected. This exploration–exploitation trade-off also appears elsewhere in online learning, but it is most clearly displayed in the bandit problem. An algorithm that deals well with this trade-off is again a simple variant of the weighted majority algorithm. This algorithm does exploration trials with some small probability, and in such exploration trials it chooses a prediction uniformly at random. This algorithm has been analyzed in Auer, Cesa-Bianchi, Freund, and Schapire (2002) for gains instead of losses. Formally, this is equivalent to considering negative losses, $\ell \in [-1, 0]$, with the equivalent gain $g = -\ell$. For losses $\ell \in [-1, 0]$ a bound for the algorithm is

$$\mathbf{E}[L_T] \le L_T^* + 3\sqrt{K|L_T^*|\log K},$$

which for gains translates into

$$\mathbf{E}[G_T] \ge G_T^* - 3\sqrt{KG_T^* \log K}$$

where G_T and G_T^* denote the accumulated gains. Compared with (2), the regret increases only by a factor of $\sqrt{K}$. Auer et al. (2002) show that the order of the regret is essentially optimal. They also present similar bounds that hold with high probability, and analyze several extensions of the bandit problem.

The Perceptron Algorithm This section considers an example for an online learning algorithm that competes with a *continuous* set of experts, in contrast to the *finite* sets of experts considered so far. This algorithm – the perceptron algorithm (Rosenblatt 1958)– was among the first online learning algorithms developed. Another of this early online learning algorithms with a continuous set of experts is the Winnow algorithm by Littlestone (1988). A unified analysis of these algorithms can be found in Cesa-Bianchi and Lugosi (2006). This analysis covers a large class of algorithms, in particular the p-norm perceptrons (Grove, Nittlestone & Schuurmans, 2001), which smoothly interpolate between the perceptron algorithm and Winnow.

The perceptron algorithm aims at learning a linear classification function. Thus inputs are from a Euclidean space, $X = \mathbf{R}^d$, the predictions and responses are binary, $Y = Z = \{0,1\}$, and the discrete misclassification loss is used. Each expert is a linear classifier, represented by its weight vector $v \in \mathbf{R}^d$, whose linear classification is given by $\Phi_v : X \to \{0,1\}$, $\Phi_v(x) = 1$ if $v \cdot x \geq 0$ and $\Phi_{v,\theta}(x) = 0$ if $v \cdot x < 0$.

The perceptron algorithm maintains a weight vector $w_t \in \mathbf{R}^d$ that is initialized as $w_1 = (0, \ldots, 0)$. After receiving input x_t, the perceptron's prediction is calculated using this weight,

$$y_t = \Phi_{w_t}(x_t),$$

and the weight vector is updated,

$$w_{t+1} = w_t + \eta(z_t - y_t)x_t,$$

where $\eta > 0$ is a learning rate parameter. Thus, if the prediction is correct, $y_t = z_t$, then the weights are not changed. Otherwise, the product $w_{t+1} \cdot x_t$ is moved into the correct direction: since $w_{t+1} \cdot x_t = w_t \cdot x_t + \eta(z_t - y_t)||x_t||^2$, $w_{t+1} \cdot x_t > w_t \cdot x_t$ if $y_t = 0$ but $z_t = 1$, and $w_{t+1} \cdot x_t < w_t \cdot x_t$ if $y_t = 1$ but $z_t = 0$.

It may be assumed that the inputs are normalized, $||x_t|| = 1$, otherwise a normalized x_t can be used in the update of the weight vector. Furthermore, it is noted that the learning rate η is irrelevant for the performance of the perceptron algorithm, since it scales only the size of the weights but does not change the predictions. Nevertheless, the learning rate is kept since it will simplify the analysis.

Analysis of the perceptron algorithm. To compare the perceptron algorithm with a fixed (and optimal) linear classifier v a potential function $||w_t - v||^2$ is again used. For the change of the potential function when $y_t \neq z_t$, one finds

$$\begin{aligned}
&||w_{t+1} - v||^2 - ||w_t - v||^2 \\
&= ||w_t + \eta(z_t - y_t)x_t - v||^2 - ||w_t - v||^2 \\
&= ||w_t - v||^2 + 2\eta(z_t - y_t)(w_t - v) \cdot x_t \\
&\quad + \eta^2(z_t - y_t)^2||x_t||^2 - ||w_t - v||^2 \\
&= 2\eta(z_t - y_t)(w_t \cdot x_t - v \cdot x_t) + \eta^2.
\end{aligned}$$

Since $w_t \cdot x_t < 0$ if $y_t = 0$ and $w_t \cdot x_t \geq 0$ if $y_t = 1$, we get $(z_t - y_t)(w_t \cdot x_t) \leq 0$ and

$$||w_{t+1} - v||^2 - ||w_t - v||^2 \leq -2\eta(z_t - y_t)(v \cdot x_t) + \eta^2.$$

Analogously, the linear classifier v makes a mistake in trial t if $(z_t - y_t)(v \cdot x_t) < 0$, and in this case $-(z_t - y_t)(v \cdot x_t) \leq ||v||$. Hence, summing over all trials (where $y_t \neq z_t$) gives

$$\begin{aligned}
&||w_{T+1} - v||^2 - ||w_1 - v||^2 \\
&\quad \leq -2\eta \sum_{t:\ell_t=1,\ell_t^v=0} |v \cdot x_t| + 2\eta||v||L_T^v + \eta^2 L_T,
\end{aligned}$$

where the sum is over all trials where the perceptron algorithm makes a mistake but the linear classifier v makes no mistake. To proceed, it is assumed that for the correct classifications of the linear classifier v, the product $v \cdot x_t$ is bounded away from 0 (which describes the decision boundary). It is assumed that $|v \cdot x_t| \geq \gamma_v > 0$. Then

$$\begin{aligned}
||w_{T+1} - v||^2 - ||w_1 - v||^2 \leq &-2\eta\gamma_v (L_T - L_T^v) \\
&+ 2\eta||v||L_T^v + \eta^2 L_T,
\end{aligned}$$

and

$$L_T(2\eta\gamma_v - \eta^2) \leq ||v||^2 + L_T^v(2\eta\gamma_v + 2\eta||v||),$$

since $\|w_{T+1} - v\|^2 \geq 0$ and $w_1 = (0, \ldots, 0)$. For $\eta = \gamma_v$ the following loss bound for the perceptron algorithm is achieved:

$$L_T \leq \|v\|^2/\gamma_v^2 + 2L_T^v(1 + \|v\|/\gamma_v).$$

Thus, the loss of the perceptron algorithm does not only depend on the loss of a (optimal) linear classifier v, but also on the gap by which the classifier can separate the inputs with $z_t = 0$ from the inputs with $z_t = 1$. The size of this gap is essentially given by $\gamma_v/\|v\|$.

Relation between the perceptron algorithm and support vector machines. The gap $\gamma_v/\|v\|$ is the quantity maximized by support vector machines, and it is the main factor determining the prediction accuracy (in a probabilistic sense) of a support vector machine. It is not coincidental that the same quantity appears in the performance bound of the perceptron algorithm, since it measures the difficulty of the classification problem.

As for support vector machines, kernels $K(\cdot, \cdot)$ can be used in the perceptron algorithm. For that, the dot product $w_t \cdot x_t$ is replaced by the kernel representation $\sum_{\tau=1}^{t-1}(z_\tau - y_\tau)K(x_\tau, x)$. Obviously this has the disadvantage that all previous inputs for which mistakes were made must be kept available.

Learning with Equivalence Queries In the *learning with equivalence queries* model (Angluin, 1988) the learner has to identify a function $f : X \to \{0,1\}$ from a class $\mathcal{F}$ by asking equivalence queries. An equivalence query is a hypothesis $h : X \to \{0,1\}$ about the function f. If $h = f$ then the answer to the equivalence query is YES, otherwise a counterexample $x \in X$ with $f(x) \neq h(x)$ is returned. The performance of a learning algorithm is measured by the number of counter examples received as response to equivalence queries.

The equivalence query model is essentially equivalent to the online learning model with $\mathcal{F}$ as the set of experts, $Y = Z = \{0,1\}$, and the discrete misclassification loss. First, it is considered how a learner for the equivalence query model can be used as a learner in the online learning model: For making a prediction y_t, the equivalence query learner uses its current hypothesis, $y_t = h_t(x_t)$. If the prediction is correct, the hypothesis is not changed. If the prediction is incorrect, the input x_t is used as a counterexample for the hypothesis h_t.

Second, it is considered how a deterministic online learner can be used in the equivalence query model. The current prediction function of the online learner that maps inputs to predictions, can be used as a hypothesis h_t in the equivalence query model. A counterexample x_t is interpreted as an input for which the response z_t disagrees with the prediction y_t of the online learner. Thus the online learner might update its prediction function, which gives a new hypothesis for an equivalence query.

These reductions show that the counterexamples of the equivalence query learner and the mistakes of the online learner coincide. Thus, the performance bounds for the equivalence query model and the online model are the same. (Here it is assumed that there is an expert that incurs no loss. If this is not the case, then an extension of the equivalence query model can be considered, where a number of equivalence queries might be answered with incorrect counterexamples.)

Cross References

►Incremental Learning

Recommended Reading

Angluin, D. (1988). Queries and concept learning. *Machine Learning, 2*, 319–342.

Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. (2002). The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing, 32*, 48–77.

Cesa-Bianchi, N., Freund, Y., Haussler, D., Helmbold, D., Schapire, R., & Warmuth, M. (1997). How to use expert advice. *Journal of the ACM, 44*, 427–485.

Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games*. New York, USA: Cambridge University Press.

Grove, A. J., Nittlestone, N., & Schuurmans, D. (2001). General convergence results for linear discriminant updates. *Machine Learning, 43*, 173–210.

Hannan, J. (1957). Approximation to Bayes risk in repeated play. *Contributions to the Theory of Games, 3*, 97–139.

Herbster, M., & Warmuth, M. (1998). Tracking the best expert. *Machine Learning, 32*, 151–178.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning, 2*, 285–318.

Littlestone, N. (1991). Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. In *Proceedings of the fourth annual workshop on computational learning theory, Santa Cruz, California* (pp. 147–156). San Francisco: Morgan Kaufmann.

Littlestone, N., & Warmuth, M. (1994). The weighted majority algorithm. *Information and Computation, 108*, 212–261.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*, 386–408.

Vovk, V. (1990). Aggregating strategies. In *Proceedings of third annual workshop on computational learning theory, Rochester, New York* (pp. 371–386). San Francisco: Morgan Kaufmann.
Vovk, V. (1998). A game of prediction with expert advice. *Journal of Computer and System Sciences, 56*, 153–173.

Ontology Learning

Different approaches have been used for building ontologies, most of them to date mainly using manual methods (►Text Mining for the Semantic Web). An approach to building ontologies was set up in the CYC project, where the main step involved manual extraction of common sense knowledge from different sources. Ontology construction methodologies usually involve several phases including *identifying the purpose of the ontology* (why to build it, how will it be used, the range of the users), *building the ontology*, *evaluation and documentation*. Ontology learning relates to the phase of building the ontology using semiautomatic methods based on text mining or machine learning.

Opinion Mining

Opinion mining is the application of mining methods concerned not with the topic a document is about, but with the opinion it expresses. Opinion mining builds on techniques from ►text mining, ►information retrieval, and computational linguistics. It is especially popular for analyzing documents that are often or even by definition opinionated, including blogs (►text mining for news and blogs analysis) and online product reviews.

Opinion mining is also known as *sentiment analysis* (*sentiment mining*, *sentiment classification*, ...) or *opinion extraction*.

Optimal Learning

►Bayesian Reinforcement Learning

OPUS

►Rule Learning

Ordered Rule Set

►Decision List

Ordinal Attribute

An *ordinal attribute* classifies data into categories that can be ranked. However, the differences between the ranks cannot be calculated by arithmetic. See ►Attribute and ►Measurement Scales.

Out-of-Sample Data

Out-of-sample data are data that were not used to learn a ►model. ►Holdout evaluation creates out-of-sample data for evaluation purposes.

Out-of-Sample Evaluation

Definition

Out-of-sample evaluation refers to ►algorithm evaluation whereby the learned model is evaluated on ►out-of-sample data. Out-of-sample evaluation provides an unbiased estimate of learning performance, in contrast to ►in-sample evaluation.

Cross References

►Algorithm Evaluation

Overall and Class-Sensitive Frequencies

The underlying idea for learning strategies processing ►missing attribute values relies on the class distribution; i.e., the class-sensitive frequencies are utilized. As soon as we substitute a missing value by a suitable one, we take the desired class of the example into consideration in order not to increase the noise in the data set. On the other hand, the overall (class-independent) frequencies are applied within classification.

Overfitting

Geoffrey I. Webb
Monash University, Victoria, Australia

Synonyms

Overtraining

Definition

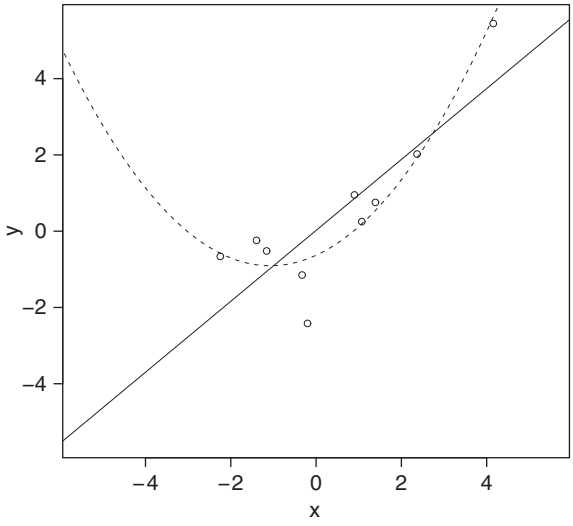
A ▶model *overfits* the ▶training data when it describes features that arise from noise or variance in the data, rather than the underlying distribution from which the data were drawn. Overfitting usually leads to loss of ▶accuracy on ▶out-of-sample data.

Discussion

In general there is a trade-off between the size of the space of distinct models that a ▶learner can produce and the risk of overfitting. As the space of models between which the learner can select increases, the risk of overfitting will increase. However, the potential for finding a model that closely fits the true underling distribution will also increase. This can be viewed as one facet of the ▶bias and variance trade-off.

Figure 1 illustrates overfitting. The points are drawn randomly from a distribution in which $y = x + \varepsilon$, where ε is random noise. The best single line fit to this distribution is $y = x$. ▶Linear regression finds a model $y = 0.02044 + 0.92978 \times x$, shown as the solid line in Fig. 1. In contrast, second degree polynomial regression finds the model $-0.6311 + 0.5128 \times x + 0.2386 \times x^2$, shown as the dashed line. The space of second degree polynomial models is greater than that of linear models, and so the second degree polynomial more closely fits the example data, returning the lower ▶squared error. However, the linear model more closely fits the true distribution and is more likely to obtain lower squared error on future samples.

While this example relates to ▶regression, the same effect also applies to ▶classification problems. For example, an overfitted ▶decision tree may include splits that reflect noise rather than underlying regularities in the data.



Overfitting. Figure 1. Linear and polynomial models fitted to random data drawn from a distribution for which the linear model is a better fit

The many approaches to avoiding overfitting include

- Using low ▶variance learners;
- ▶Minimum Description Length and ▶Minimum Message Length techniques
- ▶Pruning
- ▶Regularization
- ▶Stopping criteria

Cross References

▶Bias and Variance
▶Minimum Description Length
▶Minimum Message Length
▶Pruning
▶Regularization

Overtraining

▶Overfitting

P

PAC Identification

►PAC Learning

PAC Learning

Thomas Zeugmann
Hokkaido University
Sapparo, Japan

Synonyms

Distribution-free learning; Probably approximately correct learning; PAC identification

Motivation and Background

A very important learning problem is the task of *learning a concept*. ►*Concept learning* has attracted much attention in learning theory. For having a running example, we look at humans who are able to distinguish between different "things," e.g., chair, table, car, airplane, etc. There is no doubt that humans have to learn how to distinguish "things." Thus, in this example, each concept is a thing. To model this learning task, we have to convert "real things" into *mathematical descriptions of things*. One possibility to do this is to fix some language to express a *finite* list of properties. Afterward, we decide which of these properties are relevant for the particular things we want to deal with and which of them have to be fulfilled or not to be fulfilled, respectively. The list of properties comprises qualities or traits such as "has four legs," "has wings," "is green," "has a backrest," "has a seat," etc. So, these properties can be regarded as Boolean predicates and, provided the list of properties is large enough, each thing can be described by a conjunction of these predicates. For example, a chair is described as "has four legs and has a backrest and has a seat and has no wings." Note that the color is not relevant and thus, "is green" has been omitted.

Assume that we have n properties, where n is a natural number. In the easiest case, we can denote the n properties by Boolean variables $x_1, \ldots, x_n$, where $range(x_j) \subseteq \{0,1\}$ for $j = 1, \ldots, n$. The semantics is then obviously defined as follows: Setting $x_j = 1$ means property j is fulfilled, while $x_j = 0$ refers property j is not fulfilled. Now, setting $\mathcal{L}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2 \ldots, x_n, \bar{x}_n\}$ (set of literals), we can express each thing as a conjunction of literals. As usual, we refer to any conjunction of literals as a *monomial*.

Therefore, formally we have as *learning domain* (also called ►*instance space*), the set of all Boolean vectors of length n, i.e., $\{0,1\}^n$ and, in the learner's world, each thing (concept) is just a particular subset of $\{0,1\}^n$. As far as our example is concerned, the concept chair is then the set of all Boolean vectors for which the monomial "has four legs and has a backrest and has a seat and has no wings" evaluates to 1.

Furthermore, it is usually assumed that the concept c to be learned (the target concept) is taken from a prespecified class $\mathcal{C}$ of possible concepts called the *concept class*. In our example above, the concept class is the set of all concepts describable by a monomial. Consequently, we see that formally learning a concept is equivalent to identifying (exact or approximately) a set from a given set of possibilities by *learning* a suitable description (synonymously called representation) of it.

As in complexity theory, we usually assume that the representations are reasonable ones. Then they can be considered as strings over some fixed alphabet, and the set of representations constitutes the ►representation language. Note that a concept may have more than one representation in a given representation language (and should have at least one), and that there may be different representation languages for one and the same concept class. For example, every Boolean function can be expressed as a ►conjunctive normal form (CNF) and ►disjunctive normal form (DNF), respectively. For a

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

fixed representation language, the *size* of a concept is defined to be the length of a shortest representation for it. Since we are interested in a model of *efficient* learning, usually the following additional requirements are made: Given any string over the underlying alphabet, one can decide in time polynomial in the length of the string whether or not it is a representation. Furthermore, given any element x from the underlying learning domain and a representation r for any concept, one can uniformly decide in time polynomial in the length of both inputs whether or not x belongs to the concept c described by r.

So, we always have a representation language used to define the concept class. As we shall see below, it may be advantageous to choose a possibly different representation language used by the learner. The class of all sets described by this representation language is called *hypothesis space* (denoted by $\mathcal{H}$) and the elements of it are said to be hypotheses (commonly denoted by h).

The *learner* is specified to be an algorithm. Further details are given below. We still have to specify the information source, the criterion of success, the hypothesis space, and the prior knowledge in order to define what PAC learning is.

The abbreviation PAC stands for *probably approximately correct* and the corresponding learning model has been introduced by Valiant (1984), while its name was dubbed by Angluin (1988). Valiant's (1984) pioneering paper triggered a huge amount of research, the results of which are commonly called Computational Learning Theory (COLT) (see also the COLT and ALT conference series). Comprehensive treatises of this topic include Anthony and Biggs (1992), Kearns and Vazirani (1994) as well as Natarajan (1991).

Informally, this means that the learner has to find, on input a randomly drawn set of labeled examples (called sample), with high probability, a hypothesis such that the error of it is small. Here, the error is measured with respect to the same probability distribution D with respect to which the examples are drawn.

Let $X \neq \varnothing$ be any learning domain and let $\mathcal{C} \subseteq \wp(X)$ be any nonempty concept class (here $\wp(X)$ denotes the power set of X). If X is infinite we need some mild measure theoretic assumptions to ensure that the probabilities defined below exist. We refer to such concept classes as *well-behaved* concept classes. In particular, each $c \in \mathcal{C}$ has to be a Borel set. For a more detailed discussion see Blumer, Ehrenfeucht, Haussler, & Warmuth (1989).

Next, we formally define the *information source*. We assume any unknown probability distribution D over the learning domain X. No assumption is made concerning the nature of D and the learner has no knowledge concerning D. There is a *sampling oracle* $EX(\,)$, which has no input. Whenever $EX(\,)$ is called, it draws an element $x \in X$ according to D and returns the element x together with an indication of whether or not x belongs to the target concept c. Thus, every example returned by $EX(\,)$ may be written as $(x, c(x))$, where $c(x) = 1$ if $x \in c$ (positive examples) and $c(x) = 0$ otherwise (negative examples). If we make s calls to the example $EX(\,)$ then the elements $x_1, \ldots x_s$ are drawn independently from one another. Thus, the resulting probability distribution over all s-tuples of elements from X is the s-fold product distribution of D, i.e.,

$$\Pr(x_1, \ldots, x_s) = \prod_{i=1}^{s} D(x_i), \tag{1}$$

where $\Pr(A)$ denotes the probability of event A. Hence, the information source for a target concept c is any randomly drawn s-sample $S(c, \bar{x}) = (x_1, c(x_1), \ldots, x_s, c(x_s))$ returned by $EX(\,)$.

The *criterion of success*, i.e., *probably approximately correct* learning, is parameterized with respect to two quantities, the *accuracy parameter* ε and the *confidence parameter* δ, where $\varepsilon, \delta \in (0,1]$. Next, we define the *difference* between two sets $c,\ c' \subseteq X$ with respect to the probability distribution D as

$$d(c, c') = \sum_{x \in c \triangle c'} D(x),$$

where $c \triangle c'$ denotes the symmetric difference, i.e., $c \triangle c' = c \setminus c' \cup c' \setminus c$. We say that hypothesis h is an ε *approximation* of a concept c, if $d(c, h) \leq \varepsilon$. A learner is *successful*, if it computes an ε approximation of the target concept and it should do so with a probability at least $1 - \delta$.

The ▶*hypothesis space* $\mathcal{H}$ is any set such that $\mathcal{C} \subseteq \mathcal{H}$, and the only *prior knowledge* is that the target concept is from the concept class.

A further important feature of the PAC learning model is the demand to learn efficiently. Usually, in the

PAC learning model, the efficiency is measured with respect to the number of examples needed and the amount of computing time needed, and in both cases, the requirement is to learn with an amount that is polynomial in the "size of the problem." In order to arrive at a meaningful definition, one has to discuss the problem size and in addition, look at the asymptotic difficulty of the learning problem. That is, instead of studying the complexity of some fixed learning problems, we always look at infinite sequences of similar learning problems. Such infinite sequences are obtained by allowing the size (dimension) of the learning domain to grow or by allowing the complexity of the concepts considered to grow. In both cases, we use n to denote the relevant parameter.

Definition

A learning method $\mathcal{A}$ is said to probably approximately correctly learn a target concept c with respect to a hypothesis space $\mathcal{H}$ *and with sample complexity* $s = s(\varepsilon, \delta)$ *(or* $s = s(\varepsilon, \delta, n)$*), if for any distribution D over X and for all* $\varepsilon, \delta \in (0,1)$*, it makes s calls to the oracle $EX(\ )$, and after having received the answers produced by $EX(\ )$ (with respect to the target c), it always stops and outputs a representation of a hypothesis $h \in \mathcal{H}$ such that*

$$\Pr(d(c,h) \leq \varepsilon) \geq 1 - \delta.$$

A learning method $\mathcal{A}$ is said to probably approximately correctly identify a target concept class $\mathcal{C}$ with respect to a hypothesis space $\mathcal{H}$ *and with sample complexity* $s = s(\varepsilon, \delta)$*, if it probably approximately correctly identifies every concept $c \in \mathcal{C}$ with respect to $\mathcal{H}$ and with sample complexity s.*

A learning method $\mathcal{A}$ is said to be efficient*, if there exists a polynomial pol such that the running time of $\mathcal{A}$ and the number s of examples seen is at most* $pol(1/\varepsilon, 1/\delta, n)$.

Remarks

This looks complicated, and so, some explanation is in order. First, the inequality

$$\Pr(d(c,h) \leq \varepsilon) \geq 1 - \delta$$

says that with high probability (quantified by δ), there is not too much difference (quantified by ε) between the conjectured concept (described by h) and the target c. Formally, let $\mathcal{A}$ be any fixed learning method, and let c be any fixed target concept. For any fixed $\varepsilon, \delta \in (0,1]$, let $s = s(\varepsilon, \delta)$ be the actual sample size. We have to consider all possible outcomes of $\mathcal{A}$ when run on every labeled s-sample $S(c, \bar{x}) = (x_1, c(x_1), \ldots, x_s, c(x_s))$ returned by $EX(\ )$. Let $h(S(c, \bar{x}))$ be the hypothesis produced by $\mathcal{A}$ when processing $S(c, \bar{x})$. Then, we have to consider the set W of all s-tuples over X such that $d(c, h(S(c, \bar{x}))) \leq \varepsilon$. The condition $\Pr(d(c,h) \leq \varepsilon) \geq 1 - \delta$ can now be formally rewritten as $\Pr(W) \geq 1 - \delta$. Clearly, one has to require that $\Pr(W)$ is well defined. Note that the sample size is *not* allowed to depend on the distribution D.

To exemplify this approach, recall that our set of all concepts describable by a monomial over $\mathcal{L}_n$ refers to the set of all things. We consider a hypothetical learner (e.g., a student, a robot) that has to learn the concept of a chair. Imagine that the learner is told by a teacher whether or not particular things visible by the learner are instances of a chair. What things are visible depends on the environment the learner is in. The formal description of this dependence is provided by the unknown distribution D. For example, the learner might be led to a kitchen, a sitting room, a book shop, a beach, etc. Clearly, it would be unfair to teach the concept of a chair in a book shop and then testing the learning success at a beach. Thus, the learning success is measured with respect to the same distribution D with respect to which the sampling oracle has drawn its examples. However, the learner is required to learn with respect to any distribution. That is, independently of whether the learner is led to a kitchen, a book shop, a sitting room, a beach, etc., it has to learn with respect to the place it has been led to. The sample complexity refers to the amount of information needed to ensure successful learning. Clearly, the smaller the required distance of the hypothesis produced and the higher the confidence desired, the more examples are usually needed. But there might be atypical situations. To have an extreme example, the kitchen the learner is led to turned out to be empty. Since the learner is required to learn with respect to a typical kitchen (described by the distribution D), it may well fail under this particular circumstance. Such failure has to be restricted to atypical situations, and this is expressed

by demanding the learner to be successful with confidence $1 - \delta$.

This corresponds to real-life situations. For example, a student who has attended a course in learning theory might well suppose that she is examined in learning theory and not in graph theory. However, a good student, say in computer science, has to pass all examinations successfully, independently of the particular course attended. That is, she must successfully pass examinations in computability theory, complexity theory, cryptology, parallel algorithms, etc. Hence, she has to learn a whole concept class. The sample complexity refers to the time of interaction performed by the student and teacher. Also, the student may come up with a different representation of the concepts taught than the teacher. If we require $\mathcal{C} = \mathcal{H}$, then the resulting model is referred to as *proper* PAC learning.

The Finite Case

Having reached this point, it is natural to ask which concept classes are (efficiently) PAC learnable. We start with the finite case, i.e., learning domains X of finite cardinality. As before, the s-sample of c generated by $\bar{x}$ is denoted by $S(c, \bar{x}) = (x_1, c(x_1), \ldots, x_m, c(x_s))$. A hypothesis $h \in \mathcal{H}$ is called *consistent* for an s-sample $S(c, \bar{x})$, if $h(x_i) = c(x_i)$ for all $1 \leq i \leq s$. A learner is said to be *consistent* if all its outputs are consistent hypotheses. Then the following strategy may be used to design a PAC learner:

1. Draw a sufficiently large sample from the oracle $EX(\ )$, say s examples.
2. Find some $h \in \mathcal{H}$ that is consistent with all the s examples drawn.
3. Output h.

This strategy has a couple of remarkable features. First, provided the learner can find a consistent hypothesis, it allows for a uniform bound of the number of examples needed. That is,

$$s \geq \frac{1}{\varepsilon}\left(\ln|\mathcal{H}| + \ln\left(\frac{1}{\delta}\right)\right) \tag{2}$$

examples will always suffice (here $|S|$ denotes the cardinality of any set S).

The first insight obtained here is that increasing the confidence is exponentially cheaper than reducing the error.

Second, we see why we have to look at the asymptotic difficulty of the learning problem. If we fix $\{0,1\}^n$ as learning domain and define $\mathcal{C}$ to be the set of all concepts describable by a Boolean function, then there are 2^{2^n} many concepts over $\{0,1\}^n$. Consequently, $\ln|\mathcal{H}| = O(2^n)$ resulting in a sample complexity that is for sure infeasible if $n \geq 50$. Thus, we set $X_n = \{0,1\}^n$, consider $\mathcal{C}_n \subseteq \wp(X_n)$, and study the relevant learning problem for $(X_n, \mathcal{C}_n)_{n \geq 1}$. So, finite means that all X_n are finite.

Third, using inequality (2), it is not hard to see that the set of all concepts over $\{0,1\}^n$ that are describable by a monomial is efficiently PAC learnable. Let $\mathcal{H}_n$ be the set of all monomials containing each literal from $\mathcal{L}_n$ at most once plus the conjunction of all literals (denoted by m_{all}) (representing the empty concept). Since there are $3^n + 1$ monomials in $\mathcal{H}_n$, by (2), we see that $O(1/\varepsilon \cdot (n + \ln(1/\delta)))$ many examples suffice. Note that $2n$ is also an upper bound for the size of any concept from $\mathcal{H}_n$.

Thus it remains to deal with the problem to find a consistent hypothesis. The learning algorithm can be informally described as follows. After having received the s examples, the learner disregards all negative examples received and uses the positive ones to delete all literals from m_{all} that evaluate to 0 on at least one positive example. It then returns the conjunction of the literals not deleted from m_{all}. After a bit of reflection, one verifies that this hypothesis is consistent. This is essentially Haussler's (1987) Wholist algorithm and its running time is $O(1/\varepsilon \cdot (n^2 + \ln(1/\delta)))$. Also note that the particular choice of the representation for the empty concept was crucial here. It is worth noticing that the sample complexity is tight up to constant factors.

Using similar ideas, one can easily show that the class of all concepts over $\{0,1\}^n$ describable by a k-CNF or k-DNF (where k is fixed) is efficiently PAC learnable by using all k-CNF and k-DNF, respectively as hypothesis space (cf. Valiant, 1984).

So, what can we say in general concerning the problem to find a consistent hypothesis? Answering this question gives us the insight to understand why it is sometimes necessary to choose a hypothesis space that is different from the target concept class. This phenomenon was discovered by Pitt and Valiant (1988).

First, we look at the case where we have to efficiently PAC learn any $\mathcal{C}_n$ with respect to $\mathcal{C}_n$. Furthermore, an algorithm is said to *solve the consistency problem for* $\mathcal{C}_n$ if, on input any s-sample $S(c, \bar{x})$, where $c \subseteq X_n$, it outputs a hypothesis consistent with $S(c, \bar{x})$ provided there is one, and "there is no consistent hypothesis," otherwise.

Since we are interested in efficient PAC learning, we have to make the assumption that $|\mathcal{C}_n| \leq 2^{pol(n)}$ (cf. inequality (2)). Also, it should be noted that for the proof of the following result, the requirement that $h(x)$ is polynomial time computable is essential (cf. our discussion of representations). Furthermore, we need the notion of an $\mathcal{RP}$-algorithm (randomized polynomial time). The input is any s-sample $S(c, \bar{x})$, where $c \subseteq X_n$ and the running time is uniformly bounded by a polynomial in the length of the input. In addition to its input, the algorithm can flip a coin in every step of its computation and then branch in dependence of the outcome of the coin-flip. If there is no hypothesis consistent with $S(c, \bar{x})$, the algorithm must output "there is no consistent hypothesis," independently of the sequence of coin-flips made. If there is a hypothesis consistent with $S(c, \bar{x})$, then the $\mathcal{RP}$-algorithm is allowed to fail with a probability at most δ.

Interestingly, under the assumptions made above, one can prove the following equivalence for efficient PAC learning.

PAC learning $\mathcal{C}_n$ with respect to $\mathcal{C}_n$ is equivalent to solving the consistency problem for $\mathcal{C}_n$ by an $\mathcal{RP}$-algorithm.

We continue by looking at the class of all concepts describable by a k-term DNF$_n$. A term is conjunction of literals from $\mathcal{L}_n$, and a k-term DNF$_n$ is a disjunction of at most k terms. Consequently, there are $(3^n + 1)^k$ many k-term DNFs and thus the condition $|\mathcal{C}_n| \leq 2^{pol(n)}$ is fulfilled. Then one can show the following: *For all integers $k \geq 2$, if there is an algorithm that efficiently learns k-term DNF$_n$ with respect to k-term DNF$_n$, then $\mathcal{RP} = \mathcal{NP}$.*

For a formal definition of the complexity classes $\mathcal{RP}$ and $\mathcal{NP}$ we refer the reader to Arora and Barak (2009). This result is proved by showing that deciding the consistency problem for k-term DNF$_n$ is $\mathcal{NP}$-complete for every $k \geq 2$. The difference between deciding and solving the consistency problem is that we only have to decide if there is a consistent hypothesis in k-term DNF$_n$. However, by the equivalence established above, we know that an efficient proper PAC learner for k-term DNF$_n$ can be transformed into an $\mathcal{RP}$-algorithm even solving the consistency problem. It should be noted that we currently do not know whether or not $\mathcal{RP} = \mathcal{NP}$ (only $\mathcal{RP} \subseteq \mathcal{NP}$ has been shown) but it is widely believed that $\mathcal{RP} \neq \mathcal{NP}$. On the other hand, it easy to see that every concept describable by a k-term DNF$_n$ is also describable by a k-CNF$_n$ (but not conversely). Thus, we can finally conclude that there is an algorithm that efficiently PAC learns k-term DNF$_n$ with respect to k-CNF$_n$.

For more results along this line of research, we refer the reader to Pitt and Valiant (1988). As long as we do not have more powerful lower bound techniques allowing one to separate the relevant complexity classes $\mathcal{RP}$ and $\mathcal{NP}$ or $\mathcal{P}$ and $\mathcal{NP}$, no unconditional negative result concerning PAC learning can be shown. Another approach to show hardness results for PAC learning is based on cryptographic assumptions, and recently, one has also tried to base cryptographic assumptions on the hardness of PAC learning (cf., e.g., Xiao 2009 and the references therein).

Further, positive results comprise the efficient proper PAC learnability of k-decision lists for any fixed k.

Finally, it must be noted that the bounds on the sample size obtained via inequality (2) are *not* the best possible. Sometimes, better bounds can be obtained by using the VC Dimension (see inequality (4)).

The Infinite Case

Let us start our exposition concerning infinite concept classes with an example due to Blumer, Ehrenfeucht, Haussler, & Warmuth (1989). Consider the problem of learning concepts such as "medium built" animals. For the sake of presentation, we restrict ourselves to the parameters "weight" and "length." To describe "medium built" we use intervals "from–to." For example, a medium built cat might have a weight ranging from 3 to 7 kg and a length ranging from 25 to 50 cm. By looking at a finite database of randomly chosen animals, giving their respective weight and length and their *classification* (medium built or not), we want to form a rule that *approximates* the true concept of "medium built" for each animal under consideration.

This learning problem can be formalized as follows. Let $X = \mathbb{E}^2$ be the two-dimensional Euclidean space, and let $\mathcal{C} \subseteq \wp(\mathbb{E}^2)$ be the set of all axis-parallel rectangles, i.e., products of intervals on the x-axis with intervals on the y-axis. Furthermore, let D be any probability distribution over X. Next, we show that $\mathcal{C}$ is efficiently PAC learnable with respect to $\mathcal{C}$ by the following Algorithm *LR*

Algorithm LR: On input any $\varepsilon, \delta \in (0,1]$, call the oracle $EX(\ )$ s times, where $s = 4/\varepsilon \cdot \ln(4/\delta)$. Let $(r_1, c(r_1), r_2, c(r_2), \ldots, r_s, c(r_s))$ be the s-sample returned by $EX(\ )$, where $r_i = (x_i, y_i)$, $i = 1, \ldots s$.
Compute $x_{\min} = \min\{x_i \mid 1 \le i \le s,\ c(r_i) = 1\}$

$$x_{\max} = \max\{x_i \mid 1 \le i \le s,\ c(r_i) = 1\}$$

$$y_{\min} = \min\{y_i \mid 1 \le i \le s,\ c(r_i) = 1\}$$

$$y_{\max} = \max\{y_i \mid 1 \le i \le s,\ c(r_i) = 1\}$$

Output $h = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. In case there is no positive example, return $h = \varnothing$.
end.

It remains to show that Algorithm LR PAC learns the concept class $\mathcal{C}$ with respect to $\mathcal{C}$. Let $c = [a,b] \times [c,d]$ be the target concept. Since LR computes its hypothesis from positive examples, only, we get $h \subseteq c$. That is, h is consistent. We have to show that $d(c,h) \le \varepsilon$ with probability at least $1 - \delta$. We distinguish the following cases.

Case 1. $D(c) \le \varepsilon$
Then $d(c,h) = \sum_{r \in c \triangle h} D(r) = \sum_{r \in c \setminus h} D(r) \le D(c) \le \varepsilon$.
Hence, in this case we are done.

Case 2. $D(c) > \varepsilon$
We define four minimal side rectangles within c that each cover an area of probability of at least $\varepsilon/4$. Let

Left $= [a,x] \times [c,d]$, where $x = \inf\{\tilde{x} \mid D([a,\tilde{x}] \times [c,d]) \ge \varepsilon/4\}$

Right $= [z,b] \times [c,d]$, where $z = \inf\{\tilde{x} \mid D([\tilde{x},b] \times [c,d]) \ge \varepsilon/4\}$

Top $= [a,b] \times [y,d]$, where $y = \inf\{\tilde{x} \mid D([a,b] \times [\tilde{x},d]) \ge \varepsilon/4\}$

Bottom $= [a,b] \times [c,t]$, where $t = \inf\{\tilde{x} \mid D([a,b] \times [c,\tilde{x}]) \ge \varepsilon/4\}$

All those rectangles are contained in c, since $D(c) > \varepsilon$. If the sample size is s, the probability that a *particular* rectangle from {*Left, Right, Top, Bottom*} contains no positive example is at most $(1-\varepsilon/4)^s$. Thus, the probability that *some* of those rectangles does not contain any positive example is at most $4(1-\varepsilon/4)^s$. Hence, incorporating $s = 4/\varepsilon \cdot \ln(4/\delta)$ gives:

$$4(1-\varepsilon/4)^s < 4e^{-(\varepsilon/4)s} = 4e^{-\ln(4/\delta)} = \delta.$$

Therefore, with probability at least $1-\delta$, *each* of the four rectangles *Left, Right, Top, Bottom* contains a positive example. Consequently, we get:

$$d(c,h) = \sum_{r \in c \triangle h} D(r) = \sum_{r \in c \setminus h} D(r) = D(c) - D(h).$$

Furthermore, by construction

$$D(h) \ge D(c) - D(Left) - D(Right) - D(Top) - D(Bottom) \ge D(c) - \varepsilon$$

and hence $d(c,h) \le \varepsilon$.

Having reached this point, it is only natural to ask what makes infinite concept classes PAC learnable. Interestingly, there is a single parameter telling us whether or not a concept class is PAC learnable. This is the so-called Vapnik–Chervonenkis dimension commonly abbreviated as ▶VC Dimension. In our example of axis-parallel rectangles, the VC Dimension of $\mathcal{C}$ is 4.

In order to state this result, we have to exclude trivial concept classes. A concept class $\mathcal{C}$ is said to be *trivial* if $|\mathcal{C}| = 1$ or $\mathcal{C} = \{c_1, c_2\}$ with $c_1 \cap c_2 = \varnothing$ and $X = c_1 \cup c_2$. $\mathcal{C}$ is called nontrivial iff $\mathcal{C}$ is not trivial. Then Blumer, Ehrenfeucht, Haussler, & Warmuth (1989) showed the following:

A nontrivial well-behaved concept class is PAC learnable if and only if its VC dimension is finite.

Moreover, if the VC dimension is finite, essentially the same strategy as in the finite case applies, i.e., it suffices to construct a consistent hypothesis from $\mathcal{C}$ (or from a suitably chosen hypothesis space $\mathcal{H}$ which must be well behaved) in random polynomial time.

So, it remains to estimate the sample complexity. Let d be the VC dimension of $\mathcal{H}$. Blumer, Ehrenfeucht, Haussler, & Warmuth (1989) showed that

$$s \geq \max\left\{\frac{4}{\varepsilon}\log\frac{2}{\delta}, \frac{8d}{\varepsilon}\log\frac{13}{\varepsilon}\right\} \quad (3)$$

examples do suffice. This upper bound has been improved by Anthony et al. (1990) to

$$s \geq \frac{1}{\varepsilon(1-\sqrt{\varepsilon})}\left[\log\left(\frac{d/(d-1)}{\delta}\right) + 2d\log\left(\frac{6}{\varepsilon}\right)\right]. \quad (4)$$

Based on the work of Blumer et al. (1989) (and the lower bound they gave), Ehrenfeucht, Haussler, Kearns, & Valiant (1988) showed that if $\mathcal{C}$ is nontrivial, then no learning function exists (for any $\mathcal{H}$) if $s < \frac{1-\varepsilon}{2\varepsilon}\log\frac{2}{\delta} + \frac{d-1}{64\varepsilon}$. These results give a precise characterization of the number of examples needed (apart from the gap of a factor of $O\left(\log\frac{1}{\varepsilon}\right)$) in terms of the VC dimension. Also note the sharp dichotomy here either any consistent learner (computable or not) will do or no learner at all exists.

Two more remarks are in order here. First, these bounds apply to *uniform* PAC learning, i.e., the learner is taking ε and δ as input, only. As outlined in our discussion just before we gave the formal definition of PAC learning, it is meaningful to look at the asymptotic difficulty of learning. In the infinite case, we can increment the dimension n of the learning domain as we did in the finite case. We may set $X_n = \mathbb{E}^n$ and then consider similar concept classes $\mathcal{C}_n \subseteq \wp(X_n)$. For example, the concept classes similar to axis-parallel rectangles are axis-parallel parallelepipeds in $\mathbb{E}^n$. Then the VC dimension of $\mathcal{C}_n$ is $2n$ and all what is left is to add n as input to the learner and to express d as a function of n in the bound (4). Clearly, the algorithm *LR* can be straightforwardly generalized to a learner for $(X_n, \mathcal{C}_n)_{n\geq 1}$.

Alternatively, we use n to parameterize the complexity of the concepts to be learned. As an example, consider $X = \mathbb{E}$ and let $\mathcal{C}_n$ be the set of all unions of at most n (closed or open) intervals. Then the ▶VC Dimension of $\mathcal{C}_n$ is $2n$, and one can design an efficient learner for $(X, \mathcal{C}_n)_{n\geq 1}$. Another example is obtained for $X = \mathbb{E}^2$ by defining $\mathcal{C}_n$ to be the class of all convex polygons having at most n edges (cf. Linial, Mansour, & Rivest, 1991).

Second, all the results discussed so far are dealing with *static sampling*, i.e., any sample containing the necessary examples is drawn before any computation is performed. So, it is only natural to ask what can be achieved when *dynamic sampling* is allowed. In dynamic sampling mode, a learner alternates between drawing examples and performing computations. Under this sampling mode, even concepts classes having an infinite VC dimension are learnable (cf. Linial, Mansour, & Rivest, 1991 and the references therein). The main results in this regard are that enumerable concepts classes and decomposable concept classes are PAC learnable when using dynamic sampling.

Let us finish the general exposition of PAC learning by pointing to another interesting insight, i.e., learning is in some sense data compression. As we have seen, finding consistent hypotheses is a problem of fundamental importance in the area of PAC learning. Clearly, the more expressive the representation language for the hypothesis space, the easier it may be to find a consistent hypothesis, but it may be increasingly difficult to say something concerning its accuracy (in machine learning this phenomenon is also known as the over-fitting problem). At this point, Occam's Razor comes into play. If there is more than one explanation for a phenomenon, then Occam's Razor requires to "prefer simple explanations." So, an Occam algorithm is an algorithm which, given a sample of the target concept, outputs a consistent and relatively simple hypothesis. That is, it is capable of some *data compression*. Let us first look at the Boolean case, i.e., $X_n = \{0,1\}^n$. Then an Occam algorithm is a randomized polynomial time algorithm $\mathcal{A}$ such that there is a polynomial p and a constant $\alpha \in [0,1)$ fulfilling the following demands.

For every $n \geq 1$, every target concept $c \in \mathcal{C}_n$ of size at most m and every $\varepsilon \in (0,1)$, on input any s-sample for c, algorithm $\mathcal{A}$ outputs with probability at least $1-\varepsilon$, the representation of a consistent hypothesis from $\mathcal{C}_n$ having size at most $p(n,m,1/\varepsilon)\cdot s^{\alpha}$.

So, the parameter $\alpha < 1$ expresses the amount of compression required. If we have such an Occam algorithm, then $(X_n, \mathcal{C}_n)$ is properly PAC learnable (cf. Blumer, Ehrenfeucht, Haussler, & Warmuth, 1987). The proof is based on the observations that a hypothesis with large error is unlikely to be consistent with a large sample, and that there are only few short hypotheses. If we replace in the definition of an Occam algorithm the demand on the existence of a short hypotheses by the existence of a hypothesis space having a small VC dimension, then a similar result can be obtained for the continuous case (cf. Blumer, Ehrenfeucht, Haussler, & Warmuth, 1989). To a certain extend, the converse is also true, i.e., under quite general conditions, PAC learnability implies the existence of an Occam algorithm. We refer the reader to Kearns and Vazirani (1994) for further details.

Variations

Further variations of PAC learning are possible and have been studied. So far, we have only considered one sampling oracle. So, a natural variation is to have two sampling oracles $EX_+(\ )$ and $EX_-(\ )$ and two distributions D_+ and D_-, i.e., one for positive examples and one for negative examples. Clearly, further natural variations are possible. A larger number of them has been shown to be roughly equivalent and we refer the reader to Haussler, Kearns, Littlestone, & Warmuth (1991) for details.

We continue with another natural variation that turned out to have a fundamental impact to the whole area of machine learning, i.e., weak learning.

Weak Learning

An interesting variation of PAC learning is obtained if we weaken the requirements concerning the confidence and the error. That is, instead of requiring the PAC learner to succeed for every ε and δ, one may relax this demand as follows. We only require the learner to succeed for $\varepsilon = 1/2 - 1/pol(n)$ (n is as above) and $\delta = 1/poly(n)$ (n is as above), where *pol* and *poly* are any two fixed polynomials. The resulting model is called *weak* PAC learning.

Quite surprisingly, Schapire (1990) could prove that every weak learner can be efficiently transformed into an ordinary PAC learner. While it is not too difficult to *boost* the confidence, *boosting* the error is much more complicated and has subsequently attracted a lot of attention. We refer the reader to Schapire (1990) as well as Kearns and Vazirani (1994) and the references therein for a detailed exposition. Interesting enough, the techniques developed to prove the equivalence of weak PAC learnability and PAC learnability have an enormous impact to machine learning and may be subsumed under the title ▶Boosting.

Relations to Other Learning Models

Finally, we point out some relations of PAC learning to other learning models. Let us start with the mistake bound model also called online prediction model. The mistake-bound model has its roots in ▶Inductive Inference and was introduced by Littlestone (1988). It is conceptionally much simpler than the PAC model, since it does not involve probabilities. For the sake of presentation, we assume a finite learning domain X_n and any $\mathcal{C}_n \subseteq \wp(X_n)$ here.

In this model the following scenario is repeated indefinitely. The learner receives an instance x and has to predict $c(x)$. Then it is given the true label $c(x)$. If the learner's prediction was incorrect, then a *mistake* occurred. The learner is successful if the total number of mistakes is finite. In order to make this learning problem non-trivial, one additionally requires a polynomial *pol* such that for every $c \in \mathcal{C}_n$ and any ordering of the examples, the total number of mistakes is bounded by $pol(n, size(c))$. In the mistake-bound model, a learner is said to be efficient if its running time per stage is uniformly polynomial in n and $size(c)$.

Then, the relation to PAC learning is as follows:

If algorithm $\mathcal{A}$ learns a concept class $\mathcal{C}$ in the mistake-bound model, then $\mathcal{A}$ also PAC learn $\mathcal{C}$. Moreover, if $\mathcal{A}$ makes at most M mistakes, then the resulting PAC learner needs $\frac{M}{\varepsilon} \cdot \ln \frac{M}{\delta}$ many examples.

So, efficient mistake-bound learning translates into efficient PAC learning.

Another interesting relation is obtained when looking at the ▶Query-Based Learning model, where the only queries allowed are equivalence queries. As pointed out by Angluin (1988, 1992), any learning method that uses equivalence queries only and achieves exact identification can be transformed into a PAC learner. The number of equivalence queries necessary to

achieve success in the query learning model is polynomially related to the number of calls made to the sample oracle.

However, the converse is not true. This insight led to the definition of a *minimally adequate teacher* (cf. Angluin, 1988 and the references therein). In this setting, the teacher answers equivalence queries and membership queries. Maas and Turan (1990) provide a detailed discussion of the relationship between the different models.

These results in turn led to another modification of the PAC model, where the learner is, in addition to the s-sample returned, also allowed to ask membership queries, i.e., PAC learning with membership queries.

Let us finish this article by mentioning that the PAC model has been criticized for two reasons. The first one is the independence assumption, i.e., the requirement to learn with respect to any distribution. This is, however, also a very strong part of the theory, since it provides universal performance guarantees. Clearly, if one has additional information concerning the underlying distributions, one may be able to prove better bounds. The second reason is the "noise-free" assumption, i.e., the requirement to the sample oracle to return exclusively correct labels. Clearly, in practice, we never have noise-free data. So, one has also studied learning in the presence of noise and we refer the reader to Kearns and Vazirani (1994) as well as to conference series COLT and ALT for results along this line.

Cross References

►Statistical Machine Learning
►Stochastic Finite Learning
►VC Dimension

Recommended Reading

Angluin, D. (1988). Queries and concept learning. *Machine Learning, 2*(4), 319–342.

Angluin, D. (1992). Computational learning theory: Survey and selected bibliography. In *Proceedings of the twenty-fourth annual ACM symposium on theory of computing* (pp. 351–369). New York: ACM Press.

Anthony, M., & Biggs, N. (1992). *Computational learning theory: Cambridge tracts in theoretical computer science* (No. 30). Cambridge: Cambridge University Press.

Anthony, M., Biggs, N., & Shawe-Taylor, J. (1990). The learnability of formal concepts. In M. A. Fulk & J. Case (Eds.), *Proceedings of the third annual workshop on computational learning theory* (pp. 246–257). San Mateo, CA: Morgan Kaufmann.

Arora, S., & Barak, B. (2009). *Computational complexity: A modern approach*. Cambridge: Cambridge University Press.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). Occam's razor. *Information Processing Letters, 24*(6), 377–380.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM, 36*(4), 929–965.

Ehrenfeucht, A., Haussler, D., Kearns, M., & Valiant, L. (1988). A general lower bound on the number of examples needed for learning. In D. Haussler & L. Pitt (Eds.), *COLT '88, Proceedings of the 1988 workshop on computational learning theory*, August 3–5, 1988, MIT (pp. 139–154). San Francisco: Morgan Kaufmann.

Haussler, D. (1987). Bias, version spaces and Valiant's learning framework. In P. Langley (Ed.), *Proceedings of the fourth international workshop on machine learning* (pp. 324–336). San Mateo, CA: Morgan Kaufmann.

Haussler, D., Kearns, M., Littlestone, N., & Warmuth, M. K. (1991). Equivalence of models for polynomial learnability. *Information and Computation, 95*(2), 129–161.

Kearns, M. J., & Vazirani, U. V. (1994). *An introduction to computational learning theory*. Cambridge, MA: MIT Press.

Linial, N., Mansour, Y., & Rivest, R. L. (1991). Results on learnability and the Vapnik–Chervonenkis dimension. *Information and Computation, 90*(1), 33–49.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning, 2*(4), 285–318.

Maas, W., & Turan, G. (1990). On the complexity of learning from counterexamples and membership queries. In *Proceedings of the thirty-first annual symposium on Foundations of Computer Science (FOCS 1990)*, St. Louis, Missouri, October 22–24, 1990 (pp. 203–210). Los Alamitos, CA: IEEE Computer Society.

Natarajan, B. K. (1991). *Machine learning: A theoretical approach*. San Mateo, CA: Morgan Kaufmann.

Pitt, L., & Valiant, L. G. (1988). Computational limitations on learning from examples. *Journal of the ACM, 35*(4), 965–984.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*(2), 197–227.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM, 27*(11), 1134–1142.

Xiao, D. (2009). On basing ZK ≠ BPP on the hardness of PAC learning. In *Proceedings of the twenty-fourth annual IEEE Conference on Computational Complexity, CCC 2009*, Paris, France, July 15–18, 2009 (pp. 304–315). Los Alamitos, CA: IEEE Computer Society.

PAC-MDP Learning

►Efficient Exploration in Reinforcement Learning

Parallel Corpus

A parallel corpus (pl. corpora) is a document collection composed of two or more disjoint subsets, each written in a different language, such that documents in each subset are translations of documents in each other subset. Moreover, it is required that the translation relation is known, i.e., that given a document in one of the subset (i.e., languages), it is known what documents in the other subset are its translations. The statistical analysis of parallel corpora is at the heart of most methods for ►cross-language text mining.

Part of Speech Tagging

►POS Tagging

Partially Observable Markov Decision Processes

Pascal Poupart
University of Waterloo

Synonyms

POMDPs; Belief state Markov decision processes; Dynamic decision networks; Dual control

Definition

A partially observable Markov decision process (POMDP) refers to a class of sequential decision-making problems under uncertainty. This class includes problems with partially observable states and uncertain action effects. A POMDP is formally defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, h, \gamma \rangle$ where $\mathcal{S}$ is the set of states s, $\mathcal{A}$ is the set of actions a, $\mathcal{O}$ is the set of observations o, $T(s, a, s') = \Pr(s'|s, a)$ is the transition function indicating the probability of reaching s' when executing a in s, $Z(a, s', o') = \Pr(o'|a, s')$ is the observation function indicating the probability of observing o' in state s' after executing a, $R(s, a) \in \Re$ is the reward function indicating the (immediate) expected utility of executing a in s, $b_0 = \Pr(s_0)$ is the distribution over the initial state (also known as initial belief), h is the planning horizon (which may be finite or infinite), and $\gamma \in [0, 1]$ is a discount factor indicating by how much rewards should be discounted at each time step. Given a POMDP, the goal is to find a policy to select actions that maximize rewards over the planning horizon.

Motivation and Background

Partially observable Markov decision processes (POMDPs) were first introduced in the Operations Research community (Drake, 1962; Aström, 1965) as a framework to model stochastic dynamical systems and to make optimal decisions. This framework was later considered by the artificial intelligence community as a principled approach to planning under uncertainty (Kaelbling et al., 1998). Compared to other methods, POMDPs have the advantage of a well-founded theory. They can be viewed as an extension of the well-known, fully observable ►Markov decision process (MDP) model (Puterman, 1994), which is rooted in probability theory, utility theory, and decision theory. POMDPs do not assume that states are fully observable, but instead that only part of the state features are observable, or more generally, that the observable features are simply correlated with the underlying states. This naturally captures the fact that in many real-world problems, the information available to the decision maker is often incomplete and typically measured by noisy sensors. As a result, the decision process is much more difficult to optimize. POMDP applications include robotics (Pineau & Gordon, 2005), assistive technologies (Hoey et al., 2010), health informatics (Hauskrecht & Fraser, 2010), spoken dialogue systems (Thomson & Young, 2010), and fault recovery (Shani & Meek, 2009).

Structure of Model and Solution Algorithms

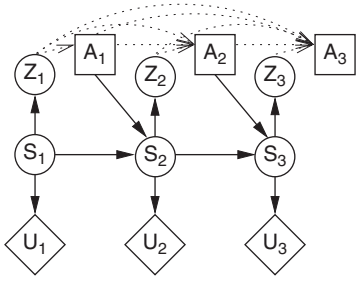
We describe below the POMDP model, some policy representations, the properties of optimal value functions, and some solution algorithms.

POMDP Model

Figure 1 shows the graphical representation of a POMDP, using the notation of influence diagrams: circles denote random variables (e.g., state variables S_t and observation variables O_t), squares denote decision variables (e.g., action variables A_t), and diamonds denote

utility variables (e.g., U_t's). The variables are indexed by time and grouped in time slices, reflecting the fact that each variable may take a different value at each time step. Arcs indicate how nodes influence each other over time. There are two types of arcs: probabilistic and informational arcs. Arcs pointing to a chance node or a utility node indicate a probabilistic dependency between a child and its parents, whereas arcs pointing to a decision node indicate the information available to the decision maker (i.e., which nodes are observable at the time of each decision). Probabilistic dependencies for the state and observation variables are quantified by the conditional distributions $\Pr(S_{t+1}|S_t, A_t)$ and $\Pr(O_{t+1}|S_{t+1}, A_t)$, which correspond to the transition and observation functions. Note that the initial state variable S_0 does not have any parent, hence its distribution $\Pr(S_0)$ is unconditioned and corresponds to the initial belief b_0 of the decision maker. Probabilistic dependencies for the utility variables are also quantified by a conditional distribution $\Pr(U_t|S_t, A_t)$ such that its expectation $\sum_u \Pr(u|S_t, A_t)u = R(S_t, A_t)$ corresponds to the reward function.

Fully observable MDPs are a special case of POMDPs since they arise when the observation function deterministically maps each state to a different unique observation. POMDPs can also be viewed as ▶hidden Markov models (HMMs) (Rabiner, 1989) extended with decision and utility nodes since the transition and observation distributions essentially define an HMM. POMDPs also correspond to a special case of decision networks called *dynamic decision networks* (Buede, 1999) where it is assumed that the transition, observation, and reward functions are *stationary* (i.e., they do not depend on time) and *Markovian* (i.e., the parents of each variable are in the same time slice or immediately preceding time slice).

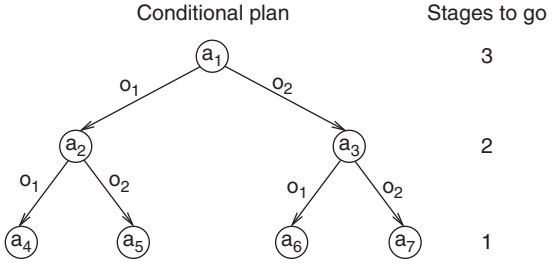


Partially Observable Markov Decision Processes. Figure 1. POMDP represented as an influence diagram

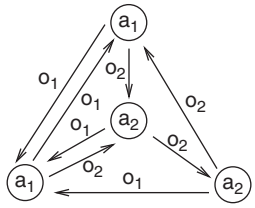
Policies

Given a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, h, \gamma \rangle$ specifying a POMDP, the goal is to find a policy π to select actions that maximize the rewards. The informational arcs indicate that each action a_t can be selected based on the history of past actions and observations. Hence, in its most general form, a policy $\pi : \langle b_0, h_t \rangle \rightarrow a_t$ is a mapping from initial beliefs b_0 and histories $h_t = \langle o_0, a_0, o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t \rangle$ to actions a_t. For a fixed initial belief, the mapping can be represented by a tree such as the one in Fig. 2. We will refer to such policy trees as conditional plans since in general a policy may consist of several conditional plans for different initial beliefs. The execution of a conditional plan follows a branch from the root to some leaf by executing the actions of the nodes traversed and following the edges labeled by the observations received.

Unfortunately, as the number of steps increases, the number of histories grows exponentially and it is infeasible to represent mappings over all such histories. Furthermore, infinite-horizon problems require mappings over arbitrarily long histories, which limit the use of trees to problems with a short horizon. Note, however, that it is possible to have mappings over infinite *cyclic* histories. Such mappings can be represented by a *finite state controller* (Hansen, 1997), which is essentially a cyclic graph of nodes labeled by actions and edges labeled by observations (see Fig. 3 for an example). Similar to conditional plans, finite state controllers are executed by starting at an initial node, executing the actions of the nodes traversed, and following the edges of the observations received.



Partially Observable Markov Decision Processes. Figure 2. Three representation of a three-step conditional plan



Partially Observable Markov Decision Processes. Figure 3. Finite state controller for a simple POMDP with two actions and two observations

Alternatively, it is possible to summarize histories by a sufficient statistic that encodes all the relevant information from previous actions and observations for planning purposes. Recall that the transition, reward, and observation functions exhibit the Markov property, which means that the outcome of future states, rewards, and observations depend only on the current state and action. If the decision maker knew the current state of the world, then she would have all the desired information to make an optimal action choice. Thus, histories of past actions and observations are only relevant to the extent that they provide information about the current state of the world. Let b_t be the belief of the decision maker about the state of the world at time step t, which we represent by a probability distribution over the state space $\mathcal{S}$. Using Bayes theorem (see ▶Bayes Rules), one can compute the current belief b_t from the previous belief b_{t-1}, previous action a_{t-1}, and current observation o_t:

$$b_t(s') = k \sum_{s \in \mathcal{S}} b_{t-1}(s) \Pr(s' | s, a_{t-1}) \Pr(o_t | a_{t-1}, s') \quad (1)$$

where k denotes a normalizing constant. Hence, a policy π can also be represented as a mapping from beliefs b_t to actions a_t. While this gets around the exponentially large number of histories, the space of beliefs is an $|\mathcal{S}| - 1$-dimensional continuous space, which is also problematic. However, a key result by Smallwood and Sondik (1973) allows us to circumvent the continuous nature of the belief space. But first, let us introduce value functions and then discuss Smallwood and Sondik's solution.

Value Functions Given a set of policies, we need a mechanism to evaluate and compare them. Roughly speaking, the goal is to maximize the amount of reward earned over time. This loosely defined criterion can be formalized in several ways: one may wish to maximize *total* (accumulated) or *average* reward, *expected* or *worst-case* reward, *discounted* or *undiscounted* reward. The rest of this article assumes an *expected total discounted* reward criterion, since it is by far the most popular in the literature. We define the value $V^\pi(b_0)$ of executing some policy π starting at belief b_0 to be the expected sum of the discounted rewards earned at each time step:

$$V^\pi(b_0) = \sum_{t=0}^{h} \gamma^t \sum_{s \in \mathcal{S}} b_t(s) R(s, \pi(b_t)) \quad (2)$$

where $\pi(b_t)$ denotes the action prescribed by policy π at belief b_t. A policy π^* is optimal when its value function V^* is at least as high as any other policy for all beliefs (i.e., $V^*(b) \geq V^\pi(b) \forall b$).

As with policies, representing a value function can be problematic because its domain is an $(|\mathcal{S}| - 1)$-dimensional continuous space corresponding to the belief space. However, Smallwood and Sondik (1973) showed that optimal value functions for finite-horizon POMDPs are piecewise-linear and convex. The value of executing a conditional plan from any state is constant. If we do not know the precise underlying state, but instead we have a belief corresponding to a distribution over states, then the value of the belief is simply a weighted average (according to b) of the values of the possible states. Thus, the value function $V^\beta(b)$ of a conditional plan β is linear with respect to b. This means that $V^\beta(b)$ can be represented by a vector α_β of size $|\mathcal{S}|$ such that $V^\beta(b) = \Sigma_s b(s) \alpha_\beta(s)$.

For a finite horizon h, an optimal policy π^h consists of the best conditional plans for each initial belief. More precisely, the best conditional plan β^* for some belief b is the one that yields the highest value: $\beta^* = \text{argmax}_\beta V^\beta(b)$. Although there are uncountably many beliefs, the set of h-step conditional plans is finite and therefore an h-step optimal value function can be represented by a finite collection Γ^h of α-vectors. For infinite horizon problems, the optimal value function may require an infinite number of α-vectors.

Figure 4 shows an optimal value function for a simple two-state POMDP. The horizontal axis represents the belief space and the vertical axis indicates the expected total reward. Assuming the two world states

are s and $\bar{s}$, then a belief is completely determined by the probability of s. Therefore, the horizontal axis represents a continuum of beliefs determined by the probability $b(s)$. Each line in the graph is an α-vector, which corresponds to the value function of a conditional plan. The upper surface of those α-vectors is a piecewise-linear and convex function corresponding to the optimal value function $V^*(b) = \max_{\alpha \in \Gamma^h} \alpha(b)$.

Note that an optimal policy can be recovered from the optimal value function represented by a set Γ of α-vector. Assuming that an action is stored with each α-vector (this would typically be the root action of the conditional plan associated with each α-vector), then the decision maker simply needs to look up the maximal α-vector for the current belief to retrieve the action. Hence, value functions represented by a set of α-vectors, each associated with an action, implicitly define a mapping from beliefs to actions.

Optimal value functions also satisfy *Bellman's equation*

$$V^{h+1}(b) = max_a R(b,a) + \gamma \sum_{o'} \Pr(o'|b,a) V^h(b^{ao'}) \quad (3)$$

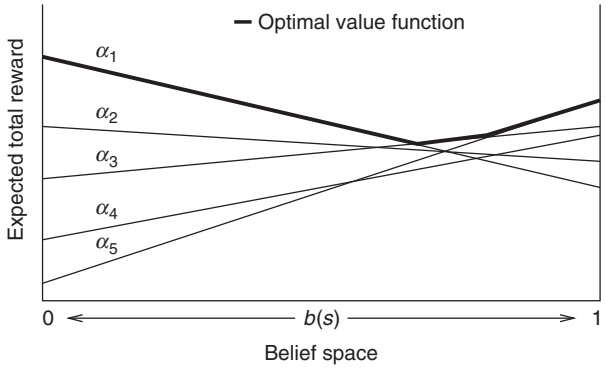
where $R(b,a) = \Sigma_s b(s) R(s,a)$, $\Pr(o'|b,a) = \Sigma_{s,s'} b(s) \Pr(s'|s,a) \Pr(o'|s',a)$, and $b^{ao'}$ is the updated belief after executing a and observing b according to Bayes theorem (Eq. 1). Intuitively, this equation says that the optimal value for $h+1$ steps to go consists of the highest sum of the current reward with the future rewards for the remaining h steps. Since we do not know exactly what rewards will be earned in the future, an expectation (with respect to the observations) is used to estimate future rewards. For discounted infinite horizon problems, the optimal value function V^* is a fixed point of Bellman's equation:

$$V^*(b) = max_a R(b,a) + \gamma \sum_{o'} \Pr(o'|b,a) V^*(b^{ao'})$$

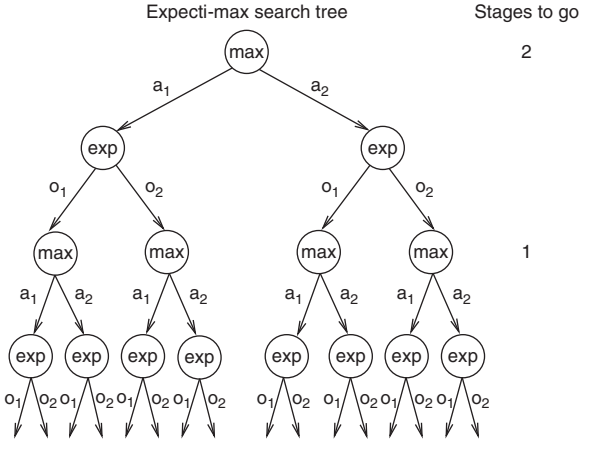
Solution Algorithms

There are two general classes of solution algorithms to optimize a policy. The first class consists of *online* algorithms that plan while executing the policy by growing a search tree. The second class consists of *offline* algorithms that precompute a policy which can be executed with minimal online computation. In practice, it is best to combine online and offline techniques since we may as well obtain the best policy possible in an offline phase and then refine it with an online search at execution time.

Forward Search Online search techniques generally optimize a conditional plan for the current belief by performing a forward search from that belief. They essentially build an *expecti-max* search tree such that expectations over observations and maximizations over actions are performed in alternation. Figure 5 illustrates such a tree for a two-step horizon (i.e., two alternations of actions and observations). An optimal policy is obtained by computing the beliefs associated with each node in a forward pass, followed by a backward pass that computes the optimal value at each node. A recursive form of this approach is described in Algorithm 1.



Partially Observable Markov Decision Processes. Figure 4. Geometric view of value function



Partially Observable Markov Decision Processes. Figure 5. Two-step expecti-max search tree

Algorithm 1 Forward Search

Inputs: Belief b and horizon h
Outputs: Optimal value V^*
if $h = 0$ **then**
  $V^* \leftarrow 0$
else
  for all a, o **do**
    $b^{ao'}(s') \leftarrow k \sum_s b(s) \Pr(s'|s,a) \Pr(o'|s',a') \forall s'$
    $V^{ao'} \leftarrow$ *forward Search*$(b^{ao'}, h-1)$
  end for
  $V^* \leftarrow \max_a R(b,a) + \gamma \sum_{o'} \Pr(o'|b,a) V^{ao'}$
end if

Beliefs are propagated forward according to Bayes theorem, while rewards are accumulated backward according to Bellman's equation.

Since the expecti-max search tree grows exponentially with the planning horizon h, in practice, the computation can often be simplified by pruning suboptimal actions by branch and bound and sampling a small set of observations instead of doing an exact expectation (Ross et al., 2008). Also, the depth of the search can be reduced by using an approximate value function at the leaves instead of 0.

The value functions computed by offline techniques can often be used for this purpose.

Value Iteration Value iteration algorithms form an important class of offline algorithms that iteratively estimate the optimal value function according to Bellman's equation (3). Most algorithms exploit the piecewise-linear and convex properties of optimal value functions to obtain a finite representation. In other words, optimal value functions V^h are represented by a set Γ^h of α-vectors that correspond to conditional plans. Algorithm 2 shows how to iteratively compute Γ^t by dynamic programming for an increasing number of time steps t.

Unfortunately, the number of α-vectors in each Γ^t increases exponentially with $|\mathcal{O}|$ and doubly exponentially with t. While several approaches can be used to prune α-vectors that are not maximal for any belief, the number of α-vectors still grows exponentially for most problems. Instead, many approaches compute a parsimonious set of α-vectors, which defines a lower bound on the optimal value function. The class of *point-based value iteration* (Pineau et al., 2006) algorithms computes the maximal α-vectors only for a set $\mathcal{B}$ of beliefs. Algorithm 2 describes how the parsimonious set Γ^h of α-vectors associated with a given set $\mathcal{B}$ of beliefs can be computed in time linear with h and $|\mathcal{O}|$ by dynamic programming. Most point-based techniques differ in how they choose $\mathcal{B}$ (which may vary at each iteration), but the general rule of thumb is to include beliefs reachable from the initial belief b_0 since these are the beliefs that are likely to be encountered at execution time.

Algorithm 2 Value Iteration

Inputs: Horizon h
Outputs: Optimal value function Γ^h
$\Gamma^0 \leftarrow \{0\}$
for $t = 1$ to h **do**
  for all $a \in \mathcal{A}$, $\langle \alpha_1, \ldots, \alpha_{|\mathcal{O}|} \rangle \in (\Gamma^{t-1})^{|\mathcal{O}|}$ **do**
    $\alpha'(s) \leftarrow R(s,a) +$
    $\gamma \sum_{o',s'} \Pr(s'|s,a) \Pr(o'|s',a) \alpha_{o'}(s') \forall s$
    $\Gamma^t \leftarrow \Gamma^t \cup \{\alpha'\}$
  end for
end for

Algorithm 3 Point Based Value Iteration

Inputs: Horizon h and set of beliefs $\mathcal{B}$
Outputs: Value function Γ^h
$\Gamma^0 \leftarrow \{0\}$
for $t = 1$ to h **do**
  for all $b \in \mathcal{B}$ **do**
    for all $a \in \mathcal{A}$, $o' \in \mathcal{O}$ **do**
      $b^{ao'}(s') \leftarrow k \Sigma_s b(s) \Pr(s'|s,a) \Pr(o'|s',a) \forall s'$
      $\alpha^{ao'} \leftarrow \text{argmax}_{alpha \in \Gamma^{t-1}} \alpha(b^{ao'})$
    end for
    $a^* \leftarrow \text{argmax}_a R(b,a) + \gamma \Sigma_{o'} \Pr(o'|b,a) \alpha^{ao'}$
    $\alpha'(s) R(s,a) + \gamma \Sigma_{o',s'} \Pr(s'|s,a) \Pr(o'|s',a)$
    $\alpha_{o'}(s') \forall s$
    $\Gamma^t \leftarrow \Gamma^t \cup \{\alpha'\}$
  end for
end for

Policy Search Another important class of offline algorithms consists of policy search techniques. These techniques search for the best policy in a predefined space of policies. For instance, finite state controllers are

a popular policy space due to their generality and simplicity. The search for the best (stochastic) controller of N nodes can be formulated as a non-convex quadratically constrained optimization problem Amato et al., 2007:

$$\max_{x,y,z} \sum_s b_0(s)\ \underbrace{\alpha_0(s)}_{x}$$

$$\text{s.t. } \underbrace{\alpha_n(s)}_{x} = \sum_a \underbrace{[\Pr(a|n)}_{y} R(s,a) + \gamma \sum_{s',o',n'} \Pr(s'|s,a) \Pr(o'|s',a) \underbrace{\Pr(a,n'|n,o')}_{z} \underbrace{\alpha_{n'}(s')}_{x}] \ \forall s,n$$

$$\underbrace{\Pr(a,n'|n,o')}_{x} \geq 0 \ \forall a,n',n,o'$$

$$\sum_{n',a} \underbrace{\Pr(a,n'|n,o)}_{z} = 1 \ \forall n,o$$

$$\sum_{n'} \underbrace{\Pr(a,n'|n,o')}_{z} = \underbrace{\Pr(a|n)}_{y} \ \forall a,n,o'$$

The variables of the optimization problem are the α-vectors and the parameters of the controller ($\Pr(a|n)$ and $\Pr(a,n'|n,o')$). Here, $\Pr(a|n)$ is the action distribution for each node n and $\Pr(a,n'|n,o') = \Pr(a|n)\Pr(n'|a,n,o')$ is the product of the action distribution and successor node distribution for each n,o'-pair. While there does not exist any algorithm that reliably finds the global optimum due to the non-convex nature of the problem, several techniques can be used to find locally optimal policies, including sequential quadratic programming, bounded policy iteration, expectation maximization, stochastic local search, and gradient descent.

Related Work

Although this entry assumes that states, actions, and observations are defined by a single variable, multiple variables can be used to obtain a *factored POMDP* (Boutilier & Poole, 1996). As a result, the state, observation, and action spaces often become exponentially large. Aggregation (Shani et al., 2008; Sim et al., 2008) and compression techniques (Poupart & Boutilier, 2004; Roy et al., 2005) are then used to speed up computation. POMDPs can also be defined for problems with continuous variables. The piecewise-linear and convex properties of optimal value functions still hold in continuous spaces, which allows value iteration algorithms to be easily extended to continuous POMDPs Porta et al., 2006. When a planning problem can naturally be thought as a hierarchy of subtasks, *hierarchical POMDPs* (Theocharous & Mahadevan, 2002; Pineau et al., 2003; Toussaint et al., 2008) can be used to exploit this structure.

In this article, we also assumed that the transition, observation, and reward functions are known, but in many domains they may be (partially) unknown and therefore the decision maker needs to learn about them while acting. This is a problem of *reinforcement learning*. While several policy search techniques have been adapted to simultaneously learn and act (Meuleau et al., 1999; Aberdeen & Baxter, 2002), it turns out that one can treat the unknown parameters of the transition, observation, and reward functions as hidden state variables, which lead to a *Bayes-adaptive POMDP* (Ross et al., 2007; Poupart & Vlassis, 2008). We also assumed a single decision maker, however POMDPs have been extended for multiagent systems. In particular, *decentralized POMDPs* (Amato et al., 2009) can model multiple cooperative agents that share a common goal and *interactive POMDPs* Gmytrasiewicz & Doshi, 2005 can model multiple competing agents.

Cross References

►Markov Decision Process

Recommended Reading

Aberdeen, D., & Baxter, J. (2002). Scalable internal-state policy-gradient methods for POMDPs. In *International Conference on Machine Learning*, pp. 3–10.

Amato, C., Bernstein, D. S., & Zilberstein, S. (2009). Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Journal of Autonomous Agents and Multi-Agent Systems, 21*, 293–320.

Amato, C., Bernstein, D. S., & Zilberstein, S. (2007). Solving POMDPs using quadratically constrained linear programs. In *International Joint Conferences on Artificial Intelligence*, pp. 2418–2424.

Aström, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications, 10*, 174–2005.

Boutilier, C., & Poole, D. (1996). Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 1168–1175

Buede, D. M. (1999). *Dynamic decision networks: An approach for solving the dual control problem*. Cincinnati: Spring INFORMS.

Drake, A. (1962). Observation of a Markov Process through a noisy channel. PhD thesis, Massachusetts Institute of Technology.

Hansen, E. (1997). An improved policy iteration algorithm for partially observable MDPs. In *Neural Information Processing Systems*, pp. 1015–1021.

Hauskrecht, M., & Fraser, H. S. F. (2010). Planning treatment of ischemic heart disease with partially observable Markov decision processes. *Artificial Intelligence in Medicine, 18*, 221–244.

Hoey, J., Poupart, P., von Bertoldi, A., Craig, T., Boutilier, C., & Mihailidis, A. (2010). Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process. *Computer Vision and Image Understanding, 114*, 503–519.

Kaelbling, L. P., Littman, M., & Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*, 99–134.

Meuleau, N., Peshkin, L., Kim, K.-E., & Kaelbling, L. P. (1999). Learning finite-state controllers for partially observable environments. In *Uncertainty in Artificial Intelligence*, pp. 427–436.

Pineau, J. & Gordon, G. (2005). POMDP planning for robust robot control. In *International Symposium on Robotics Research*, pp. 69–82.

Pineau, J., Gordon, G. J., & Thrun, S. (2003). Policy-contingent abstraction for robust robot control. In *Uncertainty in Artificial Intelligence*, pp. 477–484.

Pineau, J., Gordon, G., & Thrun, S. (2006). Anytime point-based approximations for large pomdps. *Journal of Artificial Intelligence Research, 27*, 335–380.

Piotr, J. (2005). Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research, 24*, 49–79.

Porta, J. M., Vlassis, N. A., Spaan, M. T. J., & Poupart, P. (2006). Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research, 7*, 2329–2367.

Poupart, P., & Boutilier, C. (2004). VDCBPI: An approximate scalable algorithm for large POMDPs. In *Neural Information Processing Systems*, pp. 1081–1088.

Poupart, P., & Vlassis, N. (2008). Model-based Bayesian reinforcement learning in partially observable domains. In *International Symposium on Artificial Intelligence and Mathematics* (*ISAIM*).

Puterman, M. L. (1994). *Markov decision processes*. New York: Wiley.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*, 257–286.

Ross, S., Chaib-Draa, B., & Pineau, J. (2007). Bayes-adaptive POMDPs. In *Advances in Neural Information Processing Systems* (*NIPS*).

Ross, S., Pineau, J., Paquet, S., & Chaib-draa, B. (2008). Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research, 32*, 663–704.

Roy, N., Gordon, G. J., & Thrun, S. (2005). Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research, 23*, 1–40.

Shani, G., & Meek, C. (2009). Improving existing fault recovery policies. In *Neural Information Processing Systems*.

Shani, G., Brafman, R. I., Shimony, S. E., & Poupart, P. (2008). Efficient ADD operations for point-based algorithms. In *International Conference on Automated Planning and Scheduling*, pp. 330–337.

Sim, H. S., Kim, K.-E., Kim, J. H., Chang, D.-S., & Koo, M.-W. (2008). Symbolic heuristic search value iteration for factored POMDPs. In *Twenty-Third National Conference on Artificial Intelligence* (*AAAI*), pp. 1088–1093.

Smallwood, R. D., & Sondik, E. J. (1973). The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research, 21*, 1071–1088.

Theocharous, G., & Mahadevan, S. (2002). Approximate planning with hierarchical partially observable Markov decision process models for robot navigation. In *IEEE International Conference on Robotics and Automation*, pp. 1347–1352.

Thomson, B., & Young, S. (2010). Bayesian update of dialogue state: A pomdp framework for spoken dialogue systems. *Computer Speech & Language, 24*, 562–588.

Toussaint, M., Charlin, L., & Poupart, P. (2008). Hierarchical POMDP controller optimization by likelihood maximization. In *Uncertainty in Artificial Intelligence*, pp. 562–570.

Particle Swarm Optimization

James Kennedy
U.S. Bureau of Labor Statistics
Washington, DC, USA

The Canonical Particle Swarm

The particle swarm is a population-based stochastic algorithm for optimization which is based on social–psychological principles. Unlike ►evolutionary algorithms, the particle swarm does not use selection; typically, all population members survive from the beginning of a trial until the end. Their interactions result in iterative improvement of the quality of problem solutions over time.

A numerical vector of D dimensions, usually randomly initialized in a search space, is conceptualized as a point in a high-dimensional Cartesian coordinate system. Because it moves around the space testing new parameter values, the point is well described as a particle. Because a number of them (usually $10 < N < 100$) perform this behavior simultaneously, and because they tend to cluster together in optimal regions of the search space, they are referred to as a *particle swarm*.

Besides moving in a (usually) Euclidean problem space, particles are typically enmeshed in a topological network that defines their communication pattern. Each particle is assigned a number of neighbors to which it is linked bidirectionally.

The most common type of implementation defines the particles' behaviors in two formulas. The first adjusts the velocity or step size of the particle, and the second moves the particle by adding the velocity to its previous position.

On each dimension d:

$$v_{id}^{(t+1)} \leftarrow \alpha v_{id}^{(t)} + U(0,\beta)\left(p_{id} - x_{id}^{(t)}\right) + U(0,\beta)\left(p_{gd} - x_{id}^{(t)}\right) \quad (1)$$

$$x_{id}^{(t+1)} \leftarrow x_{id}^{(t)} + v_{id}^{(t+1)} \quad (2)$$

where i is the target particle's index, d is the dimension, $\vec{x}_i$ is the particle's position, $\vec{v}_i$ is the velocity, $\vec{p}_i$ is the best position found so far by i, g is the index of i's best neighbor, α and β are constants, and $U(0,\beta)$ is a uniform random number generator.

Though there is variety in the implementations of the particle swarm, the most standard version uses $\alpha = 0.7298$ and $\beta = \psi/2$, where $\psi = 2.9922$, following an analysis published in Clerc and Kennedy (2002). The constant α is called an *inertia weight* or *constriction coefficient*, and β is known as the *acceleration constant*.

The program evaluates the parameter vector of particle i in a function $f(\vec{x})$ and compares the result to the best result attained by i thus far, called $pbest_i$. If the current result is i's best so far, the vector $\vec{p}_i$ is updated with the current position $\vec{x}_i$, and the previous best function result $pbest_i$ is updated with the current result.

When the system is run, each particle cycles around a region centered on the centroid of the previous bests $\vec{p}_i$ and $\vec{p}_g$; as these variables are updated, the particle's trajectory shifts to new regions of the search space, the particles begin to cluster around optima, and improved function results are obtained.

The Social–Psychological Metaphor

Classical social psychology theorists considered the pursuit of *cognitive consistency* to be an important motivation for human behavior (Abelson et al., 1968; Festinger, 1957; Heider, 1958). Cognitive elements might have emotional or logical aspects to them which could be consistent or inconsistent with one another; several theorists identified frameworks for describing the degree of consistency and described the kinds of processes that an individual might use to increase consistency or balance, or decrease inconsistency or cognitive dissonance.

Contemporary social and cognitive psychologists frequently cast these same concepts in terms of connectionist principles. Cognitive elements are conceptualized as a network with positive and negative vertices among a set of nodes. In some models, the elements are given and the task is to reduce error by adjusting the signs and values of the connections between them, and in other models the connections are given and the goal of optimization is to find activation values that maximize coherence (Thagard, 2000), harmony (Smolensky, 1986), or some other measure of consistency. Typically, this optimization is performed by gradient-descent programs which psychologically model processes that are private to the individual and are perfectly rational, that is, the individual always decreases error or increases consistency among elements. The particle swarm simulates the optimization of these kinds of structures through social interaction; it is commonly observed, not only in the laboratory but in everyday life, that a person faced with a problem typically solves it by talking with other people.

A direct precursor of the particle swarm is seen in Nowak, Szamrej, and Latané's (1990) cellular automaton simulation of social impact theory's predictions about interaction in human social populations. Social impact theory predicted that an individual was influenced to hold an attitude or belief in proportion to the Strength, Immediacy, and Number of sources of influence holding that position, where Strength was a measure of the persuasiveness or prestige of an individual, Immediacy was their proximity, and Number was literally the number of sources of influence holding a particular attitude or belief. In the simulation, individuals iteratively interacted, taking on the prevalent state of a binary attitude in their neighborhood, until the system reached equilibrium.

The particle swarm extends this model by supposing that various states can be evaluated, for instance, that different patterns of cognitive elements may be more or less dissonant; it assumes that individuals hold

more than one attitude or belief, and that they are not necessarily binary; and Strength is replaced with a measure of self-presented success. One feature usually found in particle swarms and not in the paper by Nowak et al. is the phenomenon of persistence or momentum, the tendency of an individual to keep changing or moving in the same direction from one time-step to the next.

Thus, the particle swarm metaphorically represents the interactions of a number of individuals, none knowing what the goal is, each knowing its immediate state and its best performance in the past, each presenting its neighbors with its best success-so-far at solving a problem, each functioning as both source and target of influence in the dynamically evolving system. As individuals emulate the successes of their neighbors, the population begins to cluster in optimal regions of a search space, reliably discovering good solutions to difficult problems featuring, for instance, nonlinearity, high dimension, deceptive gradients, local optima, etc.

The Population Topology

Several kinds of topologies have been most widely used in particle swarm research; the topic is a current focus of much research. In the *gbest* topology, the population is conceptually fully connected; every particle is linked to every other. In practice, with the best neighbor canonical version, this is simpler to implement than it sounds, as it only means that every particle receives influence from the best performing member of the population.

The *lbest* topology of degree K_i comprises a ring lattice, with the particle linked to its K_i nearest neighbors on both sides in the wrapped population array.

Another structure commonly used in particle swarm research is the von Neumann or "square" topology. In this arrangement, the population is laid out in rows and columns, and each individual is connected to the neighbors above, below, and on each side of it in the toroidally wrapped population. Numerous other topologies have been used, including random (Suganthan, 1999), hierarchical (Janson & Middendorf, 2005), and adaptive ones (Clerc, 2006).

The most important effect of the population topology is to control the spread of proposed problem solutions through the population. As a particle finds a good region of the search space, it may become the best neighbor to one of the particles it is connected to. That particle then will tend to explore in the vicinity of the first particle's success, and may eventually find a good solution there, too; it could then become the best neighbor to one of its other neighbors. In this way, information about good regions of the search space migrates through the population.

When connections are parallel, e.g., when the mean degree of particles is relatively high, then information can spread quickly through the population. On unimodal problems this may be acceptable, but where there are local optima there may be a tendency for the population to converge too soon on a suboptimal solution. The *gbest* topology has repeatedly been shown to be vulnerable to the lure of locally optimal attractors.

On the other hand, where the topology is sparse, as in the *lbest* model, problem solutions spread slowly, and subpopulations may search diverse regions of the search space in parallel. This increases the probability that the population will end up near the global optimum. It also means that convergence will be slower.

Vmax and Convergence

The particle swarm has evolved very much since it was first reported by Kennedy and Eberhart (1995) and Eberhart and Kennedy (1995). Early versions required a system constant *Vmax* to limit the velocity. Without this limit, the particles' trajectories would swing wildly out of control.

Following presentation of graphical representations of a deterministic form of the particle swarm by Kennedy (1998), early analyses by Ozcan and Mohan (1999) led to some understanding of the nature of the particle's trajectory. Analytical breakthroughs by Clerc (reported in Clerc and kennedy (2002)), and empirical discoveries by Shi and Eberhart (1998), resulted in the application of the α constant in concert with appropriate values of the acceleration constant β. These parameters brought the particle under control, allowed convergence under appropriate conditions, and made *Vmax* unnecessary. It is still used sometimes, set to very liberal values such as a half or third of the initialization range of a variable for more efficient swarm behavior, but it is not necessary.

Step Size and Consensus

Step size in the particle swarm is inherently scaled to consensus among the particles. A particle goes in one direction on each dimension until the sign of its velocity is reversed by the accumulation of $(p - x)$ differences; then it turns around and goes the other way. As it searches back and forth, its oscillation on each dimension is centered on the mean of the previous bests $(p_{id}+p_{gd})/2$, and the standard deviation of the distribution of points that are tested is scaled to the difference between them. In fact this function is a very simple one: the standard deviation of a particle's search, when p_{id} and p_{gd} are constants, is approximately $|(p_{id} - p_{gd})|$. This means that when the particles' previous best points are far from one another in the search space, the particles will take big steps, and when they are nearer the particles will take little steps.

Over time, this usually means that exploring behavior is seen in early iterations and exploiting behavior later on as particles come to a state of consensus. If it happens, however, that a particle that has begun to converge in one part of the search space receives information about a good region somewhere else, it can return to the exploratory mode of behaving.

The Fully Informed Particle Swarm (FIPS)

Mendes (2004) reported a version of swarm that featured an alternative to the best neighbor strategy. While the canonical particle is influenced by its own previous success and the previous success of its best neighbor, the fully informed particle swarm (FIPS) allowed influence by all of a particle's neighbors. The acceleration constants were set to $\beta = \psi/2$ in the traditional version; it was defined in this way because what mattered was their sum, which could be distributed among any number of difference terms. In the standard algorithm there were two of them, and thus the sum was divided by 2. In FIPS a particle of K_i degree has coefficients $\beta = \psi/K_i$.

The FIPS particle swarm removed two aspects that were considered standard features of the algorithm. First of all, the particle i no longer influenced itself directly, e.g., there is no $\vec{p}_i$ in the formula. Second, the best neighbor is now averaged in with the others; it was not necessary to compare the successes of all neighbors to find the best one.

Mendes found that the FIPS swarm was more sensitive than the canonical versions to the differences in topology. For instance, while in the standard versions the fully connected *gbest* topology meant influence by the best solution known to the entire population, in FIPS *gbest* meant that the particle was influenced by a stochastic average of the best solutions found by all members of the population; the result tended to be near-random search.

The lesson to be learned is that the *meaning* of a topology depends on the mode of interaction. Topological structure (and Mendes tested more than 1,340 of them) affects performance, but the way it affects the swarm's performance depends on how information is propagated from one particle to another.

Generalizing the Notation

Equation 2 above shows that the position is derived from the previous iteration's position plus the current iteration's velocity. By rearranging the terms, it can be shown that the current iteration's velocity $\vec{v}_i^{(t+1)}$ is the difference between the new position and the previous one: $\vec{v}_i^{(t+1)} = \vec{x}_i^{(t+1)} - \vec{x}_i^{(t)}$. Since this happened on the previous time-step as well, it can be shown that $\vec{v}_i^{(t)} = \vec{x}_i^{(t)} - \vec{x}_i^{(t-1)}$; this fact makes it possible to combine the two formulas into one:

$$x_{id}^{(t+1)} \leftarrow x_{id}^{(t)} + \alpha\left(x_{id}^{(t)} - x_{id}^{(t-1)}\right) + \sum U\left(0, \frac{\psi}{K_i}\right)\left(p_{kd} - x_{id}^{(t)}\right) \quad (3)$$

where K_i is the degree of node i, k is the index of i's kth neighbor, and adapting Clerc's (Clerc & Kennedy, 2002) scheme $\alpha = 0.7298$ and $\psi = 2.9922$.

In the canonical best neighbor particle swarm, $K_i = 2$, $\forall\ i : i = 1, 2, \ldots, N$ and $k \in (i, g)$, that is, k takes the values of the particle's own index and its best neighbor's index. In FIPS, K_i may vary, depending on the topology, and k takes on the indexes of each of i's neighbors. Thus, Eq.3 is a generalized formula for the trajectories of the particles in the particle swarm.

This notation can be interpreted verbally as:

$$\begin{aligned} \textit{NEW POSITION} = \textit{CURRENT POSITION} \\ + \textit{PERSISTENCE} \\ + \textit{SOCIAL INFLUENCE} \end{aligned} \quad (4)$$

That is, on every iteration, every particle on every dimension starts at the point it last arrived at, persists some weighted amount in the direction it was previously going, then makes some adjustments based on the differences between the best previous positions of its sources of influence and its own current position in the search space.

The Evolving Paradigm

The particle swarm paradigm is young, and investigators are still devising new ways to understand, explain, and improve the method. A divergence or bifurcation of approaches is observed: some researchers seek ways to simplify the algorithm (Owen & Harvey, 2007; Peña, Upegui, & Eduardo Sanchez, 2006), to find its essence, while others improve performance by adding features to it, e.g., (Clerc, 2006). The result is a rich unfolding research tradition with innovations appearing on many fronts.

Although the entire algorithm is summarized in one simple formula, it is difficult to understand how it operates or why it works. For instance, while the *Social Influence* terms point the particle in the direction of the mean of the influencers' successes, the *Persistence* term offsets that movement, causing the particle to bypass what seems to be a reasonable target. The result is a spiral-like trajectory that goes past the target and returns to pass it again, with the spiral tightening as the neighbors come to consensus on the location of the optimum.

Further, while authors often talk about the particle's velocity carrying it "toward the previous bests," in fact the velocity counterintuitively carries it *away from* the previous bests as often as toward them. It is more accurate to say the particle "explores around" the previous bests, and it is hard to describe this against-the-grain movement as "gradient descent," as some writers would like.

It is very difficult to visualize the effect of ever-changing sources of influence on a particle. A different neighbor may be best from one iteration to the next; the balance of the random numbers may favor one or another or some compromise of sources; the best neighbor could remain the same one, but may have found a better $\vec{p}_i$ since the last turn; and so on. The result is that the particle is pulled and pushed around in a complex way, with many details changing over time.

The paradoxical finding is that it is best not to give the particle information that is too good, especially early in the search trial. Premature convergence is the result of amplified consensus resulting from too much communication or overreliance on best neighbors, especially the population best. Various researchers have proposed ways to slow the convergence or clustering of particles in the search space, such as occasional reinitialization or randomization of particles, repelling forces among them, etc., and these techniques typically have the desired effect. In many cases, however, implicit methods work as well and more parsimoniously; the effect of topology on convergence rate has been mentioned here, for instance.

Binary Particle Swarms

A binary particle swarm is easily created by treating the velocity as a probability threshold (Kennedy & Eberhart, 1997). Velocity vector elements are squashed in a sigmoid or other function, for instance $S(v) = 1/(1+exp(-v))$, producing a result in (0..1). A random number is generated and compared to $S(v_{id})$ to determine whether x_{id} will be a 0 or a 1. Though discrete systems of higher cardinality have been proposed, it is difficult to define such concepts as distance and direction in a meaningful way within nominal data.

Alternative Probability Distributions

As was noted above, the particle's search is centered around the mean of the previous bests that influence it, and its variance is scaled to the differences among them. This has suggested to several researchers that perhaps the trajectory formula can be replaced, wholly or partly, by some type of random number generator that directly samples the search space in a desirable way.

Kennedy (2003) suggested simple Gaussian sampling, using a random number generator (RNG) $G(mean, s.d.)$ with the mean centered between $\vec{p}_i$ and $\vec{p}_g$, and with the standard deviation defined on each dimension as $s.d. = |(p_{id} - p_{gd})|$. This "bare bones" particle swarm eliminated the velocity component; it performed rather well on a set of test functions, but not as well as the usual version.

Krohling (2004) simply substituted the absolute values of Gaussian-distributed random numbers for the uniformly distributed values in the canonical particle swarm. He and his colleagues have had success on a range of problems using this approach. Richer and Blackwell (2006) replaced the Gaussian distribution of bare bones with a Lévy distribution. The Lévy distribution is bell-shaped like the Gaussian but with fatter tails. It has a parameter α which allows interpolation between the Cauchy distribution ($\alpha = 1$) and Gaussian ($\alpha = 2$) and can be used to control the fatness of the tails. In a series of trials, Richer and Blackwell (2006) were able to emulate the performance of a canonical particle swarm using $\alpha = 1.4$. Kennedy (2005) used a Gaussian RNG for the social influence term of the usual formula, keeping the "persistence" term found in the standard particle swarm. Variations on this format produced results that were competitive with the canonical version.

Numerous other researchers have begun exploring ways to replicate the overall behavior of the particle swarm by replacing the traditional formulas with alternative probability distributions. Such experiments help theorists understand what is essential to the swarm's behavior and how it is able to improve its performance on a test function over time.

Simulation of the canonical trajectory behavior with RNGs is a topic that is receiving a great deal of attention at this time, and it is impossible to predict where the research is leading. As numerous versions have been published showing that the trajectory formulas can be replaced by alternative strategies for selecting a series of points to sample, it becomes apparent that the essence of the paradigm is not to be found in the details of the movements of the particles, but in the nature of their interactions over time, the structure of the social network in which they are embedded, and the function landscape with which they interact, with all these factors working together gives the population the ability to find problem solutions.

Recommended Reading

Abelson, R. P., Aronson, E., McGuire, W. J., Newcomb, T. M., Rosenberg, M. J., & Tannenbaum, R. H. (Eds.), (1968). *Theories of cognitive consistency: A sourcebook*. Chicago: Rand McNally.

Clerc, M. (2006). *Particle swarm optimization*. London: Hermes Science Publications.

Clerc, M., & Kennedy, J. (2002). The particle swarm: Explosion, stability, and convergence in a multi-dimensional complex space. *IEEE Transactions on Evolutionary Computation, 6*, 58–73.

Eberhart, R.C., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings of the 6th international symposium on micro machine and human science, (Nagoya, Japan)* (pp. 39–43). Piscataway, NJ: IEEE Service Center.

Festinger, L. (1957). *A theory of cognitive dissonance*. Stanford, CA: Stanford University Press.

Heider, F. (1958). *The psychology of interpersonal relations*. New York: Wiley.

Janson, S., & Middendorf, M. (2005). A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man, and Cybernatics – Part B: Cybernatics*, *35*(6), 1272–1282.

Kennedy, J. (1998). The behavior of particles. In V. W. Porto, N. Saravanan, D. Waagen, & A. E. Eiben (Eds.), *Evolutionary programming VII. Proceedings of the 7th annual conference on evolutionary programming*.

Kennedy, J. (2003). Bare bones particle swarms. In *Proceedings of the IEEE swarm intelligence symposium* (pp. 80–87). Indianapolis, IN.

Kennedy, J. (2005). Dynamic-probabilistic particle swarms. In *Proceedings of the genetic and evolutionary computation conference (GECCO-2005)* (pp. 201–207). Washington, DC.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the 1995 IEEE international conference on neural networks (Perth, Australia)* (pp. 1942–1948). Piscataway, NJ: IEEE Service Center.

Kennedy, J., & Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. In *Proceedings of the 1997 conference on systems, man, and cybernetics* (pp. 4104–4109). Piscataway, NJ: IEEE Service Center.

Krohling, R. A. (2004). Gaussian Swarm. A novel particle swarm optimization algorithm. *Proceedings of the 2004 IEEE conference on cybernetics and intelligent systems* (vol. 1, pp. 372–376).

Mendes, R. (2004). *Population topologies and their influence in particle swarm performance*. Doctoral thesis, Escola de Engenharia, Universidade do Minho, Portugal.

Nowak, A., Szamrej, J., & Latané, B. (1990). From private attitude to public opinion: A dynamic theory of social impact. *Psychological Review, 97*, 362–376.

Owen, A., & Harvey, I. (2007). Adapting particle swarm optimisation for fitness landscapes with neutrality. In *Proceedings of the 2007 IEEE swarm intelligence symposium* (pp. 258–265). Honolulu, HI: IEEE Press.

Ozcan, E., & Mohan, C. K. (1999). Particle swarm optimization: Surfing the waves. In *Proceedings of the congress on evolutionary computation, Mayflower hotel, Washington D.C.* (pp. 1939–1944). Piscataway, NJ: IEEE Service Center.

Peña, J., Upegui, A., & Eduardo Sanchez, E. (2006). Particle swarm optimization with discrete recombination: An online optimizer for evolvable hardware. In *Proceedings of the 1st NASA/ESA conference on adaptive hardware and systems (AHS-2006), Istanbul, Turkey* (pp. 163–170). Piscataway, NJ: IEEE Service Center.

Richer, T. J., & Blackwell, T. M. (2006). The Levy particle swarm. In *Proceedings of the 2006 congress on evolutionary computation (CEC-2006)*. Piscataway, NJ: IEEE Service Center.

Shi, Y., & Eberhart, R. C. (1998). Parameter selection in particle swarm optimization. In *Evolutionary Programming VII: Proc. EP98* (pp. 591–600). New York: Springer.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group, (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1, Foundations* (pp. 194–281). Cambridge, MA: MIT Press.

Suganthan, P. N. (1999). Particle swarm optimisation with a neighbourhood operator. In *Proceedings of congress on evolutionary computation*. Washington DC, USA.

Thagard, P. (2000). *Coherence in thought and action*. Cambridge, MA: MIT Press.

Partitional Clustering

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

Definition

Partitional clustering decomposes a data set into a set of disjoint clusters. Given a data set of N points, a partitioning method constructs K ($N \geq K$) partitions of the data, with each partition representing a cluster. That is, it classifies the data into K groups by satisfying the following requirements: (1) each group contains at least one point, and (2) each point belongs to exactly one group. Notice that for fuzzy partitioning, a point can belong to more than one group.

Many partitional clustering algorithms try to minimize an objective function. For example, in K-means and K-medoids the function (also referred to as the distortion function) is

$$\sum_{i=1}^{K} \sum_{j=1}^{|C_i|} \text{Dist}(x_j, \text{center}(i)), \quad (1)$$

where $|C_i|$ is the number of points in cluster i, $\text{Dist}(x_j, \text{center}(i))$ is the distance between point x_j and center i. Many distance functions can be used, such as Euclidean distance and L_1 norm.

The following entries describe several representative algorithms for partitional data clustering - ►K-means clustering, ►K-medoids clustering, ►Quality Threshold Clustering, ►Expectation Maximization Clustering, ►mean shift, ►Locality Sensitive Hashing Based Clustering, and ►K-way Spectral Clustering. In the K-means algorithm, each cluster is represented by the mean value of the points in the cluster. For the K-medoids algorithm, each cluster is represented by one of the points located near the center of the cluster. Instead of setting cluster number K, the Quality Threshold algorithm uses the maximum cluster diameter as a parameter to find clusters with guaranteed quality. Expectation Maximization clustering performs expectation-maximization analysis based on statistical modeling of the data distribution, and it has more parameters. Mean Shift is a nonparameter algorithm to find any shape of clusters using density estimator. Locality Sensitive Hashing performs clustering by hashing similar points to the same bin. K-way spectral clustering represents the data as a graph and performs graph partitioning to find clusters.

Recommended Reading

Han, J., & Kamber, M. (2006). *Data mining: Concepts and techniques* (2nd ed.). San Francisco: Morgan Kaufmann Publishers.

Passive Learning

A ►passive learning system plays no role in the selection of its ►training data. Passive learning stands in contrast to ►active learning.

PCA

►Principal Component Analysis

PCFG

►Probabilistic Context-Free Grammars

Phase Transitions in Machine Learning

Lorenza Saitta[1], Michele Sebag[2]
[1]Università del Piemonte Orientale, Alessandria, Italy
[2]CNRS – INRIA – Université Paris-Sud, Orsay, France

Synonyms

Statistical Physics of learning; Threshold phenomena in learning; Typical complexity of learning

Definition

Phase transition (PT) is a term originally used in physics to denote the transformation of a system from a liquid, solid, or gas state (phase) to another. It is used, by extension, to describe any abrupt and sudden change in one of the *order* parameters describing an arbitrary system, when a *control* parameter approaches a *critical* value (While early studies on PTs in computer science inverted the notions of *order* and *control* parameters, this article will stick to the original definition used in Statistical Physics.).

Far from being limited to physical systems, PTs are ubiquitous in sciences, notably in computational science. Typically, hard combinatorial problems display a PT with regard to the probability of existence of a solution. Note that the notion of PT cannot be studied in relation to single-problem instances: it refers to emergent phenomena in an *ensemble* of problem instances, governed by a given probability distribution.

Motivation and Background

Cheeseman, Kanefsky, and Taylor (1991) were most influential in starting the study of PTs in Artificial Intelligence, experimentally showing the presence of a PT containing the most difficult instances for various **NP**-complete problems. Since then, the literature flourished both in breadth and depth, witnessing an increasing transfer of knowledge and results between Statistical Physics and Combinatorics.

As far as machine learning (ML) can be formulated as a combinatorial optimization problem (Mitchell, 1982), it is no surprise that PTs emerge in many of its facets. Early results have been obtained in the field of relational learning, either logic- (Botta, Giordana, Saitta, & Sebag, 2003; Giordana & Saitta, 2000) or kernel- (Gaudel, Sebag, & Cornuéjols, 2008) based. PTs have been studied in Neural Networks (Demongeot & Sené, 2008; Engel & Van den Broeck, 2001), Grammatical inference (Cornuéjols & Sebag, 2008), propositional classification (Baskiotis & Sebag, 2004; Rückert & De Raedt, 2008), and sparse regression (Donoho & Tanner, 2005).

Two main streams of research work emerge from the study of PT in computational problems. On the one hand, locating the PT enables to generate very difficult problem instances, most relevant to benchmark and comparatively assess new algorithms. On the other hand, PT studies stimulate the analysis of algorithmic *typical case complexity*, as opposed to the standard worst-case analysis of algorithmic complexity. It is well known that while many algorithms require exponential resources in the worst case, they are effective for a vast majority of problem instances. Studying their typical runtime thus makes sense in a probabilistic perspective (The typical runtime not only reflects the most probable runtime; overall, the probability of deviating from this typical complexity goes to 0 as the problem size increases.).

Relational Learning

In a seminal paper, Mitchell characterized ML as a search problem (Mitchell, 1982). Much attention has ever since been devoted to every component of a search problem: the search space, the search goal, and the search engine.

The search space $\mathcal{H}$ reflects the language $\mathcal{L}$ chosen to express the target knowledge, termed ►*hypothesis language*. The reader is referred to other entries of the encyclopedia (►*Attribute-value* representation, ►*First-order logic*, ►*Relational learning*, and ►*Inductive Logic Programming*) for a comprehensive presentation of the hypothesis languages and related learning approaches.

Typically, a learner proceeds iteratively: given a set $\mathcal{E}$ of examples labeled after a target concept ω, the learner maintains a list of candidate hypotheses, assessing their *completeness* (the proportion of positive examples they cover) and their *consistency* (the proportion of negative examples they do not cover) using a ►*covering test*. The covering test, checking whether some hypothesis h covers some example e, thus is a key component of

the learning process, launched a few hundred thousand times in each learning run on medium-size problems.

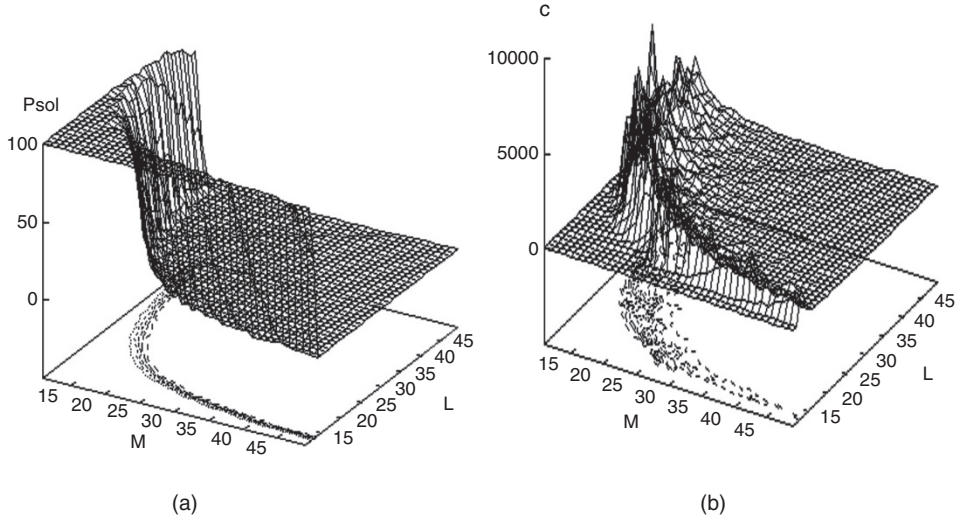
While in propositional learning the covering test is straightforward and computationally efficient, in First-Order Logics one must distinguish between *learning from interpretation* (h covers a set of facts e iff e is a model for h) and *learning from entailment* (h covers a clause e iff h entails e) (De Raedt, 1997). A correct but incomplete covering test, the ▶θ-subsumption test defined by Plotkin (1970) is most often used for its decidability properties, and much attention has been paid to optimizing it (Maloberti & Sebag, 2004).

As shown by Giordana and Saitta (2000), the θ-subsumption test is equivalent to a constraint satisfaction problem (CSP). A finite CSP is a tuple ($\mathbf{X}$, $\mathbf{R}$, D), where $\mathbf{X} = \{x_1, \ldots x_n\}$ is a set of variables, $\mathbf{R} = \{R_1, \ldots R_c\}$ is a set of constraints (relations), and D is the variable domain. Each relation R_h involves a subset of variables $x_{i_1}, \ldots, x_{i_k}$ in $\mathbf{X}$; it specifies all tuples of values $(a_{i_1}, \ldots, a_{i_k})$ in D^k such that the assignment $([x_{i_1} = a_{i_1}] \wedge \ldots \wedge [x_{i_k} = a_{i_k}])$ satisfies R_h. A CSP is satisfiable if there exists a tuple $(a_1, \ldots, a_n) \in D^n$ such that the assignment $([x_i = a_i], i = 1, \ldots, n)$ satisfies all relations in $\mathbf{R}$. Solving a CSP amounts to finding such a tuple (solution) or showing that none exists.

The probability for a random CSP instance to be satisfiable shows a PT with respect to the constraint density (control parameter $p_1 = \frac{2c}{n(n-1)}$) and constraint tightness ($p_2 = 1 - \frac{N}{L^2}$), where N denotes the cardinality of each constraint (assumed to be equal for all constraints) and L is the number of constants in the example (the universe).

The relational covering test being a CSP, a PT was expected; it has been confirmed from ample empirical evidence (Botta, Giordana, & Saitta, 1999; Giordana & Saitta, 2000). The order parameter is the probability for hypothesis h to cover example e; the control parameters are the number m of predicates and the number n of variables in h, on the one hand, and the number N of literals built on each predicate symbol (relation) and the number L of constants in example e, on the other hand. As shown in Fig. 1a, the covering probability is close to 1 (YES region) when h is general comparatively to e; it abruptly decreases to 0 (NO region) as the number m of predicates in h increases and/or the number L of constants in e decreases. In the PT region a high peak of empirical complexity of the covering test is observed (Fig. 1b).

The PT of the covering test has deep and far reaching effects on relational learning. By definition, non-trivial hypotheses (covering some examples but not all)



Phase Transitions in Machine Learning. Figure 1. PT of the covering test (h, e) versus the number m of predicates in h and the number L of constants in e. The number n of variables is set to 10 and the number N of literals per predicate is set to 100. (a) Percentage of times the covering test succeeds. (b) Runtime of the covering test, averaged over 100 pairs (h, e) independently generated for each pair (m, L)

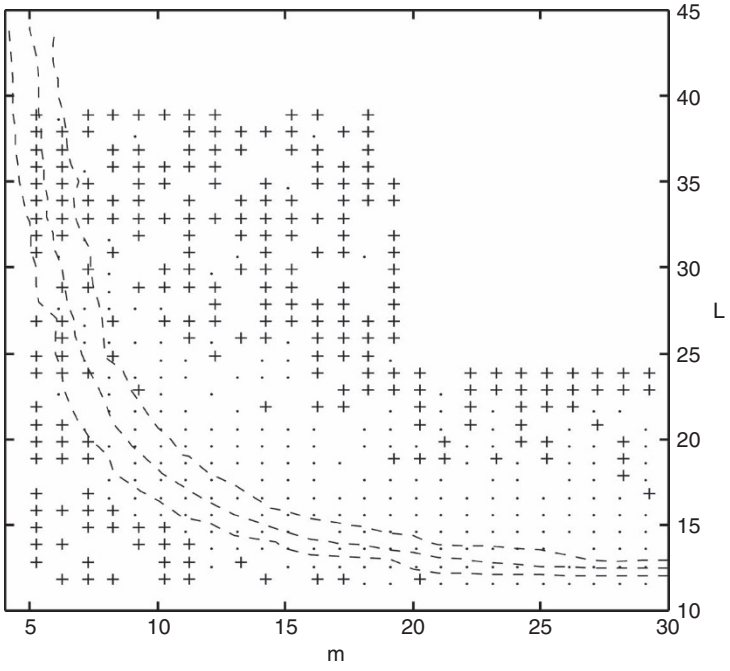
mostly belong to the PT region. The learner, searching for hypotheses covering the positive and rejecting the negative examples, is bound to explore this region and thus cannot avoid the associated computational cost. More generally, the PT region acts as an *attractor* for any learner aimed at complete and consistent hypotheses.

Secondly, top-down learners are bound to traverse the plateau of overly general hypotheses (YES region) before arriving at the PT region. In the YES region, as all hypotheses cover most examples, the learner does not have enough information to make relevant choices; the chance of gradually arriving at an accurate description of the target concept thus becomes very low. Actually, a *blind spot* has been identified close to the PT (Botta et al., 2003): when the target concept lies in this region (relatively to the available examples) every state-of-the-art top-down relational learner tends to build random hypotheses, that is, the learned hypotheses behave like random guessing on the test set (Fig 2).

This negative result has prompted the design of new relational learners, aimed at learning in the PT region and using either prior knowledge about the size of the target concept (Ales Bianchetti, Rouveirol, & Sebag, 2002) or near-miss examples (Alphonse & Osmani, 2008).

Relational Kernels and MIL Problems

Relational learning has been revisited through the so-called *kernel trick* (Cortes & Vapnik, 1995), first pioneered in the context of ▶Support Vector Machines. Relational kernels, inspired from Haussler's convolutional kernels (Haussler, 1999), have been developed for, e.g., strings, trees, or graphs. For instance, $K(\mathbf{x}, \mathbf{x}')$ might count the number of patterns shared by relational structures $\mathbf{x}$ and $\mathbf{x}'$. Relational kernels thus achieve a particular type of ▶propositionalization (Kramer,



Phase Transitions in Machine Learning. Figure 2. Competence map of FOIL versus number *m* of predicates in the target concept and number *L* of constants in the examples. The target concept involves *n* = 4 variables and each example contains *N* = 100 literals built on each predicate symbol. For each pair (m, L), a target concept ω has been generated independently, balanced 200-example training and test sets have been generated and labeled after ω. FOIL has been launched on the training set and the predictive accuracy of the hypothesis has been assessed on the test set. Symbol "–" indicates a predictive accuracy greater than 90%; symbol "–" indicates a predictive accuracy close to 50% (akin random guessing)

Lavrac, & Flach, 2001), mapping every relational example onto a propositional space defined after the training examples.

The question of whether relational kernels enable avoiding the PT faced by relational learning, described in the previous section, was investigated by Gaudel, Sebag, and Cornuéjols (2007), focusing on the so-called ▶multi-instance learning (MIL) setting. The MIL setting, pioneered by Dietterich, Lathrop, and Lozano-Perez (1997), is considered to be the "missing link" between relational and propositional learning (De Raedt, 1998).

Multi-Instance Learning: Background and Kernels

Formally, an MI example $\mathbf{x}$ is a bag of (propositional) instances noted $x^{(1)}, \ldots, x^{(N)}$, where $x^{(j)} \in \mathbb{R}^d$. In the original MI setting (Dietterich et al., 1997), an example is labeled positive iff it includes at least one instance satisfying some target concept C:

$$pos(\mathbf{x}) \text{ iff } \exists\, i \in 1 \ldots N \; s.t. \; C(x^{(i)}).$$

More generally, in application domains such as image categorization, the example label might depend on the properties of several instances:

$$pos(\mathbf{x}) \text{ iff } \forall\, j = 1 \ldots m, \; \exists\, i_j \in 1 \ldots N \; s.t. \; C_j(x^{(i_j)}).$$

In this more general setting, referred to as *presence-based* setting, it has been shown that MIL kernels do have a PT too (Gaudel et al., 2007).

Let us consider bag kernels K, built on the top of propositional kernels k on $\mathbb{R}^d$ as follows, where $\mathbf{x} = (x^{(1)}, \ldots, x^{(N)})$ and $\mathbf{xp} = (x'^{(1)}, \ldots, x'^{(N')})$ denote two MI examples:

$$K(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}).f(\mathbf{x}') \sum_{k=1}^{N} \sum_{\ell=1}^{N'} k(x^{(k)}, x'^{(\ell)}) \qquad (1)$$

where $f(\mathbf{x})$ corresponds to a normalization term, e.g., $f(\mathbf{x}) = 1$ or $1/N$ or $1/\sqrt{K(\mathbf{x}, \mathbf{x})}$.

By construction, such MI-kernels thus consider the average similarity among the example instances while relational learning is usually concerned with finding existential concepts.

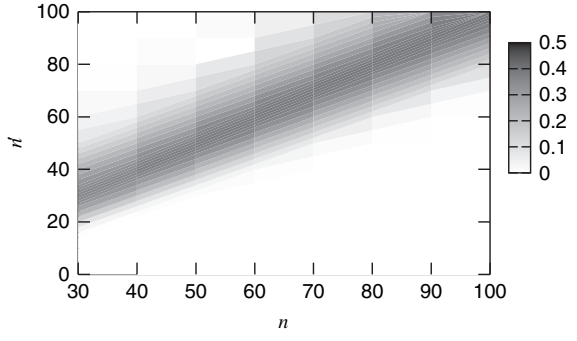
The MI-SVM PT

After Botta et al. (2003) and Giordana and Saitta (2000), the competence of MI-kernels was experimentally assessed using artificial problems. Each problem involves m sub-concept s C_i: a given sub-concept corresponds to a region of the d-dimensional space, and it is satisfied by an MI example $\mathbf{x}$ if at least one instance in $\mathbf{x}$ belongs to this region. An instance is said to be relevant if it belongs to some C_i region.

Let n (respectively n') denote the number of relevant instances in positive (respectively negative) examples. Let further τ denote the number of sub-concept s not satisfied by negative examples (by definition, a positive example satisfies all sub-concept s).

Ample empirical investigations (Gaudel et al., 2007) show that:

- The $n = n'$ region is a failure region, where hypotheses learned by relational MI-SVMs do no better than random guessing (Fig 3). In other words, while MI-SVMs grasp the notion of relevant instances, they still fail in the "truly relational region" where positive and negative examples only differ in the distribution of the relevant instances.
- The width of the failure region increases as τ increases, i.e., when fewer sub-concept s are satisfied by negative examples. This unexpected result is explained from the variance of the kernel-based propositionalization: the larger τ, the more the distribution of the positive and negative propositionalized examples overlap, hindering the discrimination.



Phase Transitions in Machine Learning. Figure 3. MI-SVM Failure Region in the (n, n') plane. Each (n, n') point reports the test error, averaged on 40 artificial problems

Propositional Learning and Sparse Coding

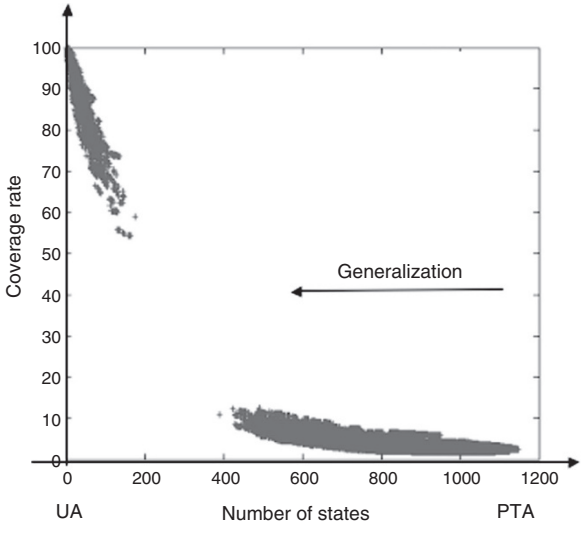
Interestingly, the emergence of a PT is not limited to relational learning. In the case of (Context Free) ►*Grammar induction* for instance (Cornuéjols & Sebag, 2008), the coverage of the candidate grammar was found to abruptly go to 1 along (uniform) generalization, as depicted in Fig. 4.

Propositional learning also displays some PTs both in the classification (Baskiotis & Sebag, 2004; Rückert & De Raedt, 2008) and in the regression (Cands, 2008; Donoho & Tanner, 2005) context.

Propositional Classification

Given a target hypothesis language, classification in discrete domains most often aims at the simplest expression complying with the training examples.

Considering randomly generated positive and negative examples, Rückert and De Raedt (2008) investigated the existence of k-term DNF solutions (disjunction of at most k conjunctions of literals) and showed that the probability of solution abruptly drops as the number of negative examples increases. They proposed a combinatorial optimization algorithm to find a k-term DNF complying with the training examples except at most ε% of them (Rückert & De Raedt, 2008).



Phase Transitions in Machine Learning. Figure 4. Gap emerging during learning in the relationship between the number of nodes of the inferred grammar and the coverage rate

Considering positive and negative examples generated after some k-term DNF target concept ω, Baskiotis and Sebag examined the solutions built by C4.5-Rules (Quinlan, 1993), among the oldest and still most used discrete learning algorithms. The observed variable is the generalization error on a test set; the order variables are the coverage of ω and the average coverage of the conjuncts in ω. Interestingly, C4.5 displays a PT behavior (Fig. 5): the error abruptly increases as the coverage and average coverage decrease.

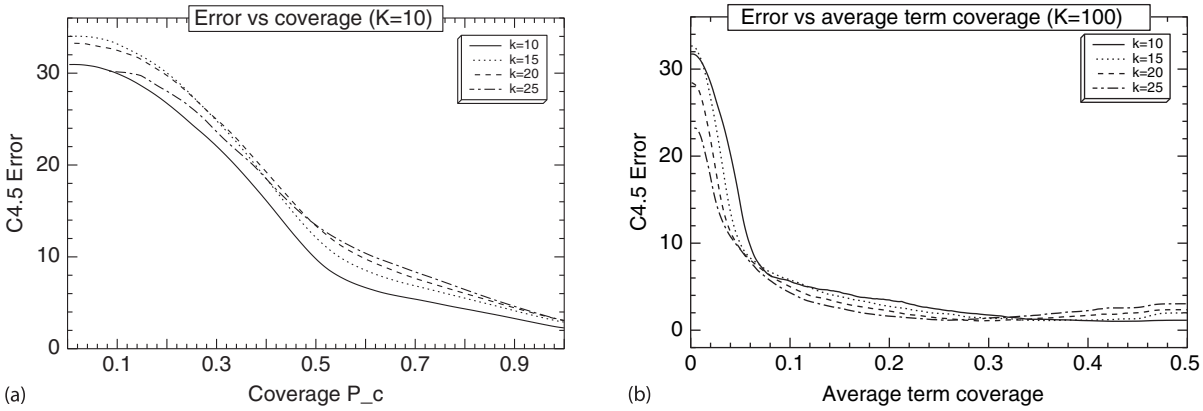
Propositional Regression

►Linear regression aims at expressing the target variable as the weighted sum of the N descriptive variables according to some vector $\mathbf{w}$. When the number N of variables is larger than the number n of examples, one is interested in finding the most sparse $\mathbf{w}$ complying with the training examples (s.t. $< \mathbf{w}, \mathbf{x_i} > = \mathbf{y_i}$). The sparsity criterion consists of minimizing the L_0 norm of $\mathbf{w}$ (number of nonzero coefficients in $\mathbf{w}$), which defines an NP optimization problem. A more tractable formulation is obtained by minimizing the L_1 norm instead:

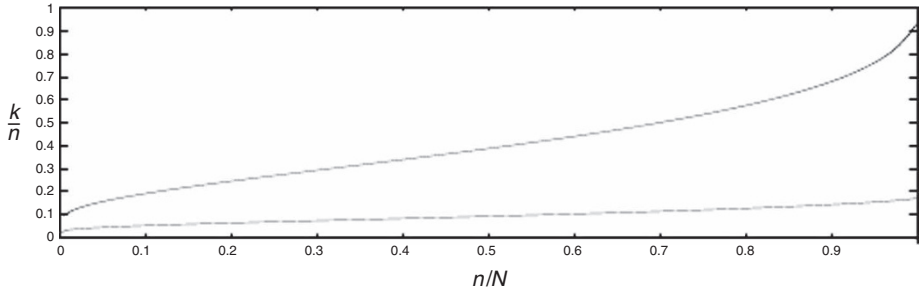
$$\text{Find } \arg\min_{\mathbf{w}\in\mathbf{R}^N}\{||\mathbf{w}||_1 \text{ subject to } < \mathbf{w}, \mathbf{x_i} > = \mathbf{y_i}, \quad i = 1 \ldots n\}. \tag{2}$$

A major result in the field of sparse coding can be stated as: *Let $\boldsymbol{w}^*$ be the solution of Eq. (2); if it is sufficiently sparse, $\boldsymbol{w}^*$ also is the most sparse vector subject to $< \boldsymbol{w}, \boldsymbol{x_i} > = \boldsymbol{y_i}$* (Donoho & Tanner, 2005). In such cases, the L_0 norm minimization can be solved by L_1 norm minimization (an NP optimization problem is solved using linear programming). More generally, the equivalence between L_0 and L_1 norm minimization shows a PT behavior: when the sparsity of the solution is lower than a given threshold w.r.t the problem size (lower curve in Fig. 6), the NP/LP equivalence holds strictly; further, there exists a region (between the upper and lower curves in Fig. 6) where the NP/LP equivalence holds with high probability.

This highly influential result bridges the gap between the statistical and algorithmic objectives. On the statistical side, the importance of sparsity in terms of robust coding (hence learning) is acknowledged since



Phase Transitions in Machine Learning. Figure 5. C4.5 error versus concept coverage (a) and average term coverage (b) in *k*-term DNF languages. The reported curve is obtained by Gaussian convolution with empirical data (15,000 learning problems, each one involving a 800-example dataset)



Phase Transitions in Machine Learning. Figure 6. Strong and weak PT in sparse regression (Donoho & Tanner, 2005). The *x*-axis is the ratio between the number *n* of constraints and the number *N* of variables; the *y*-axis is the ratio between the number *k* of variables involved in the solution, and *n*

the beginnings of Information Theory; on the algorithmic side, the sparsity criterion cannot be directly tackled as it boils down to solving a combinatorial optimization problem (minimizing a L_0 norm). The above result reconciles sparsity and tractability by noting that under some conditions the solution of the L_0 minimization problem can be found by solving the (tractable) L_1 minimization problem: whenever the solution of the latter problem is "sufficiently" sparse, it is also the solution of the former problem.

Perspectives

Since the main two formulations of ML involve constraint satisfaction and constrained optimization, it is no surprise that CSP PTs manifest themselves in ML. The diversity of these manifestations, ranging from relational learning (Botta et al., 2003) to sparse regression (Donoho & Tanner, 2005), has been illustrated in this entry, without pretending to exhaustivity.

Along this line, the research agenda and methodology of ML can benefit from the lessons learned in the CSP field. Firstly, algorithms must be assessed on problems lying in the PT region; results obtained on problems in the easy regions are likely to be irrelevant (*playing in the sandbox* Hogg, Huberman, & Williams, 1996).

In order to do so, the PT should be localized through defining control and order parameters, thus delineating several regions in the control parameter space (ML landscape). These regions expectedly correspond to different types of ML difficulty, beyond the classical computational complexity perspective.

Secondly, the response of a given algorithm to these difficulties can be made available through a competence

map, depicting its average performance conditionally to the value of the control parameters as shown in Figs. 2–3.

Finally, such competence maps can be used to tell whether a given algorithm is *a priori* relevant in a given region of the control parameter space, and support the algorithm selection task (a.k.a. meta-learning; see e.g., http:// www.cs.bris.ac.uk/ Research/ MachineLearning/ metal.html).

Recommended Reading

Ales Bianchetti, J., Rouveirol, C., & Sebag, M. (2002). Constraint-based learning of long relational concepts. In C. Sammut (Ed.), *Proceedings of international conference on machine learning, ICML'02*, (pp. 35–42). San Francisco, CA: Morgan Kauffman.

Alphonse, E., & Osmani, A. (2008). On the connection between the phase transition of the covering test and the learning success rate. *Machine Learning, 70*(2–3), 135–150.

Baskiotis, N., & Sebag, M. (2004). C4.5 competence map: A phase transition-inspired approach. In *Proceedings of international conference on machine learning*, Banff, Alberta, Canada (pp. 73–80). Morgan Kaufman.

Botta, M., Giordana, A., & Saitta, L. (1999). An experimental study of phase transitions in matching. In *Proceedings of the 16th international joint conference on artificial intelligence*, Stockholm, Sweden (pp. 1198–1203).

Botta, M., Giordana, A., Saitta, L., & Sebag, M. (2003). Relational learning as search in a critical region. *Journal of Machine Learning Research, 4*, 431–463.

Cands, E. J. (2008). The restricted isometry property and its implications for compressed sensing. *Compte Rendus de l'Academie des Sciences, Paris, Serie I, 346*, 589–592.

Cheeseman, P., Kanefsky, B., & Taylor, W. (1991). Where the really hard problems are. In R. Myopoulos & J. Reiter (Eds.), *Proceedings of the 12th international joint conference on artificial intelligence*, Sydney, Australia (pp. 331–340). San Francisco, CA: Morgan Kaufmann.

Cornuéjols, A., & Sebag, M. (2008). A note on phase transitions and computational pitfalls of learning from sequences. *Journal of Intelligent Information Systems, 31*(2), 177–189.

Cortes, C., & Vapnik, V. N. (1995). Support-vector networks. *Machine Learning, 20*, 273–297.

De Raedt, L. (1997). Logical setting for concept-learning. *Artificial Intelligence, 95*, 187–202.

De Raedt, L. (1998). Attribute-value learning versus inductive logic programming: The missing links. In *Proceedings inductive logic programming, ILP, LNCS*, (Vol. 2446, pp. 1–8). London: Springer.

Demongeot, J., & Sené, S. (2008). Boundary conditions and phase transitions in neural networks. Simulation results. *Neural Networks, 21*(7), 962–970.

Dietterich, T., Lathrop, R., & Lozano-Perez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence, 89*(1–2), 31–71.

Donoho, D. L., & Tanner, J. (2005). Sparse nonnegative solution of underdetermined linear equations by linear programming. *Proceedings of the National Academy of Sciences, 102*(27), 9446–9451.

Engel, A., & Van den Broeck, C. (2001). *Statistical mechanics of learning*. Cambridge: Cambridge University Press.

Gaudel, R., Sebag, M., & Cornuéjols, A. (2007). A phase transition-based perspective on multiple instance kernels. In *Proceedings of international conference on inductive logic programming, ILP*, Corvallis, OR (pp. 112–121).

Gaudel, R., Sebag, M., & Cornuéjols, A. (2008). A phase transition-based perspective on multiple instance kernels. *Lecture notes in computer sciences*, (Vol. 4894, pp. 112–121).

Giordana, A., & Saitta, L. (2000). Phase transitions in relational learning. *Machine Learning, 41*(2), 17–251.

Haussler, D. (1999). Convolutional kernels on discrete structures. Tech. Rep., Computer Science Department, University of California at Santa Cruz.

Hogg, T., Huberman, B. A., & Williams, C. P. (Eds.). (1996). *Artificial intelligence: Special Issue on frontiers in problem solving: Phase transitions and complexity*, (Vol. 81(1–2)). Elsevier.

Kramer, S., Lavrac, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. In S. Dzeroski & N. Lavrac (Eds.), *Relational data mining*, (pp. 262–291). New York: Springer.

Maloberti, J., & Sebag, M. (2004). Fast theta-subsumption with constraint satisfaction algorithms. *Machine Learning Journal, 55*, 137–174.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence, 18*, 203–226.

Plotkin, G. (1970). A note on inductive generalization. In *Machine Intelligence*, (Vol. 5). Edinburgh University Press.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco, CA: Morgan Kaufmann.

Rückert, U., & De Raedt, L. (2008). An experimental evaluation of simplicity in rule learning. *Artificial Intelligence, 172*(1), 19–28.

Perceptron

►Online Learning

Piecewise Constant Models

►Regression Trees

Piecewise Linear Models

►Model Trees

Plan Recognition

►Inverse Reinforcement Learning

Policy Gradient Methods

Jan Peters[1], J. Andrew Bagnell[2]
[1]Max Planck Institute for Biological Cybernetics, Tuebingen, Baden-Wuertemberg, Germany
[2]Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Policy search

Definition

A policy gradient method is a ►reinforcement learning approach that directly optimizes a parametrized control policy by a variant of gradient descent. These methods belong to the class of ►policy search techniques that maximize the expected return of a policy in a fixed policy class, in contrast with traditional ►value function approximation approaches that derive policies from a value function. Policy gradient approaches have various advantages: they enable the straightforward incorporation of domain knowledge in policy parametrization and often an optimal policy is more compactly represented than the corresponding value function; many such methods guarantee to convergence to at least a locally optimal policy; the methods naturally handle continuous states and actions and often even imperfect state information. The counterveiling drawbacks include difficulties in off-policy settings, the potential for very slow convergence and high sample complexity, as well as identifying local optima that are not globally optimal.

Structure of the Learning System

Policy gradient methods center around a parametrized policy π_θ, also known as a *direct controller*, with parameters θ that define the selection of actions a given the state s. Such a policy may be either deterministic $a = \pi_\theta(s)$ or stochastic $a \sim \pi_\theta(a|s)$. This choice also affects the policy gradient approach (e.g., a deterministic policy requires a model-based formulation when used for likelihood ratio policy gradients), chooses how the exploration–exploitation dilemma is addressed (e.g., a stochastic policy tries new actions while a deterministic policy requires the perturbation of policy parameters or sufficient stochasticity in the system), and may affect the optimal solution (e.g., for a time-invariant or stationary policy, the optimal policy can be stochastic (Sutton, McAllester, Singh, & Mansour, 2000)). Frequently used policies include Gibbs distributions $\pi_\theta(a|s) = \exp(\phi(s,a)^T\theta)/\sum_b \exp(\phi(s,b)^T\theta)$ for discrete problems (Bagnell, 2004; Sutton et al., 2000) and, for continuous problems, Gaussian policies $\pi_\theta(a|s) = \mathcal{N}(\phi(s,a)^T\theta_1, \theta_2)$ with an exploration parameter θ_2 (Peters & Schaal, 2008; Williams, 1992).

Expected Return

Policy gradient methods seek to optimize the expected return of a policy π_θ,

$$J(\theta) = Z_\gamma E\left\{\sum_{k=0}^{H} \gamma^k r_k\right\},$$

where $\gamma \in [0,1]$ denotes a discount factor, Z_γ a normalization constant, and H the planning horizon. For finite H, we have an episodic reinforcement learning scenario where the truly optimal policy is non-stationary and the normalization does not matter. For an infinite horizon $H = \infty$, we choose the normalization to be $Z_\gamma \equiv (1-\gamma)$ for $\gamma < 1$ and $Z_1 \equiv \lim_{\gamma\to 1}(1-\gamma) = 1/H$ for ►average reward reinforcement learning problem where $\gamma = 1$.

Gradient Descent in Policy Space

Policy gradient methods follow an estimate of the gradient of the expected return

$$\theta_{k+1} = \theta_k + \alpha_k g(\theta_k)$$

where $g(\theta_k) \approx \nabla_\theta J(\theta)|_{\theta=\theta_k}$ is a gradient estimate at the parameters $\theta = \theta_k$ after update k with initial policy θ_0 and α_k denotes a learning rate. If the gradient estimator is unbiased, $\sum_{k=0}^{\infty} \alpha_k \to \infty$ while $\sum_{k=0}^{\infty} \alpha_k^2$ remains bounded, convergence to a local minimum can be guaranteed. In optimal control, model-based policy gradient methods have been used since the 1960s, see the classical textbooks by Jacobson &

Mayne (1970) and by Hasdorff (1976). While these are used machine learning community (e.g., differential dynamic programming with learned models), they may be numerically brittle and must rely on accurate, deterministic models. Hence, they may suffer significantly from optimization biases and are not generally applicable, particularly not in a model-free case nor in problems that include discrete elements. Several model-free alternatives can be found in the simulation optimization literature (Fu, 2006), including, for example, finite-difference gradients, likelihood ratio approaches, response-surface methods, and mean-valued, "weak" derivatives. The advantages and disadvantages of these different approaches are still a fiercely debated topic (Fu, 2006). In machine learning, the first two approaches have been dominating the field.

Finite Difference Gradients

The simplest policy gradient approaches with perhaps the most practical applications (see Peters & Schaal, 2008 for robotics application of this method) estimate the gradient by perturbing the policy parameters. For a current policy θ_k with expected return $J(\theta_k)$, this approach will create explorative policies $\hat{\theta}_i = \theta_k + \delta\theta_i$ with the approximated expected returns given by $J(\hat{\theta}_i) \approx J(\theta_k) + \delta\theta_i^T g$ where $g = \nabla_\theta J(\pi_\theta)|_{\theta=\theta_k}$. In this case, it suffices to determine the gradient by ▶linear regression, that is, we obtain

$$g = (\Delta\Theta^T \Delta\Theta)^{-1} \Delta\Theta^T \Delta J,$$

with parameter perturbations $\Delta\Theta = [\delta\theta_1, \ldots, \delta\theta_n]$ and the mean-subtracted rollout returns $\delta J_n = J(\hat{\theta}_i) - \overline{J(\theta_k)}$ form $\Delta J = [\delta J_1, \ldots, \delta J_n]$. The choice of the parameter perturbation largely determines the performance of the approach (Spall, 2003). Limitations particular to this approach include the following: the need for many exploratory samples; the sensitivity of the system with respect to each parameter may differ by orders of magnitude; small changes in a single parameter may render a system unstable; and stochasticity requires particular care in optimization (e.g., multiple samples, fixed random seeds, etc.) (see Glynn, 1990; Spall, 2003). This method is also referred to as the *naive Monte-Carlo policy gradient.*

Likelihood-Ratio Gradients

The likelihood-ratio method relies on the stochasticityll of either the policy for model-free approaches or the system in the model-based case, and, hence, requires no explicit exploration and may cope better with noise and the parameter perturbation sensitivity problems. Denoting a time-indexed sequence of states, actions, and rewards of the joint system composed of the policy and environment as a *path*, a parameter setting induces a path distribution $p_\theta(\tau)$ and rewards $R(\tau) = Z_\gamma \sum_{k=0}^{H} \gamma^k r_k$ along a path τ. Thus, you may write the gradient of the expected return as

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \int p_\theta(\tau) R(\tau) d\tau \\ &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) d\tau \\ &= E\{\nabla_\theta \log p_\theta(\tau) R(\tau)\}.\end{aligned}$$

If our system $p(s'|s,a)$ is Markovian, we may use $p_\theta(\tau) = p(s_0) \prod_{h=0}^{H} p(s_{k+1}|s_k, a_k) \pi_\theta(a_k|s_k)$ for a stochastic policy $a \sim \pi_\theta(a|s)$ to obtain the model-free policy gradient estimator known as episodic REINFORCE (Williams, 1992)

$$\nabla_\theta J(\theta) = Z_\gamma E\left\{\sum_{h=0}^{H} \gamma^k \nabla_\theta \log \pi_\theta(a_k|s_k) \sum_{k=h}^{H} \gamma^{k-h} r_k\right\},$$

and for the deterministic policy $a = \pi_\theta(s)$, the model-based policy gradient

$$\begin{aligned}\nabla_\theta J(\theta) = Z_\gamma E\Bigg\{\sum_{h=0}^{H} \gamma^k \big(\nabla_{a_k} \log p(s_{k+1}|s_k, a_k)^T \\ \nabla_\theta \pi_\theta(s_k)\big) \sum_{k=h}^{H} \gamma^{k-h} r_k\Bigg\}\end{aligned}$$

follows from $p_\theta(\tau) = p(s_0) \prod_{h=0}^{H} p(s_{k+1}|s_k, \pi_\theta(s_k))$. Note that all rewards preceeding an action may be omitted as the cancel out in expectation. Using a state–action value function $Q^{\pi_\theta}(s,a,h) = E\left\{\sum_{k=h}^{H} \gamma^{k-h} r_k \middle| s, a, \pi_\theta\right\}$ (see ▶Value Function Approximation), we can rewrite REINFORCE in its modern form

$$\begin{aligned}\nabla_\theta J(\theta) = Z_\gamma E\Bigg\{\sum_{h=0}^{H} \gamma^k \nabla_\theta \log \pi_\theta(a_k|s_k) \\ \left(Q^{\pi_\theta}(s,a,h) - b(s,h)\right)\Bigg\},\end{aligned}$$

known as the policy gradient theorem where the baseline $b(s,h)$ is an arbitrary function that may be used to reduce the variance, and $Q^{\pi_\theta}(s,a,h)$ represents the action–value function.

While likelihood-ratio gradients have been known since the late 1980s, they have recently experienced an upsurge of interest due to their demonstrated effectiveness in applications (see, for example, Peters & Schaal, 2008), progress toward variance reduction using optimal baselines (Lawrence, Cowan, & Russell, 2003), rigorous understanding of the relationships between value functions and policy gradients (Sutton et al., 2000), policy gradients in reproducing kernel Hilbert space (Bagnell, 2004) as well as faster, more robust convergence using natural policy gradients (Bagnell, 2004; Peters & Schaal, 2008).

Cross References

►Markov Decision Process
►Reinforcement Learning
►Value Function Approximation

Recommended Reading

Bagnell, J. A. (2004). *Learning decisions: Robustness, uncertainty, and approximation*. Doctoral dissertation, Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213.

Fu, M. C. (2006). *Handbook on operations research and management science: Simulation* (Vol. 13, pp. 575–616) (Chapter 19: Stochastic gradient estimation). ISBN 10: 0-444-51428-7, Elsevier.

Glynn, P. (1990). Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM, 33*(10), 75–84.

Hasdorff, L. (1976). Gradient optimization and nonlinear control. John Wiley & Sons.

Jacobson, D. & H., Mayne, D. Q. (1970). Differential Dynamic Programming. New York: American Elsevier Publishing Company, Inc.

Lawrence, G., Cowan, N., & Russell, S. (2003). Efficient gradient estimation for motor control learning. In *Proceedings of the international conference on uncertainty in artificial intelligence (UAI)*, Acapulco, Mexico.

Peters, J., & Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks, 21*(4), 682–97.

Spall, J. C. (2003). *Introduction to stochastic search and optimization: Estimation, simulation, and control*. Hoboken: Wiley.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, & K.-R. Mueller, (Eds.), *Advances in neural information processing systems (NIPS)*, Denver, CO. Cambridge: MIT Press.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*, 229–256.

Policy Search

►Markov Decision Processes
►Policy Gradient Methods

POMDPs

►Partially Observable Markov Decision Processes

POS Tagging

Walter Daelemans
CLIPS University of Antwerp, Antwerpen, Belgium

Synonyms

Grammatical tagging; Morphosyntactic disambiguation; Part of speech tagging; Tagging

Definition

Part-of-speech tagging (POS tagging) is a process in which each word in a text is assigned its appropriate morphosyntactic category (for example *noun-singular*, *verb-past*, *adjective*, *pronoun-personal*, and the like). It therefore provides information about both morphology (structure of words) and syntax (structure of sentences). This disambiguation process is determined both by constraints from the lexicon (what are the possible categories for a word?) and by constraints from the context in which the word occurs (which of the possible categories is the right one in this context?). For example, a word like *table* can be a noun-singular, but also a verb-present (as in *I table this motion*). This is lexical knowledge. It is the context of the word that should be used to decide which of the possible categories is the correct one. In a sentence like *Put it on the table,* the fact that *table* is preceded by the determiner *the*, is a good indication that it is used as a noun here. Systems that automatically assign parts of speech to words in text should take into account both lexical and contextual constraints, and they are typically found in implementations as a lookup module and a disambiguation module.

Motivation and Background

In most natural language processing (NLP) applications, POS tagging is one of the first steps to allow abstracting away from individual words. It is not to be confused with *lemmatization*, a process that reduces morphological variants of words to a canonical form (the citation form, for example, infinitive for verbs and singular for nouns). Whereas lemmatization allows abstraction over different forms of the same word, POS tagging abstracts over sets of different words that have the same function in a sentence. It should also not be confused with *tokenization*, a process that detects word forms in text, stripping off punctuation, handling abbreviations, and so on. For example, the string *don't* could be converted to *do not*. Normally, a POS tagging system would take tokenized text as input. More advanced tokenizers may even handle multiword items, for example treating *in order to* not as three separate words but as a single lexical item.

Applications. A POS tagger is the first disambiguation module in text analysis systems. In order to determine the syntactic structure of a sentence (and its semantics), we have to know the parts of speech of each word. In earlier approaches to syntactic analysis (parsing), POS tagging was part of the parsing process. However, individually trained and optimized POS taggers have increasingly become a separate module in shallow or deep syntactic analysis systems. By extension, POS tagging is also a foundational module in text mining applications ranging from information extraction and terminology/ontology extraction to summarization and question answering.

Apart from being one of the first modules in any text analysis system, POS tagging is also useful in linguistic studies (corpus linguistics) – for example for computing frequencies of disambiguated words and of superficial syntactic structures. In speech technology, knowing the part of speech of a word can help in speech synthesis (the verb "subJECT" is pronounced differently from the noun "SUBject"), and in speech recognition, POS taggers are used in some approaches to language modeling. In spelling and grammar checking, POS tagging plays a role in increasing the precision of such systems.

Part-of-speech tag sets. The inventory of POS tags can vary from tens to hundreds depending on the richness of morphology and syntax that is represented and on the inherent morphological complexity of a language. For English, the tag sets most used are those of the Penn Treebank (45 tags; Marcus, Santorini, & Marcinkiewicz, 1993), and the CLAWS C7 tag set (146 tags; Garside & Smith, 1997). Tag sets are most often developed in the context of the construction of annotated corpora. There have been efforts to standardize the construction of tag sets to increase translatability between different tag sets, such as Eagles. (http://www.ilc.cnr.it/EAGLES96/browse.html) and ISO/TC 37/SC 4. (http://www.tc37sc4.org/)

The following example shows both tag sets. By convention, a tagged word is represented by attaching the POS tag to it, separated by a slash.

Pierre/NNP Vinken/NNP ,/, 61/CD years/NNS old/JJ ,/, will/MD join/VB the/DT board/NN as/IN a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD ./. [Penn Treebank]

Pierre/NP1 Vinken/NP1 ,/, 61/MC years/NNT2 old/JJ ,/, will/VM join/VVI the/AT Board/NN1 as/II a/AT1 nonexecutive/JJ director/NN1 Nov./NPM1 29/MC ./. [CLAWS C7]

As can be seen, the tag sets differ in level of detail. For example, NNT2 in the C7 tag set indicates a plural temporal noun (as a specialization of the word class noun), whereas the Penn Treebank tag set only specializes to plural noun (NNS).

Like most tasks in NLP, POS tagging is a disambiguation task, and both linguistic knowledge-based handcrafting methods and corpus-based learning methods have been proposed for this task. We will restrict our discussion here to the statistical and machine learning approaches to the problem, which have become mainstream because of the availability of large POS tagged corpora and because of better accuracy in general than handcrafted systems. A state of the art system using a knowledge-based approach is described in Karlsson, Voutilainen, Heikkilä, and Anttila (1995).

A decade old now, but still a complete and informative book-length introduction to the field of POS tagging is van Halteren (1999). It discusses many important issues that are not covered in this article (performance evaluation, history, handcrafting approaches, tag set development issues, handling unknown words, and more.). A more recent introductory overview is Chap. 5 in Jurafsky and Martin (2008).

Statistical and Machine Learning Approaches to Tagging

In the late 1970s, statistical approaches based on n-gram probabilities (probabilities that sequences of n tags occur in a corpus) computed on frequencies in tagged corpora have already been proposed by the UCREL team at the University of Lancaster (Garside & Smith, 1997). These early models lacked a precise mathematical framework and a principled solution to working with zero- or low probability frequencies. It was realized that Hidden Markov Models (HMM) in use in speech recognition were applicable to the tagging problem as well.

HMMs

HMMs are probabilistic finite state automata that are flexible enough to combine n-gram information with other relevant information to a limited extent. They allow supervised learning by computing the probabilities of n-grams from tagged corpora, and unsupervised learning using the Baum-Welch algorithm. Finding the most probable tag sequence given a sequence of words (decoding) is done using the Viterbi search. In combination with smoothing methods for low-frequency events and special solutions for handling unknown words, this approach results in a state-of-the-art tagging performance. A good implementation is TnT (Trigrams'n Tags, Brants, 2000).

Transformation-Based Error-Driven Learning (Brill-Tagging)

Transformation-based learning is an eager learning method in which the learner extracts a series of rules, each of which transforms a tag into another tag given a specific context. Learning starts with an initial annotation (e.g., tag each word in a text by the POS tag it is most frequently associated with in a training corpus), and compares this annotation with a gold standard annotation (annotated by humans). Discrepancies trigger the generation of rules (constrained by templates), and in each cycle, the best rule is chosen. The best rule is the one that most often leads to a correct transformation in the whole training corpus (Brill, 1995a). An unsupervised learning variant (using a lexicon with word-tag probabilities) is described in Brill (1995b). Fully unsupervised POS tagging can also be achieved using distributional clustering techniques, as pioneered by Schütze (1995). However, these methods are hard to evaluate and compare to supervised approaches. The best way to evaluate them is indirectly, in an application-oriented way, as in Ushioda (1996).

Other Supervised Learning Methods

As a supervised learning task, POS tagging has been handled mostly as in a *sliding window* representation. Instances are created by making each word in each sentence a focus feature of an instance, and adding the left and right context as additional features. The class to be predicted is the POS tag of the focus word. Instead of using the words themselves as features, information about them can be used as features as well (e.g., capitalized or not, hyphenated or not, the POS tag of the word for left context words as predicted by the tagger previously, a symbol representing the possible lexical categories of the focus word and right context words, first and last letters of the word in each position, and so on.).

The following table lists the structure of instance representations for part of the sentence shown earlier. In this case the words themselves are feature values, but most often other derived features would replace these because of sparseness problems.

		Focus			*Class*
=	=	Pierre	Vinken	,	NNP
=	Pierre	Vinken	,	61	NNP
Pierre	Vinken	,	61	years	,
Vinken	,	61	years	old	CD

Most classification-based, supervised machine learning methods can be, and have been applied to this problem, including decision tree learning (Schmid, 1994b), memory-based learning (Daelemans, Zavrel, Berck, & Gillis, 1996), maximum entropy models (Ratnaparkhi, 1996), neural networks (Schmid, 1994a), ensemble methods (van Halteren et al., 2001), and many others. All these methods seem to converge to a 96–97% accuracy rate on the Wall Street Journal corpus using the Penn Treebank tag set. In a systematic comparison of some of the methods listed here, van Halteren et al. (2001) found that TnT outperforms maximum entropy and memory-based learning methods, which in turn outperform Brill tagging. Non-propositional supervised learning methods have been applied to the task as well (Cussens, 1997) with grammatical structure

as background knowledge with similar results. The best results reported on the WSJ corpus so far is bidirectional perceptron learning (Shen, Satta, & Joshi, 2007) with a 97.33% accuracy.

Because of these high scores, POS tagging (at least for English) is considered by many a solved problem. However, as for most machine-learning based NLP systems, domain adaptation is still a serious problem for POS tagging. A tagger trained to high accuracy on newspaper language will fail miserably on other types of text, such as medical language.

Cross References

►Classification
►Clustering
►Decision Trees
►ILP
►Information Extraction
►Lazy Learning
►Maxent Models
►Text Categorization
►Text Mining

Recommended Reading

Brants, T. (2000). TnT – A statistical part-of-speech tagger. In *Proceedings of the sixth applied natural language processing conference ANLP-2000*. Seattle, WA.

Brill, E. (1995a). Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. *Computational Linguistics*, *21*(4), 543–565.

Brill, E. (1995b). Unsupervised learning of disambiguation rules for part of speech tagging. In *Proceedings of the third workshop on very large corpora* (pp. 1–13). Ohio State University, Ohio.

Cussens, J. (1997). Part-of-speech tagging using progol. In N. Lavrac, & S. Dzeroski (Eds.), *Proceedings of the seventh international workshop on inductive logic programming*, *Lecture Notes in Computer Science* (Vol. 1297 pp. 93–108). London: Springer.

Daelemans, W., Zavrel, J., Berck, P., & Gillis, S. (1996). MBT: A memory-based part of speech tagger generator. In *Proceedings of the fourth workshop on very large corpora* (pp. 14–27). Copenhagen, Denmark

Garside, R., & Smith, N. (1997). A hybrid grammatical tagger: CLAWS4. In R. Garside, G. Leech, & A. McEnery (Eds.), *Corpus annotation: Linguistic information from computer text corpora* (pp. 102–121). London: Longman.

Jurafsky, D., & Martin, J. (2008). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Karlsson, F., Voutilainen, A., Heikkilä, J., & Anttila, A. (1995). *Constraint grammar. A language-independent system for parsing unrestricted text* (p. 430). Berlin and New York: Mouton de Gruyter.

Marcus, M., Santorini, B., & Marcinkiewicz, M. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, *19*(2), 313–330.

Ratnaparkhi, A. (1996). A maximum entropy part of speech tagger. In *Proceedings of the ACL-SIGDAT conference on empirical methods in natural language processing* (pp. 17–18). Philadelphia, PA.

Schmid, H. (1994a). Part-of-speech tagging with neural networks. In *Proceedings of COLING-94* (pp. 172–176). Kyoto, Japan.

Schmid, H. (1994b). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the international conference on new methods in language processing (NeMLaP)*, (pp. 44–49). Manchester, UK.

Schutze, H. (1995). Distributional part-of-speech tagging. In *Proceedings of EACL 7* (pp. 141–148). Dublin, Ireland.

Shen, L., Satta, G., & Joshi, A. (2007). Guided learning for bidirectional sequence classification. In *Proceedings of the 45th annual meetings of the association of computational linguistics (ACL 2007)* (pp. 760–767). Prague, Czech Republic.

Ushioda, A. (1996). Hierarchical clustering of words and applications to NLP tasks. In *Proceedings of the fourth workshop on very large corpora* (pp. 28–41). Somerset, NJ.

van Halteren, H. (Ed.). (1999). *Syntactic wordclass tagging*. Boston: Kluwer Academic Publishers.

van Halteren, H. Zavrel, J., & Daelemans, W. (2001) Improving accuracy in NLP through combination of machine learning systems. *Computational Linguistics*, *27*(2), 199–229.

Positive Definite

►Positive Semidefinite

Positive Predictive Value

►Precision

Positive Semidefinite

Synonyms

Positive definite

Definition

A symmetric $m \times m$ matrix K satisfying $\forall x \in c^m : x^* K x \geq 0$ is called positive semidefinite. If the equality only holds for $x = \vec{0}$ the matrix is positive definite.

A function $k : X \times X \to c$, $X \neq \varnothing$, is positive (semi-) definite if for all $m \in n$ and all $x_1, \ldots, x_m \in X$ the $m \times m$ matrix $\vec{K}$ with elements $K_{ij} := k(x_i, x_j)$ is positive (semi-) definite.

Sometimes the term strictly positive definite is used instead of positive definite, and positive definite refers then to positive semidefiniteness.

Posterior

►Posterior Probability

Posterior Probability

Geoffrey I. Webb
Monash University

Synonyms
Posterior

Definition
In Bayesian inference, a *posterior probability* of a value x of a random variable X given a context a value y of a random variable Y, $P(X = x \mid Y = y)$, is the probability of X assuming the value x in the context of $Y = y$. It contrasts with the ►prior probability, $P(X = x)$, the probability of X assuming the value x in the absence of additional information.

For example, it may be that the prevalence of a particular form of cancer, *exoma*, in the population is 0.1%, so the prior probability of exoma, P(exoma = true), is 0.001. However, assume 50% of people who have skin discolorations of greater than 1 cm width (sd>1cm) have exoma. It follows that the posterior probability of exoma given sd>1cm, P(exoma = true | sd>1cm = true), is 0.500.

Cross References
►Bayesian Methods

Post-Pruning

Definition
Post-pruning is a ►Pruning mechanism that first learns a possibly ►Overfitting hypothesis and then tries to simplify it in a separate learning phase.

Cross References
►Overfitting
►Pre-Pruning
►Pruning

Postsynaptic Neuron

The neuron that receives signals via a synaptic connection. A chemical synaptic connection between two neurons allows to transmit signals from a presynaptic neuron to a postsynaptic neuron.

Precision

Kai Ming Ting
Monash University, Victoria, Australia

Synonyms
Positive predictive value

Definition
Precision is defined as a ratio of true positives (TP) and the total number of positives predicted by a model. This is defined with reference to a special case of the ►confusion matrix, with two classes; one designated the *positive* class, and the other the *negative* class, as indicated in Table 1.

Precision can then be defined in terms of true positives and false positives (FP) as follows.

Precision = TP/(TP + FP)

Cross References
►Precision and Recall

Precision. Table 1 The outcomes of classification into positive and negative classes

		Assigned Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Precision and Recall

Kai Ming Ting
Monash University, Vic, Australia

Definition

►Precision and recall are the measures used in the information retrieval domain to measure how well an information retrieval system retrieves the relevant documents requested by a user. The measures are defined as follows:

Precision = Total number of documents retrieved that are relevant/Total number of documents that are retrieved.

Recall = Total number of documents retrieved that are relevant/Total number of relevant documents in the database.

We can use the same terminology used in a ►confusion matrix to define these two measures. Let relevant documents be positive examples and irrelevant documents, negative examples. The two measures can be redefined with reference to a special case of the confusion matrix, with two classes, one designated the *positive* class, and the other the *negative* class, as indicated in Table 1.

Recall = True positives/Total number of actual positives = TP/(TP + FN)

Precision = True positives/Total number of positives predicted = TP/(TP + FP)

Instead of two measures, they are often combined to provide a single measure of retrieval performance called the ►F-measure as follows:

F-measure = 2 * Recall * Precision/(Recall + Precision)

Cross References

►Confusion Matrix

Precision and Recall. Table 1 The outcomes of classification into positive and negative classes

		Assigned Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Predicate

A *predicate* or predicate symbol is used in logic to denote properties and relationships. Formally, if P is a predicate with arity n, and $t_1, \ldots, t_n$ is a sequence of n terms (i.e., constants, variables, or compound terms built from function symbols), then $P(t_1, \ldots, t_n)$ is an atomic formula or *atom*. Such an atom represents a statement that can be either true or false. Using logical connectives, atoms can be combined to build well-formed formulae in ►first-order logic or ►clauses in ►logic programs.

Cross References

►Clause
►First-Order Logic
►Logic Program

Predicate Calculus

►First-Order Logic

Predicate Invention

Definition

Predicate invention is used in ►inductive logic programming to refer to the automatic introduction of new relations or predicates in the hypothesis language. Inventing relevant new predicates is one of the hardest tasks in machine learning, because there are so many possible ways to introduce such predicates and because it is hard to judge their quality. As an example, consider a situation where in the predicates `fatherof` and `motherof` are known. Then it would make sense to introduce a new predicate that is true whenever `fatherof` or `motherof` is true. The new predicate that would be introduced this way corresponds to the `parentof` predicate. Predicate invention has been introduced in the context of inverse resolution.

Cross References
►Inductive Logic Programming
►Logic of Generality

Predicate Logic

►First-Order Logic

Prior Probabilities

►Bayesian Nonparametric Models

Prior Probability

Geoffrey I. Webb
Monash University

Synonyms
Prior

Definition
In Bayesian inference, a *prior probability* of a value x of a random variable X, $P(X = x)$, is the probability of X assuming the value x in the absence of (or before obtaining) any additional information. It contrasts with the ►posterior probability, $P(X = x \mid Y = y)$, the probability of X assuming the value x in the context of $Y = y$.

For example, it may be that the prevalence of a particular form of cancer, *exoma*, in the population is 0.1%, so the prior probability of exoma, P(exoma = true), is 0.001. However, assume 50% of people who have skin discolorations of greater than 1 cm width (sd>1cm) have exoma. It follows that the posterior probability of exoma given sd>1cm, P(exoma = true | sd>1cm = true), is 0.500.

Cross References
►Bayesian Methods

Prediction with Expert Advice

►Online Learning

Predictive Software Models

►Predictive Techniques in Software Engineering

Predictive Techniques in Software Engineering

Jelber Sayyad Shirabad
University of Ottawa
Ottawa, ON, Canada

Synonyms
Predictive software models

Introduction
Software engineering (SE) is a knowledge- and decision-intensive activity. From the initial stages of the software life cycle (i.e., requirement analysis), to the later stage of testing the system, and finally maintaining the software through its operational life, decisions need to be made which impact both its success and failure. For instance, during project planning one needs to be able to forecast or predict the required resources to build the system. At the later stages such as testing or maintenance it is desirable to know which parts of the system may be impacted by a change, or are more risky or will require more intensive testing.

The process of developing software can potentially create a large amount of data and domain knowledge. The nature of the data, of course, depends on the phase in which the data were generated. During the requirement analysis, this data most times is manifested in the form of documentations. As the process moves forward, other types of artifacts such as code and test cases are generated. However, what, when, how accurately, and how much is recorded varies from one organization to the next. More mature organizations have a tendency to maintain larger amount of data about the software systems they develop.

The data generated as part of the software engineering process captures a wide range of latent knowledge about the system. Having such a source of information, the question one needs to ask is that whether there is any technology that can leverage this potentially vast amount of data to:

- Better understand a system
- Make more informative decisions as needed through the life of an existing system
- Apply lessons learned from building other systems to the creation of a new system

As this chapter will show, machine learning (ML), which provides us with a host of algorithms and techniques to learn from data, is such a technology. In preparing this entry we have drawn from over two decades of research in applying ML to various software engineering problems. The number of potential uses of ML in SE is practically enormous and the list of applications is expanding over time. The focus of this chapter is a subset of these applications, namely the ones that aim to create models for the purpose of making a prediction regarding some aspect of a software system. One could dedicate a separate article for some of these prediction tasks, as there is a large body of research covering different aspects of interest, such as algorithms, estimation methods, features used, and the like. However, due to space constraints, we will only mention a few representative research examples. The more general topic of the application of ML in SE can be studied from different points of view. A good discussion of many such aspects and applications can be found in Zhang and Tsai (2003).

Traditionally, regression-based techniques have been used in software engineering for building predictive models. However, this requires making a decision as to what kind of regression method should be used (e.g., linear or quadratic), or alternatively what kind of curve should be fit to the data. This means that the general shape of the function is determined first, and then the model is built. Some researcher, have used ML as a way to delegate such decisions to the algorithm. In other words, it is the algorithm that would produce the best fit to the data. Some of the most common replacements in the case of regression problems have been neural networks (NN) and genetic programming (GP). However, obviously the use of such methods still requires other types of decisions, such as the topology of the network, the number of generations, or the probability of mutations to be made by humans. Sometimes, a combination of different methods such as genetic algorithms and neural networks are used, where one method explores possible parameters for the actual method used to build the model.

Software engineering-related datasets, similar to many other real world datasets, are known to contain noise. Another justification for the use of ML in software engineering applications is that it provides algorithms that are less sensitive to noise.

The Process of Applying ML to SE

To apply ML to SE, similar to other applications, one needs to follow certain steps, which include:

Understanding the problem. This is an essential step that heavily influences the decisions to follow. Examples of typical problems in the software engineering domain are the need to be able to estimate the cost or effort involved in developing a software, or to be able to characterize the quality of a software system, or to be able to predict what modules in a system are more likely to have a defect.

Casting the original problem as a learning problem. To use ML technology, one needs to decide on how to formulate the problem as a learning task. For instance, the problem of finding modules that are likely to be faulty can be cast as a classification problem, (e.g., is the module faulty or not) or a numeric prediction problem (e.g., what the estimated fault density of a module is). This mapping is not always straightforward, and may require further refinement of the original problem statement or breaking down the original problem into sub-problems, for some of them ML may provide an appropriate solution.

Collection of data and relevant background knowledge. Once the ML problem for a particular SE application is identified, one needs to collect the necessary data and background knowledge in support of the learning task. In many SE applications data is much more abundant or easier to collect than the domain theory or background knowledge relevant to a particular application. For instance, collecting data regarding faults discovered in a software system and changes applied to the source to correct a fault is a common practice in software projects. On the other hand, there is no comprehensive and agreed upon domain theory describing software systems. Having said that, in the case of some

applications, if we limit ourselves to incomplete background knowledge, then it can be automatically generated by choosing a subset that is considered to be relevant. For instance, in Cohen and Devanbu (1999), the authors apply inductive logic programming to the task of predicting faulty modules in a software system. They describe the software system in terms of cohesion and coupling-based relations between classes, which are generated by parsing the source code.

Data preprocessing and encoding. Preprocessing the data includes activities such as reducing the noise, selecting appropriate subsets of the collected data, and determining a proper subset of features that describe the concept to be learned. This cleaner data will be input to a specific algorithm and implementation. Therefore, the data and background knowledge, if any, may need to be described and formatted in a manner that complies with the requirements of the algorithm used.

Applying machine learning and evaluating the results. Running a specific ML algorithm is fairly straightforward. However, one needs to measure the goodness of what is learned. For instance, in the case of classification problems, models are frequently assessed in terms of their accuracy by using methods such as hold-out and cross-validation. In case of numeric prediction, other standard measures such as mean magnitude of relative error (MMRE) are commonly used. Additionally, software engineering researchers have sometimes adopted other measures for certain applications. For instance PRED(x), which is percentage of the examples (or samples) with magnitude of relative error (MRE) $\leq x$. According to Pfleeger and Atlee (2003), most managers use PRED(25) to assess cost, effort, and schedule models, and consider the model to function well if the value of PRED(25) is greater than 75%. As for MMRE, a value of less than 25% is considered to be good; however, other researchers, such as Boehm, would recommend a value of 10% or less. Assessing the usefulness of what is learned sometimes requires feedback from domain experts or from end users. If what is learned is determined to be inadequate, one may need to either retry this step by adjusting the parameters of the algorithms used, or reconsider the decisions made in earlier stages and proceed accordingly.

Field testing and deployment. Once what is learned is assessed to be of value, it needs to actually be used by the intended users (e.g., project managers and software engineers). Unfortunately, despite the very large body of research in software engineering in general and use of ML in specific applications in SE, the number of articles discussing the actual use and impact of the research in industry is relatively very small. Very often, the reason for this is the lack of desire to share what the industry considers to be confidential information. However, there are numerous research articles that are based on industrial data, which is an indication of the practical benefits of ML in real-world SE.

Applications of Predictive Models in SE

The development of predictive models is probably the most common application of ML in software engineering. This observation is consistent with findings of previous research (Zhang & Tsai, 2003). In this section, we mention some of the predictive models one can learn from software engineering data. Our goal is to provide examples of both well established and newer applications. It should be noted that the terminology used by researchers in the field is not always consistent. As such, one may argue that some of these examples belong to more than one category. For instance, in Fenton and Neil (1999) the authors consider predicting faults as a way of estimating software quality and maintenance effort. The paper could potentially belong to any of the categories of fault, quality, or maintenance effort prediction.

Software size prediction

Software size estimation is the process of predicting the size of a software system. As software size is usually an input to models that estimate project cost schedule and planning, an accurate estimation of software size is essential to proper estimation of these dependent factors. Software size can be measured in different ways, most common of which is the number of lines of code (LOC); however, other alternatives, such as function points, which are primarily for effort estimation, also provide means to convert the measure to LOC. There are different methods for software sizing, one of which is the component-based method (CBM). In a study to validate the CBM method, Dolado, (2000) compared models generated by multiple ▶linear regression (MLR)

with the ones obtained by neural networks and genetic programming. He concluded that both NN- and GP-based models perform as well or better than the MLR models. One of the cited benefits of NN was its ability to capture non-linear relations, which is one of the weaknesses of MLR, while GP was able to generate models that were interpretable. Regolin, de Souza, Pozo, and Vergilio (2003) also used NN- and GP-based models to predict software size in terms of LOC. They use both function points and number of components metrics for this task. Pendharkar (2004) uses decision tree regression to predict the size of OO components. The total size of the system can be calculated after the size of its components is determined.

Software quality prediction

The ISO 9126 quality standard decomposes quality to functionality, reliability, efficiency, usability, maintainability, and portability factors. Other models such as McCall's, also define quality in terms of factors that are themselves composed of quality criteria. These quality criteria are further associated with measurable attributes called quality metrics, for instance fault or change counts (Fenton & Pfleeger, 1998) However, as stated in Fenton and Pfleeger (1998), many software engineers have a narrower view of quality as the lack of software defects. A de facto standard for software quality is fault density. Consequently, it is not surprising to see that in many published articles the problem of predicting the quality of a system is formulated as prediction of faults. To that end, there has been a large body of work over the years that has applied various ML techniques to build models to assess the quality of a system. For instance, Evett and Khoshgoftar (1998) used genetic programming to build models that predict the number of faults expected in each module. Neural networks have appeared in a number of software quality modeling applications such as Khoshgoftaar, Allen, Hudepohl, and Aud (1997), which applied the technique to a large industrial system to classify modules as fault-prone or not fault-prone, or Quah and Thwin (2003) who used object-oriented design metrics as features in developing the model. In El Emam, Benlarbi, Goel, and Rai (2001) the authors developed fault prediction models for the purpose of identifying high-risk modules. In this study, the authors investigated the effect of various parameter settings on the accuracy of these models. The models were developed using data from a large real-time system. More recently, Xing, Guo, and Lyu (2005) used SVMs and Seliya and Khoshgoftaar (2007) used an EM semi-supervised learning algorithm to develop software quality models. Both these works cite the ability of these algorithms to generate models with good performance in the presence of a small amount of labeled data.

Software Cost Prediction

Software cost prediction typically refers to the process of estimating the amount of effort needed to develop a software system. As this definition suggests, cost and effort estimations are often used interchangeably. Various kinds of cost estimations are needed throughout the software life cycle. Early estimation allows one to determine the feasibility of a project. More detailed estimation allows managers to better plan for the project. As there is less information available in the early stages of the project, early predictions have a tendency to be the least accurate. Software cost and effort estimation models are among some of the oldest software process prediction models. There are different methods of estimating costs including:

(1) Expert opinion; (2) analogy based on similarity to other projects; (3) decomposition of the project in terms of components to deliver or tasks to accomplish, and to generate a total estimate from the estimates of the cost of individual components or activities; and (4) the use of estimation models (Fenton & Pfleeger, 1998).

In general, organization-specific cost estimation datasets tend to be small, as many organizations deal with a limited number of projects and do not systematically collect process level data, including the actual time and effort expenditure for completion of a project. As cost estimation models are numeric predictors, many of the original modeling techniques were based on regression methods.

The study in Briand, El Emam, Surmann, and Wieczorek (1999) aims to identify methods that generate more accurate cost models, as well as to investigate the effects of the use of organization-specific versus multi-organization datasets. The authors compared the accuracy of models generated by using ordinary least squares regression, stepwise ANOVA, CART, and analogy. The measures used were MMRE, median of MRE (MdMRE), and PRED(25). While their results did not

show a statistical difference between models obtained from these methods, they suggest that CART models are of particular interest due to their simplicity of use and interpretation.

Shepperd and Schofield (1997) describes the use of analogies for effort prediction. In this method, projects are characterized in terms of attributes such as the number of interfaces, the development method, or the size of the functional requirements document. The prediction for a specific project is made based on the characteristics of projects most similar to it. The similarity measure used in Shepperd and Schofield (1997) is Euclidean distance in n-dimensional space of project features. The proposed method was validated on nine different industrial datasets, covering a total of 275 projects. In all cases, the analogy-based method outperforms algorithmic models based upon stepwise regression when measured in terms of MMRE. When results are compared using PRED(25) the analogy-based method generates more accurate models in seven out of nine datasets. Decision tree and neural network-based models are also used in a number of studies on effort estimation models.

In a more recent paper, (Oliveira, 2006), a comparative study of support vector regression (SVR), radial basis function ▶neural networks (RBFNs), and ▶linear regression-based models for estimation of a software project effort is presented. Both linear as well as RBF kernels were used in the construction of SVR models. Experiments using a dataset of software projects from NASA showed that SVR significantly outperforms RBFNs and linear regression in this task.

Software Defect Prediction

In research literature one comes across different definitions for what constitutes a defect: fault and failure. According to Fenton and Pfleeger (1998) a fault is a mistake in some software product due to a human error. Failure, on the other hand, is the departure of the system from its required behavior. Very often, defects refer to faults and failures collectively. In their study of defect prediction models, Fenton and Neil observed that, depending on the study, defect count could refer to a post-release defect, the total number of known defects, or defects that are discovered after some arbitrary point in the life cycle. Additionally, they note that defect rate, defect density, and failure rate are used almost interchangeably in the literature (Fenton & Neil, 1999). The lack of an agreed-upon definition for such a fundamental measure makes comparison of the models or published results in the literature difficult. Two major reasons cited in research literature for developing defect detection models are assessing software quality and focusing testing or other needed resources on modules that are more likely to be defective. As a result, we frequently find ourselves in a situation where a model could be considered both a quality prediction model and a defect prediction model. Therefore, most of the publications we have mentioned under software quality prediction could also be referred to in this subsection. Fenton and Neil suggest using Bayesian Belief Networks as an alternative to other existing methods (Fenton & Neil, 1999).

Software Reliability Prediction

The ANSI Software Reliability Standard defines software reliability as:

- "the probability of failure-free operation of a computer program for a specified time in a specified environment."

Software reliability is an important attribute of software quality. There are a number of publications on the use of various neural network-based reliability prediction models, including Sitte (1999) where NN-based software reliability growth models are compared with models obtained through recalibration of parametric models. Results show that neural networks are not only much simpler to use than the recalibration method, but that they are equal or better trend predictors. In Pai and Hong (2006) the authors use SVMs to predict software reliability. They use simulated annealing to select the parameters of the SVM model. Results show that an SVM-based model with simulated annealing performs better than existing Bayesian models.

Software Reusability Prediction

The use of existing software artifacts or software knowledge is known as software reuse. The aim of software reuse is to increase the productivity of software developers, and increase the quality of end product,

both of which contribute to overall reduction in software development costs. While the importance of software reuse was recognized as early as 1968 by Douglas McIlroy, applications of ML in predicting reusable components are relatively few and far between. The typical approach is to label the reusable piece of code (i.e., a module or a class) as one of reusable or non-reusable, and to then use software metrics to describe the example of interest. An early work by Esteva (1990) used ID3 to classify Pascal modules from different application domains as either reusable or not-reusable. These modules contained different number of procedures. Later work in Mao, Sahraoui, and Lounis (1998) uses models built using C4.5 as a means to verify three hypothesis of correlation between reusability and the quantitative attributes of a piece of software: inheritance, coupling, and complexity. For each hypothesis, a set of relevant metrics (e.g., complexity metrics for a hypothesis on the relation between complexity and reuse) is used to describe examples. Each example is labeled as one of four classes of reusability, ranging from "totally reusable" to "not reusable at all." If the learned model performs well then this is interpreted as the existence of a hypothesized relation between reuse and one of the abovementioned quantitative attributes.

Other Applications

In this section, we discuss some of the more recent uses of ML techniques in building predictive models for software engineering applications that do not fall into one the above widely researched areas.

In Padberg, Ragg, and Schoknecht (2004) models are learned to predict the defect content of documents after software inspection. Being able to estimate how many defects are in a software document (e.g., specifications, designs) after the inspection, allows managers to decide whether to re-inspect the document to find more defects or pass it on to the next development step. To capture the non-linear relation between the inspection process metrics, such as total number of defects found by the inspection team and the number of defects in the document, the authors train a neural network. They conclude that these models yield much more accurate estimates than standard estimation methods such as capture-recapture and detection profile.

Predicting the stability of object-oriented software, defined as the ease by which a software system or component can be changed while maintaining its design, is the subject of research in Grosser, Sahraoui, and Valtchev (2002). More specifically, stability is defined as preservation of the class interfaces through evolution of the software. To accomplish the above task, the authors use Cased-Base Reasoning. A class is considered stable if its public interface in revision J is included in revision $J + 1$. Each program class or case is represented by structural software metrics, which belong to one of the four categories of coupling, cohesion, inheritance, and complexity.

Models that predict which defects will be escalated are developed in Ling, Sheng, Bruckhaus, and Madhavji (2006). Escalated defects are the ones that were not addressed prior to release of the software due to factors such as deadlines and limited resources. However, after the release, these defects are escalated by the customer and must be immediately resolved by the vendor at a very high cost. Therefore, the ability to predict the risk of escalation for existing defect reports will prevent many escalations, and result in large savings for the vendor. The authors in this paper show how the problem of maximizing net profit (the difference in cost of following predictions made by the escalation prediction model versus the existing alternative policy) can be converted to cost-sensitive learning. The assumption here is that net profit can be represented as a linear combination of true positive, false positive, true negative, and false negative prediction counts, as is done for cost-sensitive learning that attempts to minimize the weighted cost of the abovementioned four factors. The results of the experiments performed by the authors show that an improved version of the CSTree algorithm can produce comprehensible models that generate a large positive unit net profit.

Most predictive models developed for software engineering applications, including the ones cited in this article, make prediction regarding a single entity – for instance, whether a module is defective, how much effort is needed to develop a system, is a piece of code reusable, and so on. Sayyad Shirabad, Lethbridge, and Matwin (2007) introduced the notion of relevance relations among multiple entities in software systems. As an example of such relations, the authors applied historic

problem report and software change data to learned models for the Co-update relation among files in a large industrial telecom system. These models predict whether changing one source file may require a change in another file. Different sets of attributes, including syntax-based software metrics as well as textual attributes such as source file comments and problem reports, are used to describe examples of the Co-update relation. The C5.0 decision tree induction algorithm was used to learn these predictive models. The authors concluded that text-based attributes outperform syntactic attributes in this model-building task. The best results are obtained for text-based attributes extracted from problem reports. Additionally, when these attributes are combined with syntactic attributes, the resulting models perform slightly better.

Future Directions

As we mentioned earlier due to its decision-intensive nature, there is potential for learning a large number of predictive models for software engineering tasks. A very rich area of research for future applications of predictive models in software engineering is in *Autonomic Computing*. Autonomic computing systems, as was put forward in Ganek and Corbi (2003), should be:

- *Self-configuring*: able to adapt to changes in the system in a dynamic fashion.
- *Self-optimizing*: able to improve performance and maximize resource allocation and utilization to meet end users' needs while minimizing human intervention.
- *Self-healing*: able to recover from mistakes by detecting improper operations proactively or reactively and then initiate actions to remedy the problem without disrupting system applications.
- *Self-protecting*: able to anticipate and take actions against intrusive behaviors as they occur, so as to make the systems less vulnerable to unauthorized access.

Execution of actions in support of the capabilities mentioned above follows the detection of a triggering change of state in the environment. In some scenarios, this may entail a prediction about the current state of the system; in other scenarios, the prediction may be about the future state of the system. In a two-state scenario, the system needs to know whether it is in a normal or abnormal (undesired) state. Examples of undesired states are *needs optimization* or *needs healing*. The detection of the state of a system can be cast as a classification problem. The decision as to what attributes should be used to represent each example of a normal or an abnormal state depends on the specific prediction model that we would like to build and on the monitoring capabilities of the system. Selecting the best attributes among a set of potential attributes will require empirical analysis. However, the process can be further aided by:

- *Expert knowledge*: Based on their past experience, hardware and software experts typically have evidence or reasons to believe that some attributes are better indicators of desired or undesired states of the system.
- *Documentation*: System specification and other documents sometimes include the range of acceptable values for certain parameters of the system. These parameters could be used as attributes.
- *Feature selection*: This aims to find a subset of available features or attributes that result in improving a predefined measure of goodness, such as the accuracy of the model. Reducing the number of features may also result in a simpler model. One of the benefits of such simpler models is the higher prediction speed, which is essential for timely responses by the autonomic system to changes in the environment.

Obviously, given enough examples of different system states, one can build multi-class models, which can make finer predictions regarding the state of the system.

In the context of autonomic computing, besides classification models, numeric predictors can also be used for resource estimation (e.g., what is the appropriate database cache size considering the current state of the system). Furthermore, an autonomic system can leverage the ability to predict the future value of a variable of interest, such as the use of a particular resource based on its past values. This can be accomplished through ▶time series predictions. Although researchers have used neural networks and support vector machines for time series prediction in various domains, we are not aware of an example of the usage of such algorithms in autonomic computing.

Recommended Reading

Briand, L., El Emam, K., Surmann, D., & Wieczorek, I. (1999). An assessment and comparison of common software cost estimation modeling techniques. In *Proceedings of 21st international conference on software engineering* (pp. 313–322).

Cohen, W., & Devanbu, P. (1999). Automatically exploring hypotheses about fault prediction: A comparative study of inductive logic programming methods. *International Journal of Software Engineering and Knowledge Engineering*, *9*(5), 519–546.

Dolado, J. J. (2000). A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, *26*(10), 1006–1021.

El Emam, K., Benlarbi, S., Goel, N., & Rai, S. (2001). Comparing case-based reasoning classifiers for predicting high risk software components. *Journal of Systems and Software*, *55*(3), 301–320.

Esteva, J. C. (1990). Learning to recognize reusable software modules using an inductive classification system. In *Proceedings of the fifth Jerusalem conference on information technology* (pp. 278–285).

Evett, M., & Khoshgoftar, T. (1998). GP-based software quality prediction. In *Proceedings of the third annual conference on genetic programming* (pp. 60–65).

Fenton, N. E., & Pfleeger, S. L. (1998). *Software metrics: A rigorous and practical approach* (2nd ed.). Boston: PWS.

Fenton, N., & Neil, M. (1999). A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, *25*(5), 675–689.

Ganek, A. G., & Corbi T. A. (2003). The dawning of autonomic computing era. *IBM Systems Journal*, *42*(1), 5–18.

Grosser, D., Sahraoui, H. A., & Valtchev, P. (2002). Predicting software stability using case-based reasoning. In *Proceedings of 17th IEEE international conference on automated software engineering (ASE)* (pp. 295–298).

Khoshgoftaar, T., Allen, E., Hudepohl, J., & Aud, S. (1997). Applications of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks*, *8*(4), 902–909.

Ling, C., Sheng, V., Bruckhaus, T., & Madhavji, N. (2006). Maximum profit mining and its application in software development. In *Proceedings of the 12th ACM international conference on knowledge discovery and data mining (SIGKDD)* (pp. 929–934).

Mao, Y., Sahraoui, H., & Lounis, H. (1998). Reusability hypothesis verification using machine learning techniques: A case study. In *Proceedings of the 13th IEEE international conference on automated software engineering* (pp. 84–93).

Oliveira, A. (2006). Estimation of software project effort with support vector regression. *Neurocomputing*, *69*(13–15), 1749–1753.

Padberg, F., Ragg, T., & Schoknecht, R. (2004). Using machine learning for estimating the defect content after an inspection. *IEEE Transactions on Software Engineering*, *30*(1), 17–28.

Pai, P. F., & Hong, W. C. (2006). Software reliability forecasting by support vector machines with simulated annealing algorithms. *Journal of Systems and Software*, *79*(6), 747–755.

Pendharkar, P. C. (2004). An exploratory study of object-oriented software component size determinants and the application of regression tree forecasting models. *Information and Management*, *42*(1), 61–73.

Pfleeger, S. L., & Atlee J. M. (2003). *Software engineering: Theory and practice*. Upper Saddle River, NJ: Prentice-Hall.

Quah, T. S., & Thwin, M. M. T. (2003). *Application of neural networks for software quality prediction using object-oriented metrics*. In *Proceedings of international conference on software maintenance* (pp. 22–26).

Regolin, E. N., de Souza, G. A., Pozo, A. R. T., & Vergilio, S. R. (2003). *Exploring machine learning techniques for software size estimation*. In *Proceedings of the 23rd international conference of the Chilean computer science society (SCCC)* (pp. 130–136).

Sayyad Shirabad, J., Lethbridge, T. C., & Matwin, S. (2007). Modeling relevance relations using machine learning techniques. In J. Tsai & D. Zhang (Eds.), *Advances in machine learning applications in software engineering* (pp. 168–207). IGI.

Seliya, N. & Khoshgoftaar, T. M. (2007). Software quality estimation with limited fault data: a semi-supervised learning perspective. *Software Quality Journal*, *15*(3), 327–344.

Shepperd, M., & Schofield, C. (1997). Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, *23*(11), 736–743.

Sitte, R. (1999). Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration. *IEEE Transactions on Reliability*, *48*(3), 285–291.

Xing, F., Guo, P., & Lyu, M. R. (2005). A novel method for early software quality prediction based on support vector machine. In *Proceedings of IEEE international conference on software reliability engineering* (pp. 213–222).

Zhang, Du., & Tsai, J. P. (2003). Machine learning and software engineering. *Software Quality Journal*, *11*(2), 87–119.

Preference Learning

Johannes Fürnkranz[1], Eyke Hüllermeier[2]
[1]TU Darmstadt
[2]Philipps-Universität Marburg

P

Synonyms

Learning from preferences

Definition

Preference learning refers to the task of learning to predict an order relation on a collection of objects (alternatives). In the training phase, preference learning algorithms have access to examples for which the sought order relation is (partially) known. Depending on the formal modeling of the preference context and the alternatives to be ordered, one can distinguish between *object* ranking problems and *label* ranking problems. Both types of problems can be approached in two fundamentally different ways, either by modeling the binary preference relation directly, or by inducing this relation indirectly via an underlying (latent) utility function.

Motivation and Background

Preference information plays a key role in automated decision making and appears in various guises in Artificial Intelligence (AI) research, notably in fields such as agents, non-monotonic reasoning, constraint satisfaction, planning, and qualitative decision theory (Doyle, 2004). Preferences provide a means for specifying desires in a declarative way, which is a point of critical importance for AI. In fact, considering AI's paradigm of a rationally acting (decision-theoretic) agent, the behavior of such an agent has to be driven by an underlying preference model, and an agent recommending decisions or acting on behalf of a user should clearly reflect that user's preferences. Therefore, the formal modeling of preferences can be considered an essential aspect of autonomous agent design.

Drawing on past research on knowledge representation and reasoning, AI offers qualitative and symbolic methods for *modeling* and *processing* preferences that can reasonably complement standard approaches from economic decision theory, namely numerical *utility functions* and binary *preference relations*.

In practice, preference modeling can still become a rather cumbersome task if it must be done by hand. This is an important motivation for preference *learning*, which is meant to support and partly automatize the design of preference models. Roughly speaking, preference learning is concerned with the automated acquisition of preference models from data, that is, data from which (possibly uncertain) preference information can be deduced in a direct or indirect way.

Computerized methods for revealing the preferences of individuals (users) are useful not only in AI, but also in many related fields, notably in areas such as information retrieval, information systems, and e-commerce, where an increasing trend toward *personalization* of products and services can be recognized.Correspondingly, a number of methods and tools, such as recommender systems and collaborative filtering, have been proposed in the recent literature, which could in principle be subsumed under the heading of preference learning. In fact, one should realize that preference learning is a relatively recent and emerging topic. A first attempt for setting a common framework in this area can be found in Fürnkranz and Hüllermeier (2010). In this article, we shall therefore focus on two particular learning tasks that have been studied in the realm of machine learning and can be considered as extensions of classical machine learning problems.

Before proceeding, we introduce some basic notation that will be used later on. A weak preference relation $\succeq$ on a set $\mathcal{A}$ is a reflexive and transitive binary relation. Such a relation induces a strict preference $\succ$ and an indifference relation $\sim$ as follows: $a \succ b$ iff $(a \succeq b)$ and $(b \not\succeq a)$; moreover, $a \sim b$ iff $(a \succeq b)$ and $(b \succeq a)$. In agreement with our preference semantics, we shall interpret $a \succeq b$ as "alternative a is at least as good as alternative b." Let us note, however, that the term "preference" should not be taken literally and instead always be interpreted in a wide sense as a kind of order relation. Thus, $a \succ b$ may indeed mean that alternative a is more liked by a person than b, but also, e.g., that a is an algorithm that outperforms b on a certain problem, or that a is a student finishing her studies before another student b.

Subsequently, we shall focus on an especially simple type of preference structure, namely *total strict orders* or *rankings*, that is, relations $\succ$ which are total, irreflexive, and transitive. If $\mathcal{A}$ is a finite set $\{a_1, \ldots, a_m\}$, a ranking of $\mathcal{A}$ can be identified with a permutation τ of $\{1, \ldots, m\}$, as there is a unique permutation τ such that $a_i \succ a_j$ if and only if $\tau(i) < \tau(j)$ ($\tau(i)$ is the position of a_i in the ranking). We shall denote the class of all permutations of $\{1, \ldots, m\}$ by $\mathcal{S}_m$. Moreover, by abuse of notation, we shall sometimes employ the terms "ranking" and "permutation" synonymously.

Structure of the Learning System

As mentioned before, a considerable number of diverse approaches have been proposed under terms like ranking and preference learning. In the following, we shall distinguish between *object ranking problems*, where the task is to order subsets of objects, and *label ranking problems*, where the task is to assign a permutation of a fixed set of labels to a given instance. An important difference between these problems concerns the formal representation of the preference context and the alternatives to be ordered: In object ranking, the objects themselves are characterized by properties, typically in terms of an attribute-value representation. Thus, the ranking model can refer to *properties of the alternatives* and can therefore be applied to arbitrary sets of such alternatives. In

label ranking, the alternatives to be ranked are labels as in classification learning, i.e., mere identifiers without associated properties. Instead, the ranking context is characterized in terms of a (ranking) instance from a given instance space, and the task of the model is to rank alternatives depending on *properties of the context*. Thus, the context may now change (as opposed to object ranking, where it is implicitly fixed) but the objects to be ranked remain the same. Or, stated differently, object ranking is the problem to rank varying sets of objects under invariant preferences, whereas label ranking is the problem to rank an invariant set of objects under varying preferences.

For both problem types, there are two principal ways to approach them. One possibility is to learn a *utility function* that induces the sought ranking by *evaluating individual objects*. The alternative is to *compare pairs of objects*, that is, to learn a *binary preference relation*.

Note that the first approach implicitly assumes an underlying total order relation, since numerical (or at least totally ordered) utility scores enforce the comparability of alternatives. The second approach is more general in this regard, as it also allows for partial order relations. On the other hand, this approach may lead to complications if the target is indeed a total order, since a set of *hypothetical* binary preferences induced from empirical data is not necessarily transitive.

Learning from Object Preferences

Given:

- A (potentially infinite) set $\mathcal{X}$ of objects (each object typically represented by a feature vector)
- A finite set of pairwise preferences $x_i \succ x_j$, $(x_i, x_j) \in \mathcal{X} \times \mathcal{X}$

Find:

- A ranking function $r(\cdot)$ that, given a set of objects $\mathcal{O} \subseteq \mathcal{X}$ as input, returns a permutation (ranking) of these objects

The most frequently studied problem in learning from preferences is to induce a *ranking function* $r(\cdot)$ that is able to order any subset $\mathcal{O}$ of an underlying class $\mathcal{X}$ of objects. That is, $r(\cdot)$ assumes as input a subset $\mathcal{O} = \{x_1, \ldots, x_n\} \subseteq \mathcal{X}$ of objects and returns as output a permutation τ of $\{1, \ldots, n\}$. The interpretation of this permutation is that object x_i is preferred to x_j whenever $\tau(i) < \tau(j)$. The objects themselves are typically characterized by a finite set of features as in conventional attribute-value learning. The training data consists of a set of exemplary pairwise preferences. A survey of object ranking approaches can be found in Kamishima et al. (2010).

Note that, in order to evaluate the predictive performance of a ranking algorithm, an accuracy measure is needed that compares a predicted ranking with a given reference. To this end, one can refer, for example, to so-called ▶rank correlation measures that have been proposed in statistics. In the context of ranking, such measures play the role of, say, the classification rate in classification learning.

As an example of object ranking consider the problem of learning to rank query results of a search engine (Joachims, 2002). The training information could be provided implicitly by the user who clicks on some of the links in the query result and not on others. This information can be turned into binary preferences by assuming that the selected pages are preferred over nearby pages that are not clicked on (Radlinski et al., 2010).

Learning from Label Preferences

Given:

- A set of training instances $\{x_k \mid k = 1, \ldots, n\} \subseteq \mathcal{X}$ (each instance typically represented by a feature vector)
- A set of labels $\mathcal{L} = \{\lambda_i \mid i = 1, \ldots, m\}$
- For each training instance x_k: a set of associated *pairwise preferences* of the form $\lambda_i \succ_{x_k} \lambda_j$

Find:

- A ranking function in the form of an $\mathcal{X} \longrightarrow \mathcal{S}_m$ mapping that assigns a ranking (permutation) $\succ_x$ of $\mathcal{L}$ to every $x \in \mathcal{X}$

In this learning scenario, the problem is to predict, for any instance x (e.g., a person) from an instance space $\mathcal{X}$, a preference relation (ranking) $\succ_x \subseteq \mathcal{L} \times \mathcal{L}$ among a finite set $\mathcal{L} = \{\lambda_1, \ldots, \lambda_m\}$ of labels or alternatives, where $\lambda_i \succ_x \lambda_j$ means that instance x prefers the label

λ_i to the label λ_j. More specifically, as we are especially interested in the case where $\succ_x$ is a total strict order, the problem is to predict a permutation of $\mathcal{L}$. The training information consists of a set of instances for which (partial) knowledge about the associated preference relation is available. More precisely, each training instance x is associated with a subset of all pairwise preferences. Thus, despite the assumption of an underlying ("true") target ranking, the training data is not expected to provide full information about such rankings. Besides, in order to increase the practical usefulness of the approach, learning algorithms should even allow for inconsistencies, such as pairwise preferences which are conflicting due to observation errors.

The above formulation follows (Hüllermeier et al. 2008), similar formalizations have been proposed independently by several authors (Dekel et al., 2004; Fürnkranz and Hüllermeier, 2003; Har-Peled et al., 2002). A survey can be found in Vembu and Gärtner (2010). Aiolli and Sperduti (2010) proposed an interesting generalization of this framework that allows one to specify both qualitative and quantitative preference constraints on an underlying utility function. In addition to comparing pairs of alternatives, it is possible to specify constraints of the form $\lambda_i \succeq_x t$, which means that the utility score of alternative x reaches the numerical threshold t.

Label ranking contributes to the general trend of extending machine learning methods to complex and structured output spaces (Fürnkranz and Hüllermeier, 2010; Tsochantaridis et al., 2004). Moreover, label ranking can be viewed as a generalization of several standard learning problems. In particular, the following well-known problems are special cases of learning label preferences:

- ►*Classification*: A single class label λ_i is assigned to each example x_k. This is equivalent to the set of preferences $\{\lambda_i \succ_{x_k} \lambda_j \,|\, 1 \leq j \neq i \leq m\}$.
- ►*Multi-label classification*: Each training example x_k is associated with a subset $L_k \subseteq \mathcal{L}$ of possible labels. This is equivalent to the set of preferences $\{\lambda_i \succ_{x_k} \lambda_j \,|\, \lambda_i \in L_k, \lambda_j \in \mathcal{L} \setminus L_k\}$.

In each of the former scenarios, the sought prediction can be obtained by post-processing the output of a ranking model $f : \mathcal{X} \longrightarrow \mathcal{S}_m$ in a suitable way. For example, in classification learning, where only a single label is requested, it suffices to project a label ranking to the top-ranked label.

Applications of this general framework can be found in various fields, for example in marketing research; here, one might be interested in discovering dependencies between properties of clients and their preferences for products. Another application scenario is ►meta-learning, where the task is to rank learning algorithms according to their suitability for a new dataset, based on the characteristics of this dataset. Moreover, every preference statement in the well-known CP-nets approach (Boutilier et al., 2004), a qualitative graphical representation that reflects conditional dependence and independence of preferences under a *ceteris paribus* interpretation, formally corresponds to a label ranking function that orders the values of a certain attribute depending on the values of the parents of this attribute (predecessors in the graph representation).

Learning Utility Functions

A natural way to represent preferences is to evaluate the alternatives by means of a utility function. In the object preferences scenario, such a function is a mapping $f : \mathcal{X} \longrightarrow \mathcal{U}$ that assigns a utility degree $f(x)$ to each object x and, thereby, induces a linear order on $\mathcal{X}$; the utility scale $\mathcal{U}$ is usually given by the real numbers $\mathbb{R}$, but sometimes an ordinal scale is preferred (note that an ordinal scale will typically produce many ties, which is undesirable if the target is a ranking). In the label preferences scenario, a utility function $f_i : \mathcal{X} \longrightarrow \mathcal{U}$ is needed for every label λ_i, $i = 1, \ldots, m$. Here, $f_i(x)$ is the utility assigned to alternative λ_i by instance x. To obtain a ranking for x, the alternatives are ordered according to their utility scores, i.e., a ranking $\succ_x$ is derived that satisfies $\lambda_i \succ_x \lambda_j \Rightarrow f_i(x) \geq f_j(x)$.

If the training data offers the utility scores directly, preference learning reduces to a standard regression (up to a monotonic transformation of the utility values) or an ordinal regression problem, depending on the underlying utility scale. This information can rarely be assumed, however. Instead, usually only constraints derived from comparative preference information of the form "This object (or label) should have a higher utility score than that object (or label)" are given. Thus, the challenge for the learner is to find a function that is as much as possible in agreement with a set of such constraints.

For object ranking approaches, this idea has first been formalized by Tesauro (1989) under the name *comparison training*. He proposed a symmetric neural-network architecture that can be trained with representations of two states and a training signal that indicates which of the two states is preferable. The elegance of this approach comes from the property that one can replace the two symmetric components of the network with a single network, which can subsequently provide a real-valued evaluation of single states. Similar ideas have also been investigated for training other types of classifiers, in particular support vector machines. We already mentioned Joachims (2002) who analyzed "click-through data" in order to rank documents retrieved by a search engine according to their relevance. Earlier, Herbrich et al. (1998) have proposed an algorithm for training SVMs from pairwise preference relations between objects.

For the case of label ranking, a corresponding method for learning the functions $f_i(\cdot)$, $i = 1,\ldots,m$, from training data has been proposed in the framework of *constraint classification* (Har-Peled et al., 2002). The learning method proposed in this work constructs two training examples, a positive and a negative one, for each given preference $\lambda_i \succ_x \lambda_j$, where the original N-dimensional training example (feature vector) x is mapped into an $(m \times N)$-dimensional space. The positive example copies the original training vector x into the components $((i-1)\times N+1)\ldots(i\times N)$ and its negation into the components $((j-1)\times N+1)\ldots(j\times N)$ of a vector in the new space; the remaining entries are filled with 0. The negative example has the same elements with reversed signs. In this $(m\times N)$-dimensional space, the learner tries to find a hyperplane that separates the positive from the negative examples. For classifying a new example x_0, the labels are ordered according to the response resulting from multiplying x_0 with the ith N-element section of the hyperplane vector.

Learning Preference Relations

As mentioned before, instead of learning a latent utility function that evaluates individual objects, an alternative approach to preference learning consists of comparing pairs of objects (labels) in terms of a binary preference relation. For object ranking problems, this pairwise approach has been pursued in Cohen et al. (1999). The authors propose to solve object ranking problems by learning a binary preference predicate $Q(x,x')$, which predicts whether x is preferred to x' or vice versa. A final ordering is found in a second phase by deriving a ranking that is maximally consistent with these predictions.

For label ranking problems, the pairwise approach has been introduced in Fürnkranz and Hüllermeier (2003) as a natural extension of *pairwise classification*, a well-known ▶class binarization technique. The idea is to train a separate model (base learner) $\mathcal{M}_{i,j}$ for each pair of labels $(\lambda_i,\lambda_j) \in \mathcal{L}$, $1 \le i < j \le m$; thus, a total number of $m(m-1)/2$ models is needed. For training, a preference information of the form $\lambda_i \succ_x \lambda_j$ is turned into a (classification) example (x,y) for the learner $\mathcal{M}_{a,b}$, where $a = \min(i,j)$ and $b = \max(i,j)$. Moreover, $y = 1$ if $i < j$ and $y = 0$ otherwise. Thus, $\mathcal{M}_{a,b}$ is intended to learn the mapping that outputs 1 if $\lambda_a \succ_x \lambda_b$ and 0 if $\lambda_b \succ_x \lambda_a$:

$$x \mapsto \begin{cases} 1 & \text{if} \quad \lambda_a \succ_x \lambda_b \\ 0 & \text{if} \quad \lambda_b \succ_x \lambda_a \end{cases} \tag{1}$$

The mapping (1) can be realized by any binary classifier. Instead of a $\{0,1\}$-valued classifier, one can of course also employ a scoring classifier. For example, the output of a probabilistic classifier would be a number in the unit interval $[0,1]$ that can be interpreted as a probability of the preference $\lambda_a \succ_x \lambda_b$.

At classification time, a query $x_0 \in \mathcal{X}$ is submitted to the complete ensemble of binary learners. Thus, a collection of predicted pairwise preference degrees $\mathcal{M}_{i,j}(x)$, $1 \le i,j \le m$, is obtained. The problem, then, is to turn these pairwise preferences into a ranking of the label set $\mathcal{L}$. To this end, different ranking procedures can be used. The simplest approach is to extend the (weighted) voting procedure that is often applied in pairwise classification: For each label λ_i, a score

$$S_i = \sum_{1\le j\ne i\le m} \mathcal{M}_{i,j}(x_0)$$

is derived (where $\mathcal{M}_{i,j}(x_0) = 1 - \mathcal{M}_{j,i}(x_0)$ for $i > j$), and then all labels are ordered according to these scores. Despite its simplicity, this ranking procedure has several appealing properties. Apart from its computational efficiency, it turned out to be relatively robust in practice and, moreover, it possesses some provable

optimality properties in the case where Spearman's rank correlation is used as an underlying accuracy measure. Roughly speaking, if the binary learners are unbiased probabilistic classifiers, the simple "ranking by weighted voting" procedure yields a label ranking that maximizes the expected Spearman rank correlation (Hüllermeier and Fürnkranz, 2010). Finally, it is worth mentioning that, by changing the ranking procedure, the pairwise approach can also be adjusted to accuracy measures other than Spearman's rank correlation.

Future Directions

As we already mentioned, preference learning is an emerging topic and, as a subfield of machine learning, still in its infancy. In particular, one may expect that, apart from the object and label ranking problems, other settings and frameworks will be studied in the future. But even for object and label ranking as introduced above, there are several open questions and promising lines of future research. The most obvious extension concerns the type of preference structure predicted as an output: For many applications, it is desirable to predict structures which are more general than rankings, e.g., which allow for *incomparability* (partial orders) or *indifference* between alternatives. In a similar vein, the pairwise approach to label ranking has recently been extended to the prediction of so-called "calibrated" rankings in Fürnkranz et al. (2008). A calibrated ranking is a ranking with an additional "zero-point" that separates between a positive and a negative part, thereby integrating the problems of label ranking and multi-label classification.

Cross References

►Classification
►Meta-Learning
►Rank Correlation

Recommended Reading

Aiolli, F., & Sperduti, A. (2010). A preference optimization based unifying framework for supervised learning problems. In J. Fürnkranz & E. Hüllermeier (Eds.), *Preference learning* (pp. 19–40). Springer.

Boutilier, C., Brafman, R., Domshlak, C., Hoos, H., & Poole, D. (2004). CP-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research, 21*, 135–191.

Cohen, W. W., Schapire, R. E., & Singer, Y. (1999). Learning to order things. *Journal of Artificial Intelligence Research, 10*, 243–270.

Dekel, O., Manning, C. D., & Singer, Y. (2004). Log-linear models for label ranking. In S. Thrun, L. K. Saul, & B. Schölkopf, (Eds.), *Advances in neural information processing systems (NIPS-03)* (pp. 497–504). Cambridge: MIT Press.

Doyle, J. (2004). Prospects for preferences. *Computational Intelligence, 20*(2), 111–136.

Fürnkranz, J., & Hüllermeier, E. (2003). Pairwise preference learning and ranking. In N. Lavrač, D. Gamberger, H. Blockeel, & L. Todorovski (Eds.), *Proceedings of the 14th European Conference on Machine Learning (ECML-03), volume 2837 of Lecture Notes in Artificial Intelligence*, Springer, Cavtat, Croatia, pp. 145–156.

Fürnkranz, J., & Hüllermeier, E. (Eds.). (2010). *Preference learning*. Springer.

Fürnkranz, J., & Hüllermeier, E. (2010). Preference learning and ranking by pairwise comparison. In J. Fürnkranz & E. Hüllermeier (Eds.), *Preference Learning* (pp. 63–80). Springer.

Fürnkranz, J., & Hüllermeier, E. (2010). Preference learning: an introduction. In J. Fürnkranz & E. Hüllermeier (Eds.), *Preference Learning* (pp. 1–18). Springer.

Fürnkranz, J., Hüllermeier, E., Loza Mencía, E., & Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Machine Learning, 73*(2), 133–153.

Har-Peled, S., Roth, D., & Zimak, D. (2002). Constraint classification: a new approach to multiclass classification. In N. Cesa-Bianchi, M. Numao, & R. Reischuk, (Eds.), *Proceedings of the 13th International Conference on Algorithmic Learning Theory (ALT-02)* (pp. 365–379), Springer, Lübeck, Germany.

Herbrich, R., Graepel, T., Bollmann-Sdorra, P., & Obermayer, K. (1998). Supervised learning of preference relations. *Proceedings des Fachgruppentreffens Maschinelles Lernen (FGML-98)*, pp. 43–47.

Hüllermeier, E., & Fürnkranz, J. (2010). On predictive accuracy and risk minimization in pairwise label ranking. *Journal of Computer and System Sciences, 76*(1), 49–62.

Hüllermeier, E., Fürnkranz, J., Cheng, W., & Brinker, K. (2008). Label ranking by learning pairwise preferences. *Artificial Intelligence, 172*, 1897–1916.

Joachims, T. (2002). Optimizing search engines using clickthrough data. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02)*, ACM Press, pp. 133–142.

Kamishima, T., Kazawa, H., & Akaho, S. (2010). A survey and empirical comparison of object ranking methods. In J. Fürnkranz & E. Hüllermeier (Eds.), *Preference learning* (pp. 175–195). Springer.

Radlinski, F., Kurup, M., & Joachims, T. (2010). Evaluating search engine relevance with click-based metrics. In J. Fürnkranz & E. Hüllermeier (Eds.), *Preference learning* (pp. 329–353). Springer.

Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems 1 (NIPS-88)* (pp. 99–106), Morgan Kaufmann.

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. *ICML*.

Vembu, S., & Gärtner, T. (2010). Label ranking algorithms: a survey. In J. Fürnkranz & E. Hüllermeier (Eds.), *Preference learning* (pp. 43–61). Springer.

Pre-Pruning

Synonyms
Stopping criteria

Definition
Pre-pruning is a ►Pruning mechanism that monitors the learning process and prevents further refinements if the current hypothesis becomes too complex.

Cross References
►Overfitting
►Post-Pruning
►Pruning

Presynaptic Neuron

The neuron that sends signals across a synaptic connection. A chemical synaptic connection between two neurons allows to transmit signals from a presynaptic neuron to a postsynaptic neuron.

Principal Component Analysis

Synonyms
PCA

Definition
Principal Component Analysis (PCA) is a ►dimensionality reduction technique. It is described in ►covariance matrix.

Prior

►Prior Probability

Privacy-Preserving Data Mining

►Privacy-Related Aspects and Techniques

Privacy-Related Aspects and Techniques

Stan Matwin
University of Ottawa, Ottawa, ON, Canada and Polish Academy of Sciences, Warsaw, Poland

Synonyms
Privacy-preserving data mining

Definition
The privacy-preserving aspects and techniques of machine learning cover the family of methods and architectures developed to protect the privacy of people whose data are used by machine learning (ML) algorithms. This field, also known as privacy-preserving data mining (PPDM), addresses the issues of data privacy in ML and data mining. Most existing methods and approaches are intended to hide the original data from the learning algorithm, while there is emerging interest in methods ensuring that the learned model does not reveal private information. Another research direction contemplates methods in which several parties bring their data into the model-building process without mutually revealing their own data.

Motivation and Background
The key concept for any discussion of the privacy aspects of data mining is the definition of privacy. After Alan Westin, we understand privacy as the ability "of individuals... to determine for themselves when, how, and to what extent information about them is communicated to others" (Westin, 1967). One of the main societal concerns about modern computing is that the storing, keeping, and processing of massive amounts of data may jeopardize the privacy of individuals whom the data represent. In particular, ML and its power to find patterns and infer new facts from existing data makes it difficult for people to control information

about themselves. Moreover, the infrastructure normally put together to conduct large-scale model building (e.g., large data repositories and data warehouses), is conducive to misuse of data. Personal data, amassed in large collections that are easily accessed through databases and often available online to the entire world, become – as phrased by Moor in an apt metaphor (Moor, 2004) – "greased." It is difficult for people to control the use of this data.

Theory/Solutions

Basic Dimensions of Privacy Techniques

Privacy-related techniques can be characterized by: (1) the kind of source data modification they perform, e.g., data perturbation, randomization, generalization, and hiding; (2) the ML algorithm that works on the data and how is it modified to meet the privacy requirements imposed on it; and (3) whether the data are centralized or distributed among several parties, and – in the latter case – on what the distribution is based. But even at a more basic level, it is useful to view privacy-related techniques along just two fundamental dimensions.

The first dimension defines what is protected as private – is it the data itself, or the model (the results of data mining)? As we show below, the knowledge of the latter can also lead to identifying and revealing information about individuals. The second dimension defines the protocol of the use of the data: are the data centralized and owned by a single owner, or are the data distributed among multiple parties? In the former case, the owner needs to protect the data from revealing information about individuals represented in the data when that data is being used to build a model by someone else. In the latter case, we assume that the parties have limited trust in each other: they are interested in the results of data mining performed on the union of the data of all the parties, while not trusting the other parties to see their own data without first protecting it against disclosure of information about individuals to other parties.

Moreover, work in PPDM has to apply a framework that is broader than the standard ML methodology. When privacy is an important goal, what matters in performance evaluation is not only the standard ML performance measures, but also some measure of the privacy achieved, as well as some analysis of the robustness of the approach to attacks.

In this article, we structure our discussion of the current work on PPDM in terms of the taxonomy proposed above. This leads to the following bird's-eye view of the field.

Protecting Centralized Data

This subfield emerged in 2000 with the seminal paper by Agrawal and Srikant (2000). They stated the problem as follows: given data in the standard ▶attribute-value representation, how can an accurate ▶decision tree be built so that, instead of using original attribute values x_i, the decision tree induction algorithm takes input values $x_i + r$, where r belongs to a certain distribution (Gaussian or uniform). This is a data perturbation technique: the original values are changed beyond recognition, while the distributional properties of the entire data set that decision tree ▶induction uses remain the same, at least up to a small (empirically, less than 5%) degradation in accuracy. There is a clear trade-off between the privacy assured by this approach and the quality of the model compared to the model obtained from the original data. This line of research has been continued in Evfimievski, Srikant, Agrawal, and Gehrke (2002) where the approach is extended to association rule mining. As a note of caution about these results, Kargupta, Datta, and Wang (2003) have shown, in 2003, how the randomization approaches are sensitive to attack. They demonstrate how the noise that randomly perturbs the data can be viewed as a random matrix, and that the original data can be accurately estimated from the perturbed data using a spectral filter that exploits some theoretical properties of random matrices.

The simplest and most widely used privacy preservation technique is anonymization of data (also called de-identification). In the context of de-identification, it is useful to distinguish three types of attributes.

Explicit identifiers allow direct linking of an instance to a person (e.g., a cellular phone number or a driver's license number to its holder).

Quasi-identifiers, possibly combined with other attributes, may lead to other data sources capable of unique identification. For instance, Sweeney (2001) shows that the quasi-identifier triplet < date of birth, 5 digit postal code, gender >, combined with the voters'

list (publicly available in the USA) uniquely identifies 87% of the population of the country. As a convincing application of this observation, using quasi-identifiers, Sweeney was able to obtain health records of the governor of Massachusetts from a published dataset of health records of all state employees in which only explicit identifiers have been removed.

Finally, non-identifying attributes are those for which there is no known inference linking to an explicit identifier. Usually performed as part of data preparation, anonymization removes all explicit identifiers from the data.

While anonymization is by far the most common privacy-preserving technique used in practice, it is also the most fallible one. In August 2006, for the benefit of the Web Mining Research community, AOL published 20 million search records (queries and URLs the members had visited) from 658,000 of its members. AOL had performed what it believed was anonymization, in the sense that it removed the names of the members. However, based on the queries – which often contained information that would identify a small set of members or a unique person – it was easy, in many cases, to manually re-identify the AOL member using secondary public knowledge sources. An inquisitive New York Times journalist identified one member and interviewed her.

L. Sweeney is to be credited with sensitizing the privacy community to the fallacy of anonymization: "Shockingly, there remains a common incorrect belief that if the data look anonymous, it is anonymous" (Sweeney, 2001). Even if information is de-identified today, future data sources may make re-identification possible. As anonymization is very commonly used prior to model building from medical data, it is interesting that this type of data is prone to specific kinds of re-identification, and therefore anonymization of medical data should be done with particular skill and understanding of the data. Malin (2005) shows how the four main de-identification techniques used in anonymization of genomic data are prone to known, published attacks that can re-identify the data. Moreover, he points out that there will never be certainty about de-identification for quasi-identifiers, as new attributes and data sources that can lead to a linkage to explicitly identifying attributes are constantly being engineered as part of genetics research.

Other perturbation approaches targeting binary data involve changing (flipping) values of selected attributes with a given probability (Du & Zhan, 2003; Zhan & Matwin, 2004), or replacing the original attribute with a value that is more general in some pre-agreed taxonomy (Iyengar, 2002). Generalization approaches often use the concept of *k*-anonymity: any instance in the database is indistinguishable from other *k*–1 instances (for every row in the database there are *k*–1 identical rows). Finding the least general *k*-anonymous generalization of a database (i.e., moving the least number of edges upward in a given taxonomy) is an optimization task, known to be *NP*-complete. There are heuristic solutions proposed for it; e.g., Iyengar (2002) uses a ▶genetic algorithm for this task. Friedman, Schuster, and Wolff (2006) shows how to build k-anonymity into the decision tree induction. Lately, PPDM researchers have pointed out some weaknesses of the *k*-anonymity approach. In particular, attacks on data with some properties (e.g., skewed distribution of values of a sensitive attribute, or specific background knowledge) have been described, and techniques to prevent such attacks have been proposed. The notion of *p*-sensitivity or *l*-diversity proposed in Machanavajjhala, Kifer, Gehrke, and Venkitasubramaniam (2007) addresses these weaknesses of *k*-anonymity by modifying *k*-anonymity techniques so that the abovementioned attacks do not apply. Furthermore, *t*-closeness (Ninghui, Tiancheng & Venkatasubramanian, 2007) shows certain shortcomings of these techniques and the resulting potential attacks, and proposes a data perturbation technique which ensures that the distribution of the values of the sensitive attribute in any group resulting from anonymization is close to its distribution in the original table. Some authors, e.g., Domingo-Ferrer, Sebé, and Solanas (2008), propose the integration of several techniques addressing shortcomings of *k*-anonymity into a single perturbation technique. The drawback of these solutions is that they decrease the utility of the data more than the standard *k*-anonymity approaches.

Protecting the Model (Centralized Data)

Is it true that when the data are private, there will be no violation of privacy? The answer is no. In some circumstances, the model may reveal private information about individuals. Atzori, Bonchi, Giannotti, and

Pedreschi (2005) gives an example of such a situation for association rules: suppose the ▶association rule $a_1 \wedge a_2 \wedge a_3 \Rightarrow a_4$ has support sup = 80, confidence conf = 98.7%. This rule is 80-anonymous, but considering that

$$\sup(\{a_1, a_2, a_3\}) = \frac{\sup(\{a_1, a_2, a_3, a_4\})}{\text{conf}} = \frac{80}{0.0987} \approx 81.05$$

and given that the pattern $a_1 \wedge a_2 \wedge a_3 \wedge a_4$ holds for 80 individuals, and the pattern $a_1 \wedge a_2 \wedge a_3$ holds for 81 individuals, clearly the pattern $a_1 \wedge a_2 \wedge a_3 \wedge \neg a_4$ holds for just one person. Therefore, the rule unexpectedly reveals private information about a specific person. Atzori et al. (2005) proposes to apply k-anonymity to patterns instead of data, as in the previous section. The authors define inference channels as ▶itemsets from which it is possible to infer other itemsets that are not k-anonymous, as in the above example. They then show an efficient way to represent and compute inference channels, which, once known, can be blocked from the output of an association rule finder. The inference channel problem is also discussed in Oliveira, Zaïane, and Saygin (2004), where itemset "sanitization" removes itemsets that lead to sensitive (non-k-anonymous) rules.

This approach is an interesting continuation of Sweeney's classical work (Sweeney, 2001), and it addresses an important threat to privacy ignored by most other approaches based on data perturbation or cryptographic protection of the data.

Distributed Data

Most of the work mentioned above addresses the case of centralized data. The distributed situation, however, is often encountered and has important applications. Consider, for example, several hospitals involved in a multi-site medical trial that want to mine the data describing the union of their patients. This increases the size of the population subject to data analysis, thereby increasing the scope and the importance of the trial. In another example, a car manufacturer performing data analysis on the set of vehicles exhibiting a given problem wants to represent data about different components of the vehicle originating in databases of the suppliers of these components. In general, if we abstractly represent the database as a table, there are two collaborative frameworks in which data is distributed. Horizontally partitioned data is distributed by rows (all parties have the same attributes, but different instances – as in the medical study example). Vertically partitioned data is distributed by columns (all parties have the same instances; some attributes belong to specific parties, and some, such as the class, are shared among all parties – as in the vehicle data analysis example).

An important branch of research on learning from distributed data while parties do not reveal their data to each other is based on results from computer security, specifically from cryptography and from the secure multiparty computation (SMC). Particularly interesting is the case when there is no trusted external party – all the computation is distributed among parties that collectively hold the partitioned data. SMC has produced constructive results showing how any Boolean function can be computed from inputs belonging to different parties, so that the parties never get to know input values that do not belong to them. These results are based on the idea of splitting a single data value between two parties into "shares," so that none of them knows the value but they can still do computation on the shares using a gate such as *exclusive or* Yao (1986). In particular, there is an SMC result known as secure sum: k parties have private values x_i and they want to compute $\Sigma_I\, x_i$ without disclosing their x_i to any other party. This result, and similar results for value comparison and other simple functions, are the building blocks of many privacy-preserving ML algorithms. On that basis, a number of standard ▶classifier induction algorithms, in their horizontal and vertical partitioning versions, have been published, including decision tree (▶ID3) induction (Friedman, Schuster & Wolff, 2006), ▶Naïve Bayes, the ▶Apriori association rule mining algorithm (Kantarcioglu & Clifton, 2004; Vaidya & Clifton, 2002), and many others.

We can observe that data privacy issues extend to the use of the learned ▶model. For horizontal partitioning, each party can be given the model and apply it to the new data. For vertical partitioning, however, the situation is more difficult: the parties, all knowing the model, have to compute their part of the decision that the model delivers, and have to communicate with selected other parties after this is done. For instance, for decision trees, a node n applies its test and contacts the party holding the attribute in the child c chosen by the test, giving c the test to perform. In this manner, a single

party n only knows the result of its test (the corresponding attribute value) and the tests of its children (but not their outcomes). This is repeated recursively until the leaf node is reached and the decision is communicated to all parties.

A different approach involving cryptographic tools other than Yao's circuits is based on the concept of homomorphic encryption (Paillier, 1999). Encryption e is homomorphic with respect to some operation * in the message space if there is a corresponding operation *′ in the ciphertext space, such that for any messages m1, m2, $e(\text{m1})^{*\prime}e(\text{m2}) = e(\text{m1}^{*}\text{m2})$. The standard RSA encryption is homomorphic with *′ being logical multiplication and * logical addition on sequences of bytes. To give a flavor of the use of homomorphic encryption, let us see in detail how this kind of encryption is used in computing the scalar product of two binary vectors.

Assume just two parties, Alice and Bob. They both have their private binary vectors $\text{A}_{1,\ldots,N}$, $\text{B}_{1,\ldots,N}$. In association rule mining, A_i and B_i represent A's and B's transactions projected on the set of items whose frequency is being computed. In our protocol, one of the parties is randomly chosen as a key generator. Assume Alice is selected as the key generator. Alice generates an encryption key (e) and a decryption key (d). She applies the encryption key to the sum of each value of A and a digital envelope $R_i^* X$ of A_i (i.e., $e(A_i i + R_i{}^* X)$), where R_i is a random integer and X is an integer that is greater than N. She then sends $e(A_i + R_i{}^* X)$s to Bob. Bob computes the multiplication $M = \prod_{j=1}^{N} [e(A_j + R_i{}^* X) \times \text{B}_j]$ when $B_j = 1$ (as when $B_j = 0$, the result of multiplication does not contribute to the frequency count). Now, $M = e(A_1 + A_2 + \cdots + A_j + (R_1 + R_2 + \cdots + R_1){}^* X)$ due to the property of homomorphic encryption. Bob sends the result of this multiplication to Alice, who computes $[d(e(A_1 + A_2 + \cdots + A_j + (R_1 + R_2 + \cdots + R_1){}^* X)])$ mod $X = (A_1 + A_2 + \cdots + A_1 + (R_1 + R_2 + \cdots + R_j){}^* X)$ mod X and obtains the scalar product. This scalar product is directly used in computing the frequency count of an itemset, where N is the number of items in the itemset, and A_i, B_i are Alice's and Bob's transactions projected on the itemset whose frequency is computed.

While more efficient than the SMC-based approaches, homomorphic encryption methods are more prone to attack, as their security is based on a weaker security concept (Paillier, 1999) than Yao's approach. In general, cryptographic solutions have the advantage of protecting the source data while leaving it unchanged: unlike data modification methods, they have no negative impact on the quality of the learned model. However, they have a considerable cost impact in terms of complexity of the algorithms, computation cost of the cryptographic processes involved, and the communication cost for the transmission of partial computational results between the parties (Subramaniam, Wright & Yang, 2004). Their practical applicability on real-life-sized datasets still needs to be demonstrated.

The discussion above focuses on protecting the data. In terms of our diagram in Table 1, we have to address its right column. Here, methods have been proposed to mainly address mainly the north-east entry of the diagram. In particular, in Vaidya and Clifton (2002) propose a method to compute association rules in an

Privacy-Related Aspects and Techniques. Table 1 Classification taxonomy to systematize the discussion of the current work in PPDM

	Data centralized	Data distributed
Protecting the data	Agrawal and Srikant (2000), Evfimievski, Srikant, Agrawal, and Gehrke (2002), Du and Zhan (2003), and Iyengar (2002)	Vaidya and Clifton, (2002), Vaidya, Clifton, Kantarcioglu and Patterson, (2008), and Kantarcioglu and Clifton, (2004)
Protecting the model	Oliveira, Zaïane and Saygin, (2004), Atzori, Bonchi, Giannotti, and Pedreschi (2005), Felty and Matwin (2002), Friedman, Schuster, and Wolff (2006)	Jiang and Atzori (2006)

environment where data is distributed. In particular, their method addresses the case of vertically partitioned data, where different parties hold different attribute sets for the same instances. The problem is solved without the existence of a trusted third party, using SMC. Independently, we have obtained a different solution to this task using homomorphic encryption techniques (Zhan, Matwin & Chang, 2007). Many papers have presented solutions for both vertically and horizontally partitioned data, and for different data mining tasks, e.g., Friedman, Schuster, and Wolff (2006) and Vaidya, Zhu, and Clifton (2006).

Moreover, Jiang and Atzori (2006) have obtained a solution for the model-protection case in a distributed setting (south-east quadrant in Table 1). Their work is based on a cryptographic technique, and addresses the case of vertical partitioning of the data among parties.

Evaluation

The evaluation of privacy-related techniques must be broader than standard ML evaluation. Besides evaluating the performance of the ML component using the appropriate tool (e.g., ►accuracy, ►ROC, support/confidence), one also needs to evaluate the various privacy aspects of a learned model. This is difficult, as there is no commonly accepted definition of privacy. Even if there were one, it would not be in quantitative, operational terms that can be objectively measured, but most certainly with references to moral and social values. For instance, Clifton (2005) points out that a definition of privacy as the "freedom from unauthorized intrusion" implies that we need to understand what constitutes an intrusion and that we can measure its extent. For these reasons, most definitions in current privacy-preserving data mining research are method-specific, without any comparison between different methods. For example, the classic work of Agrawal and Srikant (2000) measures privacy after data perturbation as the size of the interval to which the original value can be estimated. If we know that the original value was 0.5, and following a perturbation its best estimate is, with 95% confidence, within the interval [0.3, 0.7], then the amount of privacy is the size of this interval, (i.e., 0.4, with a confidence of 95%). Later, Agrawal and Aggrawal (2001) proposed a more general measure of data privacy measuring this property of a dataset that has been subject to one of the data perturbation techniques. The idea is that if noise from a random variable A is added to the data, we can measure the uncertainty of the perturbed values using differential entropy inherent in A. Specifically, if we add noise from a random variable A, the privacy is

$$\prod(A) = 2^{-\int_{\Omega_A}^{f_A(a)\log_2 f_A(a)\,da}},$$

where Ω_A is the domain of A. Privacy is 0 if the exact value is known (the entropy is ∞); if it is known that the data is in the interval of length a, $\prod(A) = a$.

Clifton (2005) argues that if disclosure is only possible to a group of people rather than a single person, then the size of the group is a natural measure of privacy. This is the case for k-anonymity methods. He further argues that a good evaluation measure should not only capture the likelihood of linking an ML result to an individual, but should also capture how intrusive this linking is. For instance, an association rule with a support value of 50 and a confidence level of 100% is 50-anonymous, but it also reveals the consequent of the rule to all 50 participants.

Finally, the style of evaluation needs to take into account attack analysis, as in Malin (2005).

Future Directions

One of the most pressing challenges for the community is to work out a quantifiable and socially comprehensible definition of privacy for the purpose of privacy-preserving techniques. This is clearly a difficult problem, likely not solvable by ML or even computer science alone. As privacy has basic social and economic dimensions, economics may contribute to an acceptable definition, as already explored in Rossi (2004).

Another important question is the ability to analyze data privacy, including inference from data using ML, in the context of specific rules and regulations, e.g., HIPAA (Health and Services, 2003) or the European Privacy Directive (1995). First forays in this direction using formal methods have already been made, e.g., Barth, Datta, Mitchell, and Nissenbaum (2006) and Felty and Matwin (2002).

Finally, the increasing abundance and availability of data tracking mobile devices will bring new challenges to the field. People will become potentially identifiable

by knowing the trajectories their mobile devices leave in fixed times and time intervals. Clearly such data, already collected, present an important asset from the public security point of view, but also a very considerable threat from a privacy perspective. There is early work in this area (Gianotti, Pedreschi 2008). Such data are already being collected. This is an important asset for public security, but also a considerable threat for privacy.

Recommended Reading

Agrawal, D., & Aggarwal, C. C. (2001). On the design and quantification of privacy preserving data mining algorithms. *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems*. Santa Barbara, CA: ACM.

Agrawal, R., & Srikant, R. (2000). *Privacy-preserving data mining*, *ACM SIGMOD record* (pp. 439–450).

Atzori, M., Bonchi, F., Giannotti, F., & Pedreschi, D. (2005). k-Anonymous patterns. *Proceedings of the ninth European conference on principles and practice of knowledge discovery in databases (PKDD 05)*. Porto, Portugal.

Barth, A., Datta, A., Mitchell, J. C., & Nissenbaum, H. (2006). Privacy and contextual integrity: Framework and applications. *IEEE Symposium on Security and Privacy*, 184–198.

Clifton, C. W. (2005). *What is privacy? Critical steps for privacy-preserving data mining, workshop on privacy and security aspects of data mining*.

Directive 95/46/EC of the European Parliament on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the European Communities*. (1995).

Domingo-Ferrer, J., Sebé, F., & Solanas, A. (2008). *An anonymity model achievable via microaggregation VLDB workshop on secure data management*. Springer, (pp. 209–218).

Du, W., & Zhan, Z. (2003). Using randomized response techniques for privacy-preserving data mining. *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (Vol. 510).

Evfimievski, A., Srikant, R., Agrawal, R., & Gehrke, J. (2002). Privacy preserving mining of association rules. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 217–228).

Felty, A., & Matwin, S. (2002). Privacy-oriented data mining by proof checking. *Sixth European conference on principles of data mining and knowledge discovery* (Vol. 2431) (pp. 138–149).

Friedman, A., Schuster, A., & Wolff, R. (2006). *k-Anonymous decision tree induction*, *PKDD 2006* (pp. 151–162).

Health, U. D. o., & Services, H. (Eds.) (2003). *Summary of HIPAA privacy rule*.

Gianotti, F., & Pedreschi, D. (2008). Mobility, Data Mining and Privacy: Geographic Knowledge Discovery, Springer.

Iyengar, V. S. (2002). Transforming data to satisfy privacy constraints. *Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 279–288).

Jiang, W., & Atzori, M. (2006). Secure distributed k-Anonymous pattern mining, proceedings of the sixth international conference on data mining. *IEEE Computer Society*.

Kantarcioglu, M. & Clifton, C. (2004). Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*, *16*, 1026–1037.

Kargupta, H., Datta, S., & Wang, Q. (2003). On the privacy preserving properties of random data perturbation techniques. *Third IEEE international conference on data mining. ICDM 2003* (pp. 99–106).

Machanavajjhala, A., Kifer, D., Gehrke, J., & Venkitasubramaniam, M. (2007). L -diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1, 3.

Malin, B. A. (2005). An evaluation of the current state of genomic data privacy protection technology and a roadmap for the future. *Journal of the American Medical Informatics Association*, *12*, 28.

Moor, J. (2004). Towards a theory of privacy in the information age. In T. Bynum, & S. Rodgerson (Eds.), *Computer Ethics and Professional Responsibility*. Blackwell.

Ninghui, L., Tiancheng, L., & Venkatasubramanian, S. (2007). t-Closeness: Privacy beyond k-Anonymity and l-Diversity. *IEEE 23rd international conference on data engineering. ICDE 2007* (pp. 106–115).

Oliveira, S. R. M., Zaïane, O. R., & Saygin, Y. (2004). Secure association rule sharing. *Proceedings of the eighth PAKDD and advances in knowledge discovery and data mining* (pp. 74–850).

Paillier, P. (1999). *The 26th international conference on privacy and personal data protection, advances in cryptography – EUROCRYPT'99* (pp. 23–38).

Rossi, G. (2004). *Privacy as quality in modern economy*, *the 26th international conference on privacy and personal data protection*.

Subramaniam, H., Wright, R. N., & Yang, Z. (2004). Experimental analysis of privacy-preserving statistics computation. *Proceedings of the VLDB worshop on secure data management* (pp. 55–66).

Sweeney, L. (2001). *Computational disclosure control: a primer on data privacy protection*. Cambridge, MA: Massachusetts Institute of Technology, Deptartment of Electrical Engineering and Computer Science.

Vaidya, J., & Clifton, C. (2002). Privacy preserving association rule mining in vertically partitioned data. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 639–644) Edmonton, Alberta: ACM.

Vaidya, J., Clifton, C., Kantarcioglu, M., & Patterson, A. S. (2008). Privacy-preserving decision trees over vertically partitioned data. *ACM Transactions on Knowledge Discovery from Data*, *2*, 1–27.

Vaidya, J., Zhu, Y. M., & Clifton, C. W. (2006). *Privacy preserving data mining*. New York: Springer.

Website of the GeoPKDD Project.(2006).

Westin, A. (1967). *Privacy and freedom*. New York: Atheneum.

Yao, A. (1986). *How to generate and exchange secrets. 27th FOCS*.

Zhan, J., Matwin, S., & Chang, L. (2007). Privacy-preserving collaborative association rule mining. *Journal of Network and Computer Appliactions, 30*, 1216–1227.

Zhan, J. Z. & Matwin, S. (2004). *Privacy-prteserving data mining in electronic surveys*, *ICEB 2004* (pp. 1179–1185).

Probabilistic Context-Free Grammars

Yasubumi Sakakibara
Keio University, Hiyoshi, Kohoku-ku, Japan

Synonyms

PCFG

Definition

In formal language theory, formal grammar (phrase-structure grammar) is developed to capture the generative process of languages (Hopcroft & Ullman, 1979). A formal grammar is a set of productions (rewriting rules) that are used to generate a set of strings, that is, a *language*. The productions are applied iteratively to generate a string, a process called *derivation*. The simplest kind of formal grammar is a *regular* grammar.

Context-free grammars (CFG) form a more powerful class of formal grammars than regular grammars and are often used to define the syntax of programming languages. Formally, a *CFG* consists of a set of nonterminal symbols N, a terminal alphabet Σ, a set P of productions (rewriting rules), and a special nonterminal S called the start symbol. For a nonempty set X of symbols, let X^* denote the set of all finite strings of symbols in X. Every CFG production has the form $S \to \alpha$, where $S \in N$ and $\alpha \in (N \cup \Sigma)^*$. That is, the left-hand side consists of one nonterminal and there is no restriction on the number or placement of nonterminals and terminals on the right-hand side. The *language generated* by a CFG G is denoted $L(G)$.

A *probabilistic context-free grammar* (PCFG) is obtained by specifying a probability for each production for a nonterminal A in a CFG, such that a probability distribution exists over the set of productions for A.

A CFG $G = (N, \Sigma, P, S)$ is in *Chomsky normal form* if each production rule is of the form $A \to BC$ or $A \to a$, where $A, B, C \in N$ and $a \in \Sigma$.

Given a PCFG G and a string $w = a_1 \ldots a_m$, there are three basic problems:

1. Calculating the probability $\Pr(w|G)$ that the grammar G assigns to w
2. Finding the most likely derivation (parse tree) of w by G
3. Estimating the parameters of G to maximize $\Pr(w|G)$

The first two problems, calculating the probability $\Pr(w|G)$ of a given string w assigned by a PCFG G and finding the most likely derivation of w by G, can be solved using dynamic programming methods analogous to the Cocke-Younger-Kasami or Early parsing methods. A polynomial-time algorithm for solving the second problem is known as *Viterbi* algorithm, and a polynomial-time algorithm for the third problem is known as the *inside-outside* algorithm (Lari & Young, 1990).

Derivation Process

A *derivation* is a rewriting of a string in $(N \cup \Sigma)^*$ using the production rules of a CFG G. In each step of the derivation, a nonterminal from the current string is chosen and replaced with the right-hand side of a production rule for that nonterminal. This replacement process is repeated until the string consists of terminal symbols only. If a derivation begins with a nonterminal A and derives a string $\alpha \in (N \cup \Sigma)^*$, we write $A \Rightarrow \alpha$.

For example, the grammar in Fig. 1 generates an RNA sequence AGAAACUUGCUGGCCU by the following derivation: Beginning with the start symbol S, any production with S left of the arrow can be chosen to replace S. If the production $S \to \mathrm{A}X_1\mathrm{U}$ is selected (in this case, this is the only production available), the effect is to replace S with $\mathrm{A}X_1\mathrm{U}$. This one derivation step is written $S \Rightarrow \mathrm{A}X_1\mathrm{U}$, where the double arrow signifies application of a production. Next, if the production $X_1 \to \mathrm{G}X_2\mathrm{C}$ is selected, the derivation step is $\mathrm{A}X_1\mathrm{U} \Rightarrow \mathrm{AG}X_2\mathrm{CU}$. Continuing with similar derivation operations, each time choosing a nonterminal symbol and replacing it with the right-hand side of an appropriate production, we obtain the following derivation terminating with the desired sequence:

$$
\begin{aligned}
S &\Rightarrow \mathrm{A}X_1\mathrm{U} \Rightarrow \mathrm{AG}X_2\mathrm{CU} \Rightarrow \mathrm{AG}X_3X_4\mathrm{CU} \\
&\Rightarrow \mathrm{AGA}X_5\mathrm{U}X_4\mathrm{CU} \Rightarrow \mathrm{AGAA}X_6\mathrm{U}\mathrm{U}X_4\mathrm{CU} \\
&\Rightarrow \mathrm{AGAAACUU}X_4\mathrm{CU} \Rightarrow \mathrm{AGAAACUUG}X_{15}\mathrm{CCU} \\
&\Rightarrow \mathrm{AGAAACUUGC}X_{16}\mathrm{GCCU} \\
&\Rightarrow \mathrm{AGAAACUUGCUGGCCU}.
\end{aligned}
$$

$$
\begin{aligned}
G_{rna} &= (N, \Sigma, P, S) \\
N &= \{S, X_1, \ldots, X_{16}\}, \quad \Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\} \\
P &= \left\{ \begin{array}{llll}
S \to \mathtt{A}X_1\mathtt{U}, & & & \\
X_1 \to \mathtt{G}X_2\mathtt{C}, & X_2 \to X_3X_4, & X_3 \to \mathtt{A}X_5\mathtt{U}, & X_4 \to \mathtt{G}X_{15}\mathtt{C}, \quad X_5 \to \mathtt{A}X_6\mathtt{U}, \\
X_6 \to \mathtt{AC}, & X_6 \to X_7X_8, & X_7 \to \mathtt{A}X_9\mathtt{U} & X_8 \to \mathtt{G}X_{12}\mathtt{C}, \quad X_9 \to \mathtt{G}X_{10}\mathtt{C}, \\
X_{10} \to \mathtt{A}X_{11}, & X_{11} \to \mathtt{UG}, & X_{12} \to \mathtt{A}X_{13}\mathtt{U}, & X_{13} \to \mathtt{A}X_{14}, \quad X_{14} \to \mathtt{GC}, \\
X_{15} \to \mathtt{C}X_{16}\mathtt{G}, & X_{16} \to \mathtt{UG} & &
\end{array} \right\}
\end{aligned}
$$

Probabilistic Context-Free Grammars. Figure 1. This set of productions *P* generates RNA sequences with a certain restricted structure. *S*, $X_1, \ldots, X_{16}$ are nonterminals; A, U, G, and C are terminals representing the four nucleotides. Note that only for X_6 is there a choice of productions

Such a derivation can be arranged in a tree structure called a *parse tree*.

The *language generated* by a CFG G is denoted $L(G)$, that is, $L(G) = \{w \mid S \Rightarrow w, w \in \Sigma^*\}$. Two CFGs G and G' are said to be *equivalent* if and only if $L(G) = L(G')$.

Probability Distribution

A PCFG assigns a probability to each string which it derives and hence defines a probability distribution on the set of strings. The probability of a derivation can be calculated as the product of the probabilities of the productions used to generate the string. The probability of a string w is the sum of probabilities over all possible derivations that could generate w, written as follows:

$$
\begin{aligned}
\Pr(w \mid G) &= \sum_{\text{all derivations } d} \Pr(S \overset{d}{\Rightarrow} w \mid G) \\
&= \sum_{\alpha_1, \ldots, \alpha_n} \Pr(S \Rightarrow \alpha_1 \mid G) \cdot \Pr(\alpha_1 \Rightarrow \alpha_2 \mid G) \\
&\quad \cdot \cdots \cdot \Pr(\alpha_n \Rightarrow w \mid G).
\end{aligned}
$$

Parsing Algorithm

Efficiently computing the probability of a string w, $\Pr(s \mid G)$, presents a problem because the number of possible derivations for w is exponential in the length of the string. However, a dynamic programming technique analogous to the Cocke-Kasami-Young or Earley methods for nonprobabilistic CFGs can accomplish this task efficiently (in time proportional to the cube of the length of w).

The CYK algorithm is a polynomial time algorithm for solving the parsing (membership) problem of CFGs using dynamic programming. The CYK algorithm assumes Chomsky normal form of CFGs, and the essence of the algorithm is the construction of a triangular *parse table* T. Given a CFG $G = (N, \Sigma, P, S)$ and an input string $w = a_1 a_2 \ldots a_n$ in Σ^* to be parsed according to G, each element of T, denoted $t_{i,j}$, for $1 \le i \le n$ and $1 \le j \le n - i + 1$, has a value which is a subset of N. The interpretation of T is that a nonterminal A is in $t_{i,j}$ if and only if $A \Rightarrow a_i a_{i+1} \ldots a_{i+j-1}$, that is, A derives the substring of w beginning at position i and of length j. To determine whether the string w is in $L(G)$, the algorithm computes the parse table T and look to see whether S is in entry $t_{1,n}$.

In the first step of constructing the parse table, the CYK algorithm sets $t_{i,1} = \{A \mid A \to a_i \text{ is in } P\}$. In the jth step, the algorithm assumes that $t_{i,j'}$ has been computed for $1 \le i \le n$ and $1 \le j' < j$, and it computes $t_{i,j}$ by examining the nonterminals in the following pairs of entries:

$$
(t_{i,1}, t_{i+1,j-1}), (t_{i,2}, t_{i+2,j-2}), \ldots, (t_{i,j-1}, t_{i+j-1,1})
$$

and if B is in $t_{i,k}$ and C is in $t_{i+k,j-k}$ for some k $(1 \le k < j)$ and the production $A \to BC$ is in P, A is added to $t_{i,j}$.

For example, we consider a simple CFG $G = (N, \Sigma, P, S)$ of Chomsky normal form where $N = \{S, A\}$, $\Sigma = \{a, b\}$ and

$$
P = \{S \to AA,\ S \to AS,\ S \to b,\ A \to SA,\ A \to a\}.
$$

This CFG generates a string "*abaaa*," that is, $S \Rightarrow abaaa$, and the parse table T for *abaaa* is shown in Fig. 2. The parse table can efficiently store all possible parse trees of G for *abaaa*.

Learning

The problem of learning PCFGs from example strings has two aspects: determining a discrete structure (topology) of the target grammar and estimating probabilistic parameters in the grammar (Sakakibara, 1997). Based

on the maximum likelihood criterion, an efficient estimation algorithm for probabilistic parameters has been proposed: the inside-outside algorithm for PCFGs. On the other hand, finding an appropriate discrete structure of a grammar is a harder problem.

The procedure to estimate the probabilistic parameters of a PCFG is known as the *inside-outside* algorithm. Just like the forward-backward algorithm for HMMs, this procedure is an expectation-maximization (EM) method for obtaining maximum likelihood of the grammar's parameters. However, it requires the grammar to be in Chomsky normal form, which is inconvenient to handle in many practical problems (and requires more nonterminals). Further, it takes time at least proportional to n^3, whereas the forward-backward procedure for HMMs takes time proportional to n^2, where n is the length of the string w. There are also many local maxima in which the method can get caught. Therefore, the initialization of the iterative process is crucial since it affects the speed of convergence and the goodness of the results.

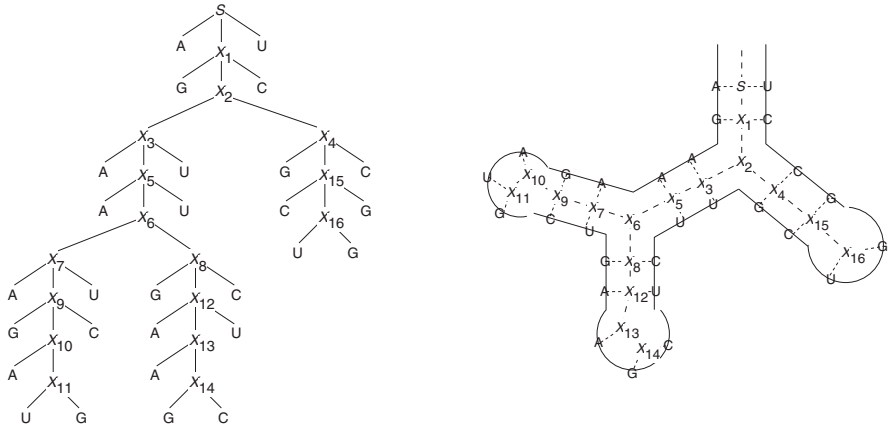
5	S, A				
4	S, A	S, A			
3	S, A	S	S, A		
2	S	A	S	S	
$j = 1$	A	S	A	A	A
	$i = 1$	2	3	4	5
	a	b	a	a	a

Probabilistic Context-Free Grammars. Figure 2. The parse table *T* of *G* for "*abaaa*"

Application to Bioinformatics

An effective method for learning and building PCFGs has been applied to modeling a family of RNA sequences (Durbin, Eddy, Krogh, & Mitchison, 1998; Sakakibara, 2005). In RNA, the nucleotides adenine (A), cytosine (C), guanine (G), and uracil (U) interact in specific ways to form characteristic secondary-structure motifs such as helices, loops, and bulges. In general, the folding of an RNA chain into a functional molecule is largely governed by the formation of intramolecular A-U and G-C Watson–Crick pairs. Such base pairs constitute the so-called biological palindromes in a genome and can be clearly described by a CFG. In particular, productions of the forms $X \to$ A Y U, $X \to$ U Y A, $X \to$ G Y C, and $X \to$ C Y G describe a structure in RNA due to Watson–Crick base pairing. Using productions of this type, a CFG can specify a language of biological palindromes.

For example, the application of productions in the grammar shown in Fig. 1 generates the RNA sequence CAUCAGGGAAGAUCUCUUG and the derivation can be arranged in a tree structure of a *parse tree* (Fig. 3, left). A parse tree represents the syntactic structure of a sequence produced by a grammar. For the RNA sequence, this syntactic structure corresponds to the physical secondary structure (Fig. 3, right). PCFGs are applied to perform three tasks in RNA sequence analysis: to discriminate RNA-family sequences from nonfamily sequences, to produce multiple alignments, and to ascertain the secondary structure of new sequences.



Probabilistic Context-Free Grammars. Figure 3. A parse tree (*left*) generated by a simple context-free grammar (CFG) for RNA molecules and the physical secondary structure (*right*) of the RNA sequence which is a reflection of the parse tree

Recommended Reading

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis*. Cambridge, UK: Cambridge University Press.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to automata theory, languages and computation*. Reading, MA: Addison-Wesley.

Lari, K., & Young, S. J. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language, 4*, 35–56.

Sakakibara, Y. (1997). Recent advances of grammatical inference. *Theoretical Computer Science, 185*, 15–45.

Sakakibara, Y. (2005). Grammatical inference in bioinformatics. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*, 1051–1062.

Probably Approximately Correct Learning

►PAC Learning

Process-Based Modeling

►Inductive Process Modeling

Program Synthesis From Examples

►Inductive Programming

Programming by Demonstration

Pierre Flener[1,2], Ute Schmid[3]
[1]Sabancı University, Orhanlı, Tuzla, Turkey
[2]Uppsala University, Uppsala, Sweden
[3]University of Bamberg, Bamberg, Germany

Synonyms

Programming by example

Definition

Programming by demonstration (PBD) describes a collection of approaches for the support of end-user programming with the goal of making the power of computers fully accessible to all users. The general objective is to *teach* computer systems new behavior by demonstrating (repetitive) actions on concrete examples. A user provides examples of how a program should operate, either by demonstrating trace steps or by showing examples of the inputs and outputs, and the system infers a generalized program that achieves those examples and can be applied to new examples. Typical areas of application are macro generation (e.g., for text editing), simple arithmetic functions in spreadsheets, simple shell programs, XML transformations, or query-replace commands, as well as the generation of helper programs for web agents, geographic information systems, or computer-aided design. The most challenging approach to PBD is to obtain generalizable examples by minimal intrusion, where the user's ongoing actions are recorded without an explicit signal for the start of an example and without explicit confirmation or rejection of hypotheses. An example of such a system is EAGER (Cypher, 1993a).

Current PBD approaches incorporate some simple forms of ►generalization learning, but typically no or only highly problem-dependent methods for the induction of loops or recursion from examples or traces of repetitive commands. Introducing ►inductive programming or ►trace-based programming methods into PBD applications could significantly increase the possibilities of end-user programming support.

Acknowledgement

Most of the work by this author was done while on leave of absence in 2006/2007 as a Visiting Faculty Member and Erasmus Exchange Teacher at Sabancı University.

Cross References

►Inductive Programming
►Trace-Based Programming

Recommended Reading

Cypher, A. (1993a). Programming repetitive tasks by demonstration. In A. Cypher (Ed.), *Watch what I do: Programming by demonstration* (pp. 205–217). Cambridge, MA: MIT Press.

Cypher A. (Ed.) (1993b). *Watch what I do: Programming by demonstration*. Cambridge, MA: MIT Press.

Lieberman, H. (Ed.) (2001). *Your wish is my command: Programming by example*. San Francisco, CA: Morgan Kaufmann.

Programming by Example

►Programming by Demonstration

Programming from Traces

►Trace-Based Programming

Projective Clustering

Cecilia M. Procopiuc
AT&T Labs, Florham Park, NJ, USA

Synonyms

Local feature selection; Subspace clustering

Definition

Projective clustering is a class of problems in which the input consists of high-dimensional data, and the goal is to discover those subsets of the input that are strongly correlated in subspaces of the original space. Each subset of correlated points, together with its associated subspace, defines a *projective cluster*. Thus, although all cluster points are close to each other when projected on the associated subspace, they may be spread out in the full-dimensional space. This makes projective clustering algorithms particularly useful when mining or indexing datasets for which full-dimensional clustering is inadequate (as is the case for most high-dimensional inputs). Moreover, such algorithms compute projective clusters that exist in different subspaces, making them more general than global dimensionality-reduction techniques.

Motivation and Background

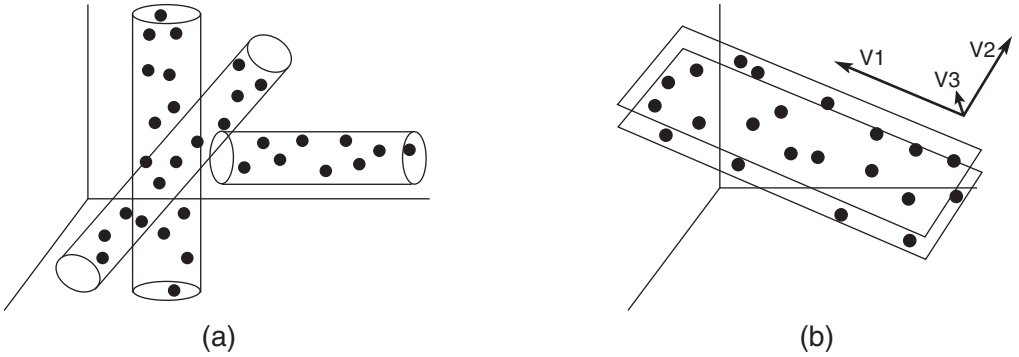
Projective clustering is a type of data mining whose main motivation is to discover correlations in the input data that exist in subspaces of the original space. This is an extension of traditional full-dimensional clustering, in which one tries to discover point subsets that are strongly correlated in all dimensions. Figure 1a shows an example of input data for which full-dimensional clustering cannot discover the three underlying patterns. Each pattern is a projective cluster.

It is well known (Beyer, Goldstein, Ramakrishnan, & Shaft, 1999) that for a broad class of data distributions, as the dimensionality increases, the distance to the nearest neigbor of a point approaches the distance to its farthest neighbor. This implies that full-dimensional clustering will fail to discover significantly correlated subsets on such data, since the diameter of a cluster is almost the same as the diameter of the entire dataset. In practice, many applications from text and image processing generate data with hundreds or thousands of dimensions, which makes them extremely bad candidates for full-dimensional clustering methods.

One popular technique to classify high-dimensional data is to first project it onto a much lower-dimensional subspace, and then employ a full-dimensional clustering algorithm in that space. The projection subspace is the same for all points, and is computed so that it best "fits" the data. A widely used dimensionality-reduction technique, called ►principal component analysis (PCA), defines the best projection subspace to be the one that minimizes least-square error. While this approach has been proven successful in certain areas such as text mining, its effectiveness depends largely on the characteristics of the data. The reason is that there may be no way to choose a single projection subspace without encountering a significant error; or alternatively, setting a maximum bound on the error results in a subspace with high dimensionality. Figure 1b shows the result of PCA on a good candidate set. The points are projected on the subspace spanned by vectors $V1$ and $V2$, along which they have greatest variance. However, for the example in Fig.1a, no plane or line fits the data well enough. Projective clustering can thus be viewed as a generalized dimensionality-reduction method, in which different subsets of the data are projected on different subspaces.

There are many variants of projective clustering, depending on what quality measure one tries to optimize for the clustering. Most such measures, however, are expressed as a function of the distances between points in the clusters. The distance between two cluster points is computed with respect to the subspace associated with that cluster. Alternative quality measures consider the density of cluster points inside the associated subspace.

Meggido and Tamir (1982) showed that it is NP-Hard to decide whether a set of n points in the plane can be covered by k lines. This early result implies not only that most projective clustering problems are NP-Complete even in the planar case, but also that



Projective Clustering. Figure 1. Dimensionality reduction via (a) projective clustering and (b) principal component analysis

approximating the objective function within a constant factor is NP-Complete. Nevertheless, several approximation algorithms have been proposed, with running time polynomial in the number of points n and exponential in the number of clusters k. Agrawal, Gehrke, Gunopulos, and Raghavan (1998) proposed a subspace clustering method based on density measure that computes clusters in a bottom-up approach (from lower to higher dimensions). Aggarwal, Wolf, Yu, Procopiuc, and Park (1999) designed a partitioning-style algorithm.

Theory

Many variants of projective clustering problems use a distance-based objective function and thus have a natural geometric interpretation. In general, the optimization problem is stated with respect to one or more parameters that constrain the kind of projective clusters one needs to investigate. Examples of such parameters are: the number of clusters, the dimensionality (or average dimensionality) of the clusters, the maximum size of the cluster in its associated subspace, the minimum density of cluster points, etc. Below we present the most frequently studied variants for this problem.

Distance-Based Projective Clustering Given a set S of n points in $\mathbb{R}^d$ and two integers $k < n$ and $q \leq d$, find k q-dimensional flats $h_1, \ldots, h_k$ and partition S into k subsets $C_1, \ldots, C_k$ so that one of the following objective functions is minimized:

$$\max_{1 \leq i \leq k} \max_{p \in C_i} d(p, h_i) \qquad (k\text{-center})$$

$$\sum_{1 \leq i \leq k} \sum_{p \in C_i} d(p, h_i) \qquad (k\text{-median})$$

$$\sum_{1 \leq i \leq k} \sum_{p \in C_i} d^2(p, h_i) \qquad (k\text{-means})$$

These types of problems are also referred to as *geometric clustering problems*. They require all cluster subspaces to have the same dimensionality, i.e., $d - q$ (the subspace associated with C_i is orthogonal to h_i). The number of clusters is also fixed, and the clustering must be a partitioning of the original points.

Further variants are defined by introducing slight modifications in the above framework. For example, one can allow the existence of outliers, i.e., points that do not belong to any projective cluster. This is generally done by providing an additional parameter, which is the maximum percentage of outliers. The problems can also be changed to a dual formulation, in which a maximum value for the objective function is specified, and the goal is to minimize the number of clusters k.

Special cases for the k-center objective function are $q = d - 1$ and $q = 1$. In the first case, the problem is equivalent to finding k hyper-strips that contain S so that the maximum width of a hyper-strip is minimized. If $q = 1$, then the problem is to cover S by k congruent hyper-cylinders of smallest radius. Since this is equivalent to finding the k lines that are the axes of the hyper-cylinders, this problem is also referred to as *k-line-center*. Figure 1a is an example of 3-line-center.

In addition, k-median problems have also been studied when cluster subspaces have different dimensionalities. In that case, distances computed in each cluster are normalized by the dimensionality of the corresponding subspace.

Density-Based Projective Clustering A convex region in a subspace is called dense if the number of data points that project inside it is larger than some user-defined threshold. For a fixed subspace, the convex regions of interest in that subspace are defined in one of several ways, as detailed below. Projective clusters are then defined to be connected unions of dense regions of interest. The different variants for defining regions of interest can be broadly classified in three classes:

(ε-Neighborhoods) Regions of interest are L_p-balls of radius ε centered at the data points. In general, L_p is either L_2 (hyper-spheres) or L_∞ (hyper-cubes).

(Regular Grid Cells) Regions of interest are cells defined by an axis-parallel grid in the subspace. The grid hyper-planes are equidistant along each dimension.

(Irregular Grid Cells) Regions of interest are cells defined by an irregular grid in the subspace. Parallel grid hyper-planes are not necessarily equidistant, and they may also be arbitrarily oriented.

Another variant of projective clustering defines a so-called *quality measure* for a projective cluster, which depends both on the number of cluster points and the number of dimensions in the associated subspace. The goal is to compute the clusters that maximize this measure. Projective clusters are required to be L_p-balls of fixed radius in their associated subspace, which means that clusters in higher dimensions tend to have fewer points, and vice-versa. Hence, the quality measure provides a way to compare clusters that exist in different number of dimensions. It is related to the notion of dense ε-neighborhoods.

Many other projective clustering problems are application driven and do not easily fit in the above classification. While they follow the general framework of finding correlations among data in subspaces of the original space, the notion of projective cluster is specific to the application. One such example is presented later in this section.

Algorithms

Distance-based projective clustering problems are NP-Complete when the number of clusters k is an input parameter. Moreover, k-center problems cannot be approximated within a constant factor, unless $P = NP$. This follows from the result of Meggido and Tamir (1982), who showed that it is NP-Hard to decide whether a set of n points in the plane can be covered by k lines.

Agarwal and Procopiuc (2003) first proposed approximation algorithms for k-center projective clustering in two and three dimensions. The algorithms achieve constant factor approximation by generating more clusters than required.

Subsequent work by several other authors led to the development of a general framework in which $(1 + \varepsilon)$-approximate solutions can be designed for several types of full-dimensional and projective clustering. In particular, k-center and k-means projective clustering can be approximated in any number of dimensions. The idea is to compute a so-called *coreset*, which is a small subset of the points, such that the optimal projective clusters for the coreset closely approximate the projective clusters for the original set. Computing the optimal solution for the coreset has (super) exponential dependence on the number of clusters k, but it is significantly faster than computing the optimal solution for the original set of points. The survey by Agarwal, Har-Peled, and Varadarajan (2005) gives a comprehensive overview of these results.

While the above algorithms have approximation guarantees, they are not practical even for moderate values of n, k, and d. As a result, heuristic methods have also been developed for these problems. The general approach is to iteratively refine a current set of clusters, either by re-assigning points among them, or by merging nearby clusters. When the set of points in a cluster changes, the new subspace associated with the cluster is also recomputed, in a way that tries to optimize the objective function for the new clustering. Aggarwal et al. (1999) proposed the PROCLUS algorithm for k-median projective clustering with outliers. The cluster subspaces can have different dimensionalities, but they must be orthogonal to coordinate axes. Aggarwal and Yu (2000) subsequently extended the algorithm to arbitrarily oriented clusters, but with the same number of dimensions. Agarwal and Mustafa (2004) proposed a heuristic approach for k-means projective clustering with arbitrary orientation and different dimensionalities.

The first widely used method for density-based projective clustering was proposed by Agrawal et al. (1998). The algorithm, called CLIQUE, computes projective

clusters based on regular grid cells in orthogonal subspaces, starting from the lowest-dimensional subspaces (i.e., the coordinate axes) and iterating to higher dimensions. Pruning techniques are used to skip subspaces in which a large fraction of points lie outside dense regions. Subsequent strategies improved the running time and accuracy by imposing irregular grids and using different pruning criteria.

Bohm, Kailing, Kroger, and Zimek (2004) designed an algorithm called 4C for computing density-connected ε-neighborhoods in arbitrarily oriented subspaces. The method is agglomerative: It computes the local dimensionality around each point p by using PCA on all points inside the (full-dimensional) ε-neighborhood of p. If the dimensionality is small enough and the neighborhood is dense, then p and its neighbors form a projective cluster. Connected projective clusters with similarly oriented subspaces are then repeatedly merged.

The OptiGrid algorithm by Hinneburg and Keim (1999) was the first method to propose irregular grid cells of arbitrary (but fixed) orientation. Along each grid direction, grid hyper-planes are defined to pass through the local minima of a probability density function. This significantly reduces the number of cells compared with a regular grid that achieves similar overall accuracy. The probability density function is defined using the kernel-density estimation framework. Input points are projected on the grid direction, and their distribution is extrapolated to the entire line by the density function

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - s_i}{h}\right),$$

where $s_1, \ldots, s_n$ denote the projections of the input points, and h is a parameter. The function $K(x)$, called the *kernel*, is usually the Gaussian function, although other kernels can also be used.

The DOC algorithm proposed by Procopiuc, Jones, Agarwal, and Murali (2002) approximates optimal clusters for a class of quality measures. Orthogonal projective clusters are computed iteratively via random sampling. If a sample is fully contained in a cluster then it can be used to determine the subspace of that cluster, as well as (a superset of) the other cluster points. Such a sample is called a *discriminating set*. Using the properties of the quality measure, the authors show that a discriminating set is found with high probability after a polynomial number of trials.

An overview of most of these practical methods, as well as of subsequent work expanding their results, can be found in the survey by Parsons, Haque, and Liu (2004).

Applications

Similar to full-dimensional clustering, projective clustering methods provide a way to efficiently organize databases for searching, as well as for pattern discovery and data compression. In a broad sense, they can be used in any application that handles high-dimensional data, and which can benefit from indexing or mining capabilities. In practice, additional domain-specific information is often necessary. We present an overview of the generic database usage first, and then discuss several domain-specific applications.

Data Indexing An index tree is a hierarchical structure defined on top of a data set as follows. The root corresponds to the entire data set. For each internal node, the data corresponding to that node is partitioned in some pre-defined manner, and there is a child of the node corresponding to each subset in the partition. Often, the partitioning method is a distance-based clustering algorithm. In addition, each node stores the boundary of a geometric region that contains its points, to make searching the structure more efficient. For many popular indexes, the geometric region is the minimum axis-parallel bounding box. Index trees built with full-dimensional clustering methods become inefficient for dimensionality about 10 or higher, due to the large overlap in the geometric regions of sibling nodes. Chakrabarti and Mehrotra (2000) first proposed an index tree that uses projective clustering as a partitioning method. In that case, each node also stores the subspace associated with the cluster.

Pattern Discovery A projective cluster, by definition, is a pattern in the data, so any of the above algorithms can be used in a pattern discovery application. However, most applications restrict the projective clusters to

be orthogonal to coordinate axes, since the axes have special interpretations. For example, in a database of employees, one axis may represent salary, another the length of employment, and the third one the employees' age. A projective cluster in the subspace spanned by salary and employment length has the following interpretation: there is a correlation between salaries in range A and years of employment in range B, which is independent of employees' age.

Data Compression As discussed in the introduction, projective clusters can be used as a dimensionality-reduction technique, by replacing each point with its projection on a lower dimensional subspace. The projection subspace is orthogonal to the subspace of the cluster that contains the point. In general, this method achieves smaller information loss and higher compression ratio than a global technique such as PCA.

Image Processing A picture can be represented as a high-dimensional data point, where each pixel represents one dimension, and its value is equal to the RGB color value of the pixel. Since this representation loses pixel adjacency information, it is generally used in connection with a smoothing technique, which replaces the value of a pixel with a function that depends both on the old pixel value, and the values of its neighbors. A projective cluster groups images that share some similar features, while they differ significantly on others. The DOC algorithm has been applied to the face detection problem as follows: Projective clusters were computed on a set of (pre-labeled) human faces, then used in a classifier to determine whether a new image contained a human face.

Document Processing Text documents are often represented as sparse high-dimensional vectors, with each dimension corresponding to a distinct word in the document collection. Several methods are used to reduce the dimensionality, e.g., by eliminating so-called stop words such as "and," "the," and "of." A non zero entry in a vector is usually a function of the corresponding word's frequency in the document. Because of the inherent sparsity of the vectors, density-based clustering, as well as k-center methods, are poor choices for such data. However, k-means projective clustering has been successfully applied to several document corpora (Li, Ma, & Ogihara, 2004).

DNA Microarray Analysis A gene-condition expression matrix, generated by a DNA microarray, is a real-valued matrix, such that each row corresponds to a gene, and each column corresponds to a different condition. An entry in a row is a function of the relative abundance of the mRNA of the gene under that specific condition. An orthogonal projective cluster thus represents several genes that have similar expression levels under a subset of conditions. Genetics researchers can infer connections between a disease and the genes in a cluster. Due to the particularities of the data, different notions of similarity are often required. For example, order preserving clusters group genes that have the same tendency on a subset of attributes, i.e., an attribute has the same rank (rather than similar value) in each projected gene. See the results of Liu and Wang (2003).

Principal Component Analysis

PCA also referred to as the Karhunen-Loève Transform, is a global ►dimensionality reduction technique, as opposed to projective clustering, which is a local dimensionality reduction method. PCA is defined as an orthogonal linear transformation with the property that it transforms the data into a new coordinate system, such that the projection of the data on the first coordinate has the greatest variance among all projections on a line, the projection of the data on the second coordinate has the second greatest variance, and so on. Let X denote the data matrix, with each point written as a column vector in X, and modified so that X has empirical mean zero (i.e., the mean vector is subtracted from each data point). Then the eigenvectors of the matrix XX^T are the coordinates of the new system. To reduce the dimensionality, keep only the eigenvectors corresponding to the largest few eigenvalues.

Coresets

Let $P \subseteq \mathbb{R}^d$ be a set of points, and μ be a measure function defined on subsets of $\mathbb{R}^d$, such that μ is monotone (i.e., for $P_1 \subseteq P_2$, $\mu(P_1) \leq \mu(P_2)$). A subset $Q \subseteq P$ is

an ε-*coreset* with respect to μ if $(1-\varepsilon)\mu(P) \leq \mu(Q)$. The objective functions for k-center, k-median, and k-means projective clustering are all examples of measure functions μ.

Cross References

►Clustering
►Curse of Dimensionality
►Data Mining
►Dimensionality Reduction
►k-Means Clustering
►Kernel Methods
►Principal Component Analysis

Recommended Reading

Agarwal, P. K., Har-Peled, S., & Varadarajan, K. R. (2005). Geometric approximation via coresets. *Combinatorial and Computational Geometry* (pp. 1–30).

Agarwal, P. K., & Mustafa, N. (2004). k-means projective clustering. In *Proceeding of ACM SIGMOD-SIGACT-SIGART symposium principles of database systems* (pp. 155–165).

Agarwal, P. K., & Procopiuc, C. M. (2003). Approximation algorithms for projective clustering. *Journal of Algorithms*, *46*(2), 115–139.

Agrawal, R., Gehrke, J., Gunopulos, D., & Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. In *Proceeding of ACM SIGMOD international conference management of data* (pp. 94–105).

Aggarwal, C. C., Procopiuc, C. M., Wolf, J. L., Yu, P. S., & Park, J. S. (1999). Fast algorithms for projected clustering. In *Proceeding of ACM SIGMOD international conference management of data* (pp. 61–72).

Aggarwal, C. C., & Yu, P. S. (2000). Finding generalized projected clusters in high dimensional spaces. In *Proceeding of ACM SIGMOD international conference management of data* (pp. 70–81).

Beyer, K., Goldstein, J., Ramakrishnan, R., & Shaft, U. (1999). When is "nearest neighbour" meaningful? In *Proceeding of 7th international conference data theory* (Vol. 1540, pp. 217–235).

Böhm, C., Kailing, K., Kröger, P., & Zimek, A. (2004). Computing clusters of correlation connected objects. In *Proceeding of ACM SIGMOD international conference management of data* (pp. 455–466).

Chakrabarti, K., & Mehrotra, S. (2000). Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proceeding of 26th international conference very large data bases* (pp. 89–100).

Hinneburg, A., & Keim, D. A. (1999). Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *Proceeding of 25th international conference very large data bases* (pp. 506–517).

Li, T., Ma, S., & Ogihara, M. (2004). Document clustering via adaptive subspace iteration. In *Proceeding of 27th international ACM SIGIR conference research and development in information retrieval* (pp. 218–225).

Liu, J., & Wang, W. (2003). Op-cluster: Clustering by tendency in high dimensional space. In *Proceeding of international conference on data mining* (pp. 187–194).

Megiddo, N., & Tamir, A. (1982). On the complexity of locating linear facilities in the plane. *Operations Research Letters*, *1*, 194–197.

Parsons, L., Haque, E., & Liu, H. (2004). Subspace clustering for high dimensional data: A review. *ACM SIGKDD Explorations Newsletter*, *6*(1), 90–105.

Procopiuc, C. M., Jones, M., Agarwal, P. K., & Murali, T. M. (2002). A monte carlo algorithm for fast projective clustering. In *Proceeding of ACM SIGMOD international conference management of data* (pp. 418–427).

Prolog

Prolog is a declarative programming language based on logic. It was conceived by French and British computer scientists in the early 1970s. A considerable number of public-domain and commercial Prolog interpreters are available today. Prolog is particularly suited for applications requiring pattern matching or search. Prolog programs are also referred to as ►logic programs.

In machine learning, classification rules for structured individuals can be expressed using a subset of Prolog. Learning Prolog programs from examples is called ►inductive logic programming (ILP). ILP systems are sometimes – but not always – implemented in Prolog. This has the advantage that classification rules can be executed directly by the Prolog interpreter.

Cross References

►Clause
►First-Order Logic
►Inductive Logic Programming
►Logic Program

Recommended Reading

Colmerauer, A., Kanoui, H., Pasero, R., & Roussel, P. (1973) *Un système de communication homme-machine an Français*. Rep., Groupè d'Intelligence Artificielle, Univ. d'Aix Marseille II. Luminy, France.

Kowalski, R. A. (1972) The predicate calculus as a programming language. In *Proceedings of the International Symposium and Sum-*

mer School on Mathematical Foundations of Computer Science. Jablonna, Poland.
Roussel, P. (1975). *Prolog: Manual de reference et d'utilization*. Technical report, Groupe d'Intelligence Artificielle, Marseille-Luminy.

Property

►Attribute

Propositional Logic

Propositional logic is the logic of propositions, i.e., expressions that are either true or false. Complex propositions are built from propositional atoms using logical connectives. Propositional logic is a special case of predicate logic, where all ►predicates have zero arity; see the entry on first-order logic for details.

Cross References

►First-Order Logic
►Propositionalization

Propositionalization

Nicolas Lachiche
University of Strasbourg, Strasbourg, France

Definition

Propositionalization is the process of explicitly transforming a ►Relational dataset into a propositional dataset.

The input data consists of examples represented by structured terms (cf. ►Learning from Structured Data), several predicates in ►First-Order Logic, or several tables in a relational database. We jointly refer to these as *relational representations*. The output is an ►Attribute-value representation in a single table, where each example corresponds to one row and is described by its values for a fixed set of attributes. New attributes are often called features to emphasize that they are built from the original attributes. The aim of propositionalization is to pre-process relational data for subsequent analysis by attribute-value learners. There are several reasons for doing this, the most important of which are: to reduce the complexity and speed up the learning; to separate modeling the data from hypothesis construction; or to use familiar attribute-value (or propositional) learners.

Motivation and Background

Most domains are naturally modeled by several tables in a relational database or several classes in an object-oriented language, for example: customers and their transactions; molecules, their atoms and bonds; or patients and their examinations. A proper relational dataset involves at least two tables linked together. Typically, one table of the relational representation corresponds to the individuals of interest for the machine learning task, and the other tables contain related information that could be useful. The first table is the individual, or the primary table, the other tables are complementary tables.

Example 1 *Let us consider a simplified medical domain as an example. This is inspired by a real medical dataset (Tomečková, Rauch, & Berka, 2002). It consists of four tables.*

The patient table is the primary table. It contains data on each patient such as the patient identifier (pid), name, date of birth, height, job, the identifier of the company where the patient works, etc.:

Patient

pid	name	birth	height	job	company	...
I	Smith	15/06/1956	1.67	manager	a	...
II	Blake	13/02/1968	1.82	salesman	a	...
⋮	⋮	⋮	⋮	⋮	⋮	...

The company table contains its name, its location, and so on. There is a many-to-one relationship from the patient table to the company table: A patient works for a single company, but a company may have several employees.

The examination table contains the information on all examinations of all patients. For each examination, its identifier (eid), the patient identifier (pid), the date,

Company

cid	name	location	...
a	Eiffel	Paris	...
⋮	⋮	⋮	...

the patient's weight, whether the patient smokes, his or her blood pressure, etc. are recorded. Of course, each examination corresponds to a single patient, and a given patient can have several examinations, i.e., there is a one-to-many relationship from the patient table to the examination table.

Additional tests can be prescribed at each examination. Their identifiers (tid), corresponding examinations (eid), names, values, and interpretations are recorded in the additional_test table:

Examination

eid	pid	date	weight	smokes	BP	...
1	I	10/10/1991	60	yes	10	...
2	I	04/06/1992	64	yes	12	...
⋮	⋮	⋮	⋮	⋮	⋮	...
23	II	20/12/1992	80	yes	10	...
24	II	15/11/1993	78	no	11	...
⋮	⋮	⋮	⋮	⋮	⋮	...

Additional_test

tid	eid	date	name	value	interpretation
t237	1	19/10/1991	red blood cells	35	bad
t238	1	23/10/1991	radiography	nothing	good
⋮	⋮	⋮	⋮	⋮	⋮
t574	2	07/06/1992	red blood cells	43	good
⋮	⋮	⋮	⋮	⋮	⋮

Several approaches exist to deal directly with relational data, e.g., ►Inductive Logic Programming, ►Relational Data Mining (Džeroski & Lavrač, 2001), or ►Statistical Relational Learning. However, if the relational representation does not require recursion or complex quantifiers, relational hypotheses can be transformed into propositional expressions.

Generally, a richer representation language permits the description of more complex concepts, however, the cost of this representational power is that the search space for learning greatly increases. Therefore, mapping a relational representation into a propositional one generally reduces search complexity.

A second motivation of propositionalization is to focus on the construction of features before combining them into an hypothesis (Srinivasan, Muggleton, King, & Theories, 1996). This is related to Feature Construction, and to the use of background knowledge. One could say that propositionalization aims at building an intermediate representation of the data in order to simplify the hypothesis subsequently found by a propositional learner.

A third motivation is pragmatic. Most available machine learning systems deal with propositional data only, but tend to include a range of algorithms in a single environment, whereas relational learning systems tend to concentrate on a single algorithm. Propositional systems are therefore often more versatile and give users the possibility to work with the algorithms they are used to.

Solutions

There are various ways to propositionalize relational data consisting of at least two tables linked together through a relationship. We first focus on a single relationship between two tables. Most approaches can then iteratively deal with several relationships as explained below.

Propositionalization mechanisms depend on whether that relationship is functional or non-determinate. This distinction explains most common mistakes made by newcomers.

Functional Relationship (Many-To-One, One-To-One)

When the primary table has a many-to-one or one-to-one relationship to the complementary table, each row of the primary table links to one row of the complementary table. A simple join of the two tables results in a single table where each row of the primary table

is completed with the information derived from the complementary table.

Example 2 *In our simplified medical domain, there is a many-to-one relationship from each patient to his or her company. Let us focus on those two tables only. A join of the two tables results in a single table where each row describes a single patient and the company he or she works for:*

The resulting table is suitable for any attribute-value learner.

Non-Determinate Relationship (One-To-Many, Many-To-Many)

Propositionalization is less trivial in a non-determinate context, when there is a one-to-many or many-to-many relationship from the primary table to the complementary table, i.e., when one individual of the primary table is associated to a set of rows of the complementary table.

A propositional attribute is built by applying an aggregation function to a column of the complementary table over a selection of rows. Of course a lot of conditions can be used to select the rows. Those conditions can involve other columns than the aggregated column. Any aggregation function can be used, e.g., to check whether the set is not empty, to count how many elements there are, to find the mean (for numerical) or the mode (for categorical) values, etc.

Example 3 *In our simplified medical domain, there is a one-to-many relationship from the patient to his or her examinations. Let us focus on those two tables only. Many features can be constructed. Simple features are aggregation functions applied to a scalar (numerical or categorical) column. The number of occurrences of the different values of every categorical attributes can be counted. For instance, the f60 feature in the table below counts in how many examinations the patient stated he or she smoked. The maximum, minimum, average, and standard deviation of every numerical columns can be estimated, e.g., the f84 and f85 features in the table below respectively estimates the average and the maximum blood pressure of the patient over his or her examinations. The aggregation functions can be applied to any selection of rows, e.g., the f135 feature in the table below estimates the average blood pressure over the examinations when the patient smoked.*

Patient and his/her examinations

pid	name	...	f60	...	f84	f85	...	f135	...
I	Smith	...	2	...	11	12	...	11	...
II	Blake	...	1	...	10.5	11	...	10	...
⋮	⋮	...	⋮	...	⋮	⋮	...	⋮	...

From this example, it is clear that non-determinate relationships can easily lead to a combinatorial explosion of the number of features.

Common Mistakes and Key Rules to Avoid them

Two mistakes are frequent when machine learning practitioners face a propositionalization problem, i.e., when they want to apply a propositional learner to an existing relational dataset (Lachiche, 2005).

The first mistake is to misuse the (universal) join. Join is valid in a functional context, as explained earlier. When applied to a non-determinate relationship, it produces a table where several rows correspond to a single individual, leading to a multiple instance problem (Dietterich, Lathrop, & Lozano-Pérez, 1997) (cf. ►Multi-Instance Learning).

Example 4 *In our simplified medical domain, there is a one-to-many relationship from the patient table to the examination table. If a join is performed, each row of the examination table is completed with the information on the examined patient, i.e., there are as many rows as examinations.*

Examination and its patient

eid	date	weight	smokes	BP	...	pid	name	...
1	10/10/1991	60	yes	10	...	I	Smith	...
2	04/06/1992	64	yes	12	...	I	Smith	...
⋮	⋮	⋮	⋮	⋮	...	⋮	⋮	...
23	20/12/1992	80	yes	10	...	II	Blake	...
24	15/11/1993	78	no	11	...	II	Blake	...
⋮	⋮	⋮	⋮	⋮	...	⋮	⋮	...

Patient and his/her company

pid	name	birth	height	job	cid	company	location	...
I	Smith	15/06/1956	1.67	manager	a	Eiffel	Paris	...
II	Blake	13/02/1968	1.82	salesman	a	Eiffel	Paris	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

In this example, the joined table deals with the examinations rather than with the patients. An attribute-value learner could be used to learn hypotheses about the examinations, not about the patients.

This example reinforces a key representation rule in attribute-value learning: "Each row corresponds to a single individual, and vice-versa."

The second mistake is a meaningless column concatenation. This is more likely when a one-to-many relationship can be misinterpreted as several one-to-one relationships, i.e., when the practitioner is led to think that a non-determinate relationship is actually functional.

Example 5 *In our simplified medical domain, let us assume that the physician numbered the successive examinations (1, 2, 3, and so on) of each patient. Then given that each patient has a first examination, it is tempting to consider that there is a functional relationship from the patient to his or her "first" examination, "second" examination, and so on . This would result in a new patient table with concatenated columns: weight at the first examination, whether he or she smoked at the first examination, . . . , weight at the second examination, etc.*

Patient and his or her examinations (incorrect representation!)

			"first" examination			"second" examination			
pid	name	...	weight	smokes	...	weight	smokes	...	...
I	Smith	...	60	yes	...	64	yes	...	...
II	Blake	...	80	yes	...	78	no	...	...
⋮	⋮	...	⋮	...	⋮	⋮	⋮	...	...

This could easily lead to an attribute-value learner generalizing over a patient's weight at their i-th examination, which is very unlikely to be meaningful.

Two aspects should warn the user of such a representation problem: first, the number of columns depends on the dataset, and as a consequence lots of columns are not defined for all individuals. Moreover, when the absolute numbering does not make sense, there is no functional relationship. Such a misunderstanding can be avoided by remembering that in an attribute-value representation, "each column is uniquely defined for each row."

Further Relationships

The first complementary table can itself have a non-determinate relationship with another complementary table, and so on. Two approaches are available.

A first approach is to consider the first complementary table, the one having a one-to-many relationship, as a new primary table in a recursive propositionalization.

Example 6 *In our simplified medical domain, the examination table has a one-to-many relationship with the additional test table. The propositionalization of the examination and additional test tables will lead to a new examination table completed with new features, such as a count of how many tests were bad:*

Examination and its additional_tests

eid	pid	date	weight	smokes	BP	...	bad tests	...
1	I	10/10/1991	60	yes	10	...	1	...
2	I	04/06/1992	64	yes	12	...	0	...
⋮	⋮	⋮	⋮	⋮	⋮	...	⋮	...

Then the propositionalization of the patient table and the already propositionalized examination tables is performed, producing a new patient table completed with new features such as the mean value for each patient of

the number of bad tests among all his or her examinations (f248):

Patient, his or her examinations and additional_tests

pid	name	...	f60	...	f248	...
1	Smith	...	2	...	1	...
⋮	⋮	...	⋮	...	⋮	...

It is not necessarily meaningful to aggregate at an intermediate level. An alternative is to join complementary tables first, and apply the aggregation at the individual level only.

Example 7 *In our simplified medical domain, it is perhaps more interesting to first relate all additional tests to their patients, then aggregate on similar tests. First the complementary tables are joined:*

Additional_test and its examination

tid	name	value	interpretation	eid	pid	weight	...
t237	red blood cells	35	bad	1	1	60	...
t238	radiography	nothing	good	1	1	60	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	...
t574	red blood cells	43	good	2	1	64	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

Let us emphasize the difference with the propositionalized examination and its additional_tests table of example 6.

There is a one-to-many relationship from the patient table to that new additional_test and its examination table. Aggregation functions can be used to build features such as the minimum percentage of red blood cells (f352):

Finally, different propositionalization approaches can be combined, by a simple join.

Future Directions

Propositionalization explicitly aims at leaving attribute selection to the propositional learner applied afterward. The number of potential features is large. No existing propositionalization system is able to enumerate all imaginable features. Historically existing approaches have focused on a subset of potential features, e.g., numerical aggregation functions without selection (Knobbe, de Haas, & Siebes, 2001), selection based on a single elementary condition and existential aggregation (Flach & Lachiche, 1999; Kramer, Lavrač, & Flach, 2001). Most approaches can be combined to provide more features. The propositionalization should be guided by the user.

Propositionalization is closely related to knowledge representation. Specific representational issues require appropriate propositionalization techniques, e.g., Perlich and Provost (2006) introduce new propositionalization operators to deal with high-cardinality categorical attributes. New data sources, such as geographical or multimedia data, will need an appropriate representation and perhaps appropriate propositionalization operators to apply off-the-shelf attribute-value learners.

Propositionalization raises three fundamental questions. The first question is related to knowledge representation. The question is whether the user should adapt to existing representations and accept a need to propositionalize, or whether data can be mined from the data sources, requiring the algorithms to be adapted or invented. The second question is whether propositionalization is needed. Propositionalization explicitly allows the user to contribute to the feature elaboration and invites him or her to guide the search thanks to that ►language bias. It separates feature elaboration from model extraction. Conversely, relational data mining techniques automate the elaboration of the relevant attributes during the model extraction, but at the same time leave less opportunity to select the features by hand.

The third issue is one of efficiency. A more expressive representation necessitates a more complex search. Relational learning algorithms face the same dilemma as attribute-value learning in the form of a choice between an intractable search in the complete search space and an ad hoc heuristic/search bias (cf. ►Search Bias). They only differ in the size of the search space (cf. ►Hypothesis Space). Propositionalization is concerned with generating the search space. Generating all potential features is usually impossible. So practitioners have to constrain the propositionalization, e.g., by choosing the aggregation functions, the complexity of the selections, etc., by restricting the numbers of operations, and so on. Different operators fit different problems and might lead to differences in performance (Krogel, Rawles, Železný, Flach, Lavrač, & Wrobel, 2003).

Patient, his or her additional_tests and examinations

pid	name	...	f60	...	f352	...
1	Smith	...	2	...	35	...
⋮	⋮	...	⋮	...	⋮	...

Cross References

►Attribute
►Feature Construction
►Feature Selection
►Inductive Logic Programming
►Language Bias
►Learning from Structured Data
►Multi-Instance learning
►Relational Learning
►Statistical Relational Learning

Recommended Reading

Dietterich, T. G., Lathrop, R. H., & Lozano-Pérez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence, 89*(1–2), 31–71.

Džeroski, S., & Lavrač, N. (Ed.). (2001). *Relational data mining*. Berlin: Springer.

Flach, P., & Lachiche, N. (1999). 1BC: A first-order bayesian classifier. In S. Džeroski & P. Flach (Eds.), *Proceedings of the ninth international workshop on inductive logic programming (ILP'99), Vol. 1634 of lecture notes in computer science* (pp. 92–103). Berlin: Springer.

Knobbe, A. J., de Haas, M., & Siebes, A. (2001). Propositionalization and aggregates. In *Proceedings of the sixth European conference on principles of data mining and knowledge discovery, Vol. 2168 of lecture notes in artificial intelligence* (pp. 277–288). Berlin: Springer.

Kramer, S., Lavrač, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. In S. Džeroski & N. Lavrač (Eds.), *Relational data mining* (Chap. 11, pp. 262–291). Berlin: Springer.

Krogel, M.-A., Rawles, S., Železný, F., Flach, P. A., Lavrač, N., & Wrobel, S. (2003). Comparative evaluation of approaches to propositionalization. In T. Horváth & A. Yamamoto (Eds.), *Proceedings of the thirteenth international conference on inductive logic programming, Vol. 2835 of lecture notes in artificial intelligence* (pp. 197–214). Berlin: Springer.

Lachiche, N. (2005). Good and bad practices in propositionalization. In S. Bandini & S. Manzoni (Eds.), *Proceedings of advances in artificial intelligence, ninth congress of the Italian association for artificial intelligence (AI*IA'05), Vol. 3673 of lecture notes in computer science* (pp. 50–61). Berlin: Springer.

Perlich, C., & Provost, F. (2006). Distribution-based aggregation for relational learning with identifier attributes. *Machine Learning, 62*, 62–105.

Srinivasan, A., Muggleton, S., King, R. D., & Stenberg, M. (1996). Theories for mutagenicity: A study of first-order and feature based induction. *Artificial Intelligence, 85*(1–2), 277–299.

Tomečková, M., Rauch, J., & Berka, P. (2002). Stulong data from longitudinal study of atherosclerosis risk factors. In P. Berka (Ed.), *Discovery challenge workshop notes. ECML/PKDD'02*, Helsinki, Finland. http://lisp.vse.cz/challenge/ecmlpkdd2002/proceedings/Tomeckova.pdf

Pruning

Johannes Fürnkranz
TU Darmstadt, Fachbereich Informatik, Darmstadt, Germany

Definition

Pruning describes the idea of avoiding ►Overfitting by simplifying a learned concept, typically after the actual induction phase. The word originates from ►Decision Tree learning, where the idea of improving the decision tree by cutting some of its branches is related to the concept of pruning in gardening.

One can distinguish between ►Pre-Pruning, where pruning decisions are taken during the learning process, and ►Post-Pruning, where pruning occurs in a separate phase after the learning process. Pruning techniques are particularly important for state-of-the-art decision tree and ►Rule Learning algorithms.

The key idea of pruning is essentially the same as ►Regularization in statistical learning, with the key difference that regularization incorporates a complexity penalty directly into the learning heuristic, whereas pruning uses a separate pruning criterion or pruning algorithm.

Cross References

►Decision Tree
►Pre-Pruning
►Post-Pruning
►Regularization
►Rule Learning

Pruning Set

Definition

A pruning set is a subset of a ►training set containing data that are used by a ►learning system to evaluate models that are learned from a ►growing set.

Cross References

►Data Set

Q

Q-Learning

Peter Stone
The University of Texas at Austin, Austin, TX, USA

Definition

Q-learning is a form of ►temporal difference learning. As such, it is a model-free ►reinforcement learning method combining elements of ►dynamic programming with Monte Carlo estimation. Due in part to Watkins' (1989) proof that it converges to the optimal value function, Q-learning is among the most commonly used and well-known reinforcement learning algorithms.

Cross References

►Reinforcement Learning
►Temporal Difference Learning

Recommended Reading

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis. Cambridge: King's College.

Quadratic Loss

►Mean Squared Error

Qualitative Attribute

►Categorical Attribute

Quality Threshold Clustering

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

Synonyms

QT Clustering

Quality threshold (QT) clustering (Heyer, Kruglyak, & Yooseph 1999) is a partitioning clustering algorithm originally proposed for gene clustering. The focus of the algorithm is to find clusters with guaranteed quality. Instead of specifying K, the number of clusters, QT uses the maximum cluster diameter as the parameter.

The basic idea of QT is as follows: Form a candidate cluster by starting with a random point and iteratively add other points, with each iteration adding the point that minimizes the increase in cluster diameter. The process continues until no point can be added without surpassing the diameter threshold. If surpassing the threshold, a second candidate cluster is formed by starting with a point and repeating the procedure. In order to achieve reasonable clustering quality, already assigned points are available for forming another candidate cluster.

For data partition, QT selects the largest candidate cluster, removes the points which belong to the cluster from consideration, and repeats the procedure on the remaining set of data.

Recommended Reading

Heyer, L., Kruglyak, S., & Yooseph, S. (1999). Exploring expression data: Identification and analysis of coexpressed genes. *Genome Research*, 9, 1106–1115.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Quantitative Attribute

►Numeric Attribute

Query-Based Learning

Sanjay Jain, Frank Stephan
National University of Singapore, Republic of Singapore

Definition

Most learning scenarios consider learning as a relatively passive process where the learner observes a set of data and eventually formulates a hypothesis that explains the data observed. Query-based learning is an ►active learning process where the learner has a dialogue with a teacher, which provides on request useful information about the concept to be learnt.

Detail

This article will mainly focus on query-based learning of finite classes and of parameterized families of finite classes. In some cases, an infinite class has to be learnt where then the behaviour of the learner is measured in terms of a parameter belonging to the concept. For example, when learning the class of all singletons $\{x\}$ with $x \in \{0,1\}^*$, the parameter would be the length n of x and an algorithm based on membership queries would need up to $2^n - 1$ queries of the form "Is y in L?" to learn an unknown set $L = \{x\}$ with $x \in \{0,1\}^n$. Query-based learning studies questions like the following: Which classes can be learnt using queries of this or that type? If queries of a given type are used to learn a parameterized class $\bigcup C_n$, is it possible to make a learner which (with or without knowledge of n) succeeds to learn every $L \in C_n$ with a number of queries that is polynomial in n? What is the exact bound on queries needed to learn a finite class C in dependence of the topology of C and the cardinality of C? If a query-based learner using polynomially many queries exists for a parameterized class $\bigcup C_n$, can this learner also be implemented such that it is computable in polynomial time?

In the following, let C be the class of concepts to be learnt and the concepts $L \in C$ are subsets of some basic set X. Now the learning process is a dialogue between a learner and a teacher in order to identify a language $L \in C$, which is known to the teacher but not to the learner. The dialogue goes in turns and follows a specific protocol that goes over a finite number of rounds. Each round consists of a query placed by the learner to the teacher and the answer of the teacher to this query. The query and the answer have to follow a specific format and there are the following common types, where $a \in X$ and $H \in C$ are data items and concepts chosen by the learner and $b \in X$ is a counterexample chosen by the teacher:

Query-Name	Precise Query	Answer if true	Answer if false
Membership-Query	Is $a \in L$?	"Yes"	"No"
Equivalence-Query	Is $H = L$?	"Yes"	"No" plus b (where $b \in H-L \cup L-H$)
Subset-Query	Is $H \subseteq L$?	"Yes"	"No" plus b (where $b \in H - L$)
Superset-Query	Is $H \supseteq L$?	"Yes"	"No" plus b (where $b \in L - H$)
Disjointness-Query	Is $H \cap L = \emptyset$?	"Yes"	"No" plus b (where $b \in H \cap L$)

While, for subset queries and superset queries, it is not required by all authors that the teacher provides a counterexample in the case that the answer is "no," this requirement is quite standard for the case of equivalence queries. Without counterexamples, a learner would not have any real benefit from these queries in settings where faster convergence is required, than by just checking "Is $H_0 = L$?," "Is $H_1 = L$?," "Is $H_2 = L$?," ..., which would be some trivial kind of algorithm.

Here is an example: Given the class C of all finite subsets of $\{0,1\}^*$, a learner using superset queries could

just work as follows to learn each set of the form $L = \{x_1, x_2, \ldots, x_n\}$ with $n+1$ queries:

Round	Query	Answer	Counter example
1	Is $L \subseteq \varnothing$?	"No"	x_1
2	Is $L \subseteq \{x_1\}$?	"No"	x_2
3	Is $L \subseteq \{x_1, x_2\}$?	"No"	x_3
⋮	⋮	⋮	⋮
n	Is $L \subseteq \{x_1, x_2, \ldots, x_{n-1}\}$?	"No"	x_n
$n+1$	Is $L \subseteq \{x_1, x_2, \ldots, x_{n-1}, x_n\}$?	"Yes"	—

Here, of course, the order on how the counterexamples come up does not matter; the given order was just preserved for the reader's convenience. Note that the same algorithm works also with equivalence queries in place of superset queries. In both cases, the algorithm stops with outputting "$L = \{x_1, x_2, \ldots, x_n\}$" after the last query. However, the given class is not learnable using membership and subset queries which can be seen as follows: Assume that such a learner learns $\varnothing$ using the subset queries "Is $H_0 \subseteq L$?," "Is $H_1 \subseteq L$?," "Is $H_2 \subseteq L$?," ..., "Is $H_m \subseteq L$?" and the membership queries "Is $y_0 \in L$?," "Is $y_1 \in L$?," "Is $y_2 \in L$?," ..., "Is $y_k \in L$?" Furthermore, let D be the set of all counterexamples provided by the learner to subset queries. Now let $E = D \cup H_0 \cup H_1 \cup \ldots \cup H_m \cup \{y_0, y_1, \ldots, y_k\}$. Note that E is a finite set and let x be an element of $\{0,1\}^* - E$. If $L = \{x\}$ then the answers to these queries are the same to the case that $L = \varnothing$. Hence, the learner cannot distinguish between the sets $\varnothing$ and $\{x\}$; therefore, the learner is incorrect on at least one of these sets.

In the case that C is finite, one could just ask what is the number of queries needed to determine the target L in the worst case. This depends on the types of queries permitted and also on the topology of the class C. For example, if C is the power set of $\{x_1, x_2, \ldots, x_n\}$, then n membership queries are enough; but if C is the set of all singleton sets $\{x\}$ with $x \in \{0,1\}^n$, then $2^n - 1$ membership queries are needed to learn the concept, although in both cases the cardinality of C is 2^n.

Angluin (2004) provides a survey of the prior results on questions like how many queries are needed to learn a given finite class. Maass and Turán (1992) showed that usage of membership queries in addition to equivalence queries does not speed up learning too much compared to the case of using equivalence queries alone. If EQ is the number of queries needed to learn C from equivalence queries alone (with counterexamples) and EMQ is the number of queries needed to learn C with equivalence queries and membership queries then

$$\frac{EQ}{\log(EQ+1)} \leq EMQ \leq EQ;$$

here the logarithm is base 2. This result is based on a result of Littlestone (1988) who characterized the number of queries needed to learn from equivalence queries alone and provided a "standard optimal algorithm" for this task.

Angluin (1987) showed that the class of all regular languages can be learnt in polynomial time using queries and counterexamples. Here the learning time is measured in terms of two parameters: the number n of states that the smallest determinisitic finite automaton generating the language has and the number m of symbols in the longest counterexample provided by the teacher. Ibarra and Jiang (1988) showed that the algorithm can be improved to need at most dn^3 equivalence queries when the teacher always returns the shortest counterexample; Birkendorf, Böker, and Simon (2000) improved the bound to dn^2. In these bounds, d is the size of the alphabet used for defining the regular languages to be learnt.

Much attention has been paid to the following question: Which classes of Boolean formulas over n variables can be learnt with polynomially many queries, uniformly in n (see, for example, Aizenstein et al. (1995); Aizenstein (1995); Angluin, Hellerstein, and Karpinski (1993); Hellerstein, Pillaipakkamnatt, Raghavan, and Wilkins (1996))? Angluin, Hellerstein, and Karpinski (1993) showed that read-once formulas, in which every variable occurs only once, are learnable in polynomial time using membership and equivalence queries. On the other hand, read-thrice DNF (disjunctive normal form) formulas cannot be learnt in polynomial time using the same queries (Aizenstein et al., 1992) unless

P = NP. In other words, such a learner would not succeed because of the limited computational power of a polymomial time learner; hence, equipping the learner with an additional oracle that can provide this power would permit to build such a learner. Here an oracle - in contrast to a teacher - does not know the task to be learnt but gives information which is difficult or impossible to compute. Such an oracle could, for example, be the set SAT of all satisfiable formulas and thus the learner could gain additional power by asking the oracle whether certain formulas are satisfiable. A special class of Boolean formulas is that of Horn clauses and the study in this field is still active (see, for example, Angluin, Frazier and Pitt (1992), Arias (2004), Arias & Balcazar (2009) and Arias & Khardon (2002)).

There are links to other fields. Angluin (1988, 1990) investigated the relation between query learning and ►PAC Learning. She found that every class which is learnable using membership queries and equivalence queries is also PAC learnable (Angluin, 1988); the PAC learner also works in polynomial time and needs at most polynomially many examples. More recent research on learning Boolean formulas also combines queries with probabilistic aspects (Jackson, 1997). Furthermore, query learning has also been applied to ►Inductive Inference (see, for example, Gasarch (2008, 1992); Jain et al (2007); Lange (2005)). Here the power of the learner depends not only on the type of queries permitted but also on whether queries of the corresponding type can be asked finitely often or infinitely often; the latter applies of course only to learning models where the learner converges in the limit and may revise the hypothesis from time to time. Furthermore, queries to oracles have been studied widely, see the entry on ►Complexity of Inductive Inference.

Recommended Reading

Aizenstein, H., Hellerstein, L., & Pitt, L. (1995). Read-thrice DNF is hard to learn with membership and equivalence queries. In *Thirty-third annual symposium on foundations of computer science*, Pittsburgh, 24–27 October 1992 (pp. 523–532). Washington, DC: IEEE Computer Society.

Aizenstein, H., & Pitt, L. (1995). On the learnability of disjunctive normal form formulas. *Machine Learning, 19*(3), 183–208.

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation, 75*(2), 87–106.

Angluin, D. (1988). Queries and concept learning. *Machine Learning, 2*(4), 319–342.

Angluin, D. (1990). Negative results for equivalence queries. *Machine Learning, 5*, 121–150.

Angluin, D. (2004). Queries revisited. *Theoretical Computer Science, 313*, 175–194.

Angluin, D., Frazier, M., & Pitt, L. (1992). Learning conjunctions of Horn clauses. *Machine Learning, 9*, 147–164.

Angluin, D., Hellerstein, L., & Karpinski, M. (1993). Learning read-once formulas with queries. *Journal of the Association for Computing Machinery, 40*, 185–210.

Arias, M. (2004). Exact learning of first-order Horn expressions from queries. Ph.D. Thesis, Tufts University.

Arias, M., & Balcázar, J. L. (2009). Canonical Horn representations and query learning. *Algorithmic learning theory: Twentieth international conference ALT 2009*, LNAI (Vol. 5809, pp. 156–170). Berlin: Springer.

Arias, M., & Khardon, R. (2002). Learning closed Horn expressions. *Information and Computation, 178*(1), 214–240.

Birkendorf, A., Böker, A., & Simon, H. U. (2000). Learning deterministic finite automata from smallest counterexamples. *SIAM Journal on Discrete Mathematics, 13*(4), 465–491.

Hellerstein, L., Pillaipakkamnatt, K., Raghavan, V. V., & Wilkins, D. (1996). How many queries are needed to learn? *Journal of the Association for Computing Machinery, 43*, 840–862.

Gasarch, W., & Lee, A. C. Y. (2008). Inferring answers to queries. *Journal of Computer and System Sciences, 74*(4), 490–512.

Gasarch, W., & Smith, C. H. (1992). Learning via queries. *Journal of the Association for Computing Machinery, 39*(3), 649–674.

Ibarra, O. H., & Jiang, T. (1988). Learning regular languages from counterexamples. In *Proceedings of the first annual workshop on computational learning theory*, MIT, Cambridge (pp. 371–385). San Francisco: Morgan Kaufmann.

Jackson, J. (1997). An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences, 55*(3), 414–440.

Jain, S., Lange, S., & Zilles, S. (2007). A general comparison of language learning from examples and from queries. *Theoretical Computer Science, 387*(1), 51–66.

Lange, S., & Zilles, S. (2005). Relations between Gold-style learning and query learning. *Information and Computation, 203*, 211–237.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. *Machine Learning, 2*, 285–318.

Maass, W., & Turán, G. (1992). Lower bound methods and separation results for on-line learning models. *Machine Learning, 9*, 107–145.

R

Rademacher Average

►Rademacher Complexity

Rademacher Complexity

Synonyms

Rademacher average

Definition

Rademacher complexity is a measure used in ►generalization bounds to quantify the "richness" of a class of functions. Letting $\rho_1, \ldots, \rho_n$ denote *Rademacher variables* – independent random variables that take the values ± 1 with equal probability – the *empirical* or *conditional Rademacher complexity* of a class of real-valued functions $\mathcal{F}$ on the points $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ is the conditional expectation

$$\hat{R}_{\mathbf{x}}(\mathcal{F}) = \mathbb{E}_{\mathbf{x},\rho}\left[\sup_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \rho_i f(x_i)\middle|\mathbf{x}\right].$$

Intuitively, the empirical Rademacher average $R_n(\mathcal{F})$ measures how well functions $f \in \mathcal{F}$ evaluated on $\mathbf{x} \in \mathcal{X}$ can align with randomly chosen labels. The (full) *Rademacher complexity* $R_n(\mathcal{F})$ with respect to a distribution P over $\mathcal{X}$ is the average empirical complexity when the arguments $x_1, \ldots, x_n$ are independent random variables drawn from P. That is,

$$R_n(\mathcal{F}) = \mathbb{E}_{\mathbf{x}}\left[\hat{R}_{\mathbf{x}}(\mathcal{F})\right].$$

There are several properties of the Rademacher average that make it a useful quantity in analysis: for any two classes $\mathcal{F} \subseteq \mathcal{G}$ we have $R_n(\mathcal{F}) \leq R_n(\mathcal{G})$; when $c \cdot \mathcal{F} := \{cf : f \in \mathcal{F}\}$ for $c \in \mathbb{R}$ we have $R_n(c \cdot \mathcal{F}) = |c| R_n(\mathcal{F})$; when $\mathcal{F} + g := \{f + g : f \in \mathcal{G}\}$ for some fixed function g we have $R_n(\mathcal{F} + g) = R_n(\mathcal{F})$; and if $\mathrm{conv}(\mathcal{F})$ is the convex hull of $\mathcal{F}$ then $R_n(\mathrm{conv}(\mathcal{F})) = R_n(\mathcal{F})$.

Radial Basis Function Approximation

►Radial Basis Function Networks

Radial Basis Function Networks

M.D. Buhmann
Justus-Liebig University,
Giessen, Germany

Synonyms

Networks with kernel functions; Radial basis function approximation; Radial basis function neural networks; Regularization networks

Definition

Radial basis function networks are a means of approximation by algorithms using linear combinations of translates of a rotationally invariant function, called the radial basis function. The coefficients of these approximations usually solve a minimization problem and can also be computed by interpolation processes. The radial basis functions constitute the so-called reproducing kernels on certain Hilbert-spaces or – in a slightly more general setting – semi-Hilbert spaces. In the latter case, the aforementioned approximation also contains an element from the nullspace of the semi-norm of the semi-Hilbert space. That is usually a polynomial space.

Motivation and Background

Radial basis function networks are a method to approximate functions and data by applying ►kernel methods to ►neural networks. More specifically, approximations

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

of functions or data via algorithms that make use of networks (or neural networks) can be interpreted as either interpolation or minimization problems using kernels of certain shapes, called radial basis functions in the form in which we wish to consider them in this chapter. In all cases, they are usually *high-dimensional approximations*, that is the number of unknowns n in the argument of the kernel may be very large. On the other hand, the number of learning examples ("data") may be quite small. The name neural networks comes from the idea that this learning process simulates the natural functioning of neurons.

At any rate, the purpose of this approach will be the modelization of the learning process by mathematical methods. In most practical cases of networks, the data from which we will learn in the method are rare, i.e., we have few data "points." We will consider this learning approach as an approximation problem in this description, essentially it is a minimizing (regression) problem.

Structure of the Network/Learning System

To begin with, let $\varphi : \mathbb{R}_+ \to \mathbb{R}$ be a univariate continuous function and $\|\cdot\|$ be the Euclidean norm on $\mathbb{R}^n$ for some $n \in \mathbb{N}$, as used for approximation in the seminal paper by Schoenberg (1938). Here, $\mathbb{R}_+$ denotes the set of nonnegative reals. Therefore,

$$\varphi(\|\cdot\|) : \mathbb{R}^n \to \mathbb{R}, \quad (x_1, x_2, \ldots, x_n)^{\mathrm{T}} \mapsto \varphi\left(\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}\right)$$

is a multivariate function and here the number n of unknowns may be very large in practice. This function is rotationally invariant. Incidentally, much of what is going to be said here will work if we replace this function by a general, n-variate function which need no longer be rotationally invariant, but then, strictly speaking, we are no longer talking about radial basis functions. Then other conditions may replace the restriction to radiality. Nonetheless, we stick to the simple case (which is entirely sufficient for many practical applications) when the function really is radially symmetric.

We also require for the time being that this n-variate function be positive definite, that is for all finite sets Ξ of pairwise different the so-called *centers* or data sites $\xi \in \Xi \subset \mathbb{R}^n$, the symmetric matrix

$$A = \{\varphi(\|\xi - \zeta\|)\}_{\xi,\zeta\in\Xi}$$

is a positive definite matrix. The condition of pairwise different data in Ξ may, of course, in practice, not be necessarily met.

This property is usually obtained by requiring that $\varphi(\|\cdot\|)$ be absolutely integrable and its Fourier transform – which thereby exists and is continuous – is positive everywhere ("Bochner's theorem"). An example for such a useful function is the exponential (the "Gauß-kernel") $\varphi(r) = \exp(-c^2 r^2)$, $r \geqslant 0$, where c is a positive parameter. For this the above positive definiteness is guaranteed for all positive c and all n. Another example is the Poisson-kernel $\varphi(r) = \exp(-cr)$. However, we may also take the nonintegrable "inverse multiquadrics" $\varphi(r) = 1/\sqrt{r^2 + c^2}$, which has a Fourier transform in the generalized or distributional sense that is also positive everywhere except at zero. There it has a singularity. Nonetheless, the aforementioned matrices of the form A are still always positive definite for these exponentials and the inverse multiquadrics so long as $c > 0$ and $n = 1, 2, \ldots$

This requirement of positive definiteness guarantees that for all given finite sets Ξ and "data" $f_\xi \in \mathbb{R}$, $\xi \in \Xi$, there is a unique linear combination

$$s(x) = \sum_{\xi\in\Xi} \lambda_\xi \varphi(\|x - \xi\|), \quad x \in \mathbb{R}^n,$$

which satisfies the linear interpolation conditions

$$s(\xi) = f_\xi, \quad \forall\ \xi \in \Xi.$$

This is because the interpolation matrix which is used to compute the coefficients λ_ξ is just the matrix A above which is positive definite, thus regular. The expression in the pen-ultimate display is the network that approximates the data given by the user. Of course the interpolation conditions are just what is meant by learning from examples, the data being the $|\Xi|$ examples. Here as always, $|\Xi|$ denotes the cardinality of the set Ξ. In the learning theory the linear space spanned by the above translates of $\varphi(\|\cdot\|)$ by $\xi \in \Xi$ is called the feature space with φ as activation function.

Incidentally, it is straightforward to generalize the approximation method to an approximation to data in $\mathbb{R}^m$, $m \in \mathbb{N}$, by approximating the data $f_\xi \in \mathbb{R}^m$ componentwise by m such expressions as the above, call them $s_1, s_2, \ldots, s_m$.

Applications

Applications include classification of data, pattern recognition, ▶time series analysis, picture smoothing similar to diffusion methods, and optimization.

Theory/Solution

Returning to interpolation, the problem may also be reinterpreted as a minimization problem. If the weighted L^2-integral is defined as

$$\|g\|_\varphi := \frac{1}{(2\pi)^{n/2}} \sqrt{\int_{\mathbb{R}^n} \frac{1}{\hat{\varphi}(\|x\|)} |\hat{g}(x)|^2 \, dx},$$

with $\hat{\varphi}$ still being the above positive Fourier transform, for all $g : \mathbb{R}^n \to \mathbb{R}$ for which the Fourier transform in the sense of $L^2(\mathbb{R}^n)$ is well-defined and for which the above integral is finite, we may ask for the approximant to the above data – which still must satisfy the aforementioned interpolation conditions – that minimizes $\|\cdot\|_\varphi$. As Duchon noted, for example, for the thin-plate spline case $\varphi(r) = r^2 \log r$ in this seminal papers this is just the above interpolant, i.e., that linear combination s of translates of radial basis functions, albeit in the thin-plate spline case with a linear polynomial added as we shall see below.

This works immediately both for the two examples of exponential functions and the inverse multiquadrics. Note the fact that the latter has a Fourier transform with a singularity at the origin, does not matter as its reciprocal appears as a weight function in the integral above. The important requirement is that the Fourier transform has no zero. It also works for the positive definite radial basis functions of compact support for instance in Buhmann (1998).

Regularization and Generalizations

Generally, since the interpolation problem to data may be ill-conditioned or unsuitable in the face of ▶noise, smoothing or ▶regularization are appropriate as an alternative. Indeed, the interpolation problem may be replaced by a smoothing problem which is of the form

$$\frac{1}{|\Xi|} \sum_{\xi \in \Xi} (s(\xi) - f_\xi)^2 + \mu \|s\|_\varphi^2 = \min_s !.$$

Here the L^2-integral is still the one used in the description above and μ is a positive smoothing parameter.

However, when there is only a trivial nullspace of the $\|\cdot\|_\varphi$, i.e., $g = 0$ is the only g with $\|g\|_\varphi = 0$, then it is a norm and the solution of this problem will have the form

$$s(x) = \sum_{\xi \in \Xi} \lambda_\xi \varphi(\|x - \xi\|), \quad x \in \mathbb{R}^n.$$

This is where the name regularization network comes from, regularization and smoothing being used synonymously. The form used here in the pen-ultimate display is a classical regularizing network problem or in the spline-terminology a smoothing spline problem. For ▶support vector machines, the square of the residual term $s(\xi) - f_\xi$ should be replaced by another expression, for example, the one by Vapnik (1996)

$$|s(\xi) - f_\xi|_\epsilon := \begin{cases} f_\xi - s(\xi) - \epsilon & \text{if } |f_\xi - s(\xi)| \geq \epsilon, \\ 0 & \text{otherwise,} \end{cases}$$

and for the support vector machines classification by the truncated power function $(\cdot)_+^\nu$ which is a positive power for positive argument and otherwise zero.

In the case of a classical regularizing network, the coefficients of the solution may be found by solving a similar linear system to the standard interpolation linear system mentioned above, namely

$$(A + \mu I)\lambda = f,$$

where f is the vector $(f_\xi)_{\xi \in \Xi}$ in $\mathbb{R}^\Xi$ of the data given, and $\lambda = (\lambda_\xi)_{\xi \in \Xi}$. The I denotes the $|\Xi| \times |\Xi|$ identity matrix and A is still the same matrix as above. Incidentally, scaling mechanisms may also be introduced into the radial basis function by replacing the simple translate $\varphi(\|x - \xi\|)$ by $\varphi(\|x - \xi\|/\delta)$ for a positive δ which may even depend on ξ.

The ideas of regularization and smoothing are of course not new; for instance, regularization goes back to Tichonov et al. (1977) ("Tichonov regularization") and spline smoothing to Wahba (1985), especially when the smoothing parameter is adjusted via cross-validation or generalized cross-validation (GCV).

Now to the case of semi-norms $\|\cdot\|_\varphi$ with non-trivial nullspaces: indeed, the same idea can be carried through for other radial basis functions as well. In particular we are thinking here of those that do not provide positive definite radial basis interpolation matrices but strictly conditionally positive definite ones. We have

strictly positive definite radial basis functions of order $k+1$, $k \geq -1$, if the above interpolation matrices A are still positive definite but only on the subspace of those nonzero vectors $\lambda = (\lambda_\xi)$ in $\mathbb{R}^\Xi$ which satisfy

$$\sum_{\xi \in \Xi} \lambda_\xi p(\xi) = 0, \quad \forall\, p \in \mathbb{P}_n^k,$$

where $\mathbb{P}_n^k$ denotes the linear space of polynomials in n variables with total degree at most k. In other words, the quadratic form, $\lambda^{\mathrm{T}} A \lambda$, need only be positive for such $\lambda \neq 0$. For simplicity of the presentation, we shall let $\mathbb{P}_n^{-1}$ denote $\{0\}$. In particular, if the radial basis function is conditionally positive definite of order 0, its interpolation matrices A are always positive definite, i.e., without condition. Also, we have the minimal requirement that the sets of centers Ξ are unisolvent for this polynomial space, i.e., the only polynomial $p \in \mathbb{P}_n^k$ that vanishes identically on Ξ is the zero-polynomial.

The connection of this with a layered neural network is that the approximation above is a weighted sum (weighted by the coefficients λ_ξ) over usually nonlinear activation functions φ. The entries in the sum are the radial basis function neurons and there are usually many of them. The number of nodes in the model is n. The hidden layer of "radial basis function units" consists of $|\Xi|$ nodes, i.e., the number of centers in our radial basis function approximation. The output layer has m responses if the radial basis function approximation above is generalized to m-variate data, then we get $s_1, s_2, \ldots, s_m$ instead of just s, as already described. This network here is of the type of a nonlinear, layered, and feedforward network. More than one hidden layer is unusual. The choice of the radial basis functions (its smoothness for instance) and the flexibility in the positioning of the centers in clusters, grids (Buhmann, 1990, for example) or otherwise provide much of the required freedom for good approximations.

The properties of conditional positive definiteness are fulfilled now for a much larger realm of radial basis functions, which have still nowhere vanishing, generalized Fourier transforms but with higher order singularities at the origin. (Remember that this creates no problem for the well-defineness of $\|\cdot\|_\varphi$.) For instance, the above properties are true for the thin-plate spline function $\varphi(r) = r^2 \log r$, the shifted logarithm $\varphi(r) = (r^2 + c^2) \log(r^2 + c^2)$, and for the multiquadric $\varphi(r) = -\sqrt{r^2 + c^2}$. Here we still have a parameter c which may now be arbitrary real. The order of the above is one for the multiquadric and two for the thin-plate spline. Another commonly used radial basis function which gives rise to conditional positive definiteness is the $\varphi(r) = r^3$.

Hence the norm becomes a semi-norm with nullspace $\mathbb{P}_n^k$ but it still has the same form as a square-integral with the reciprocal of the Fourier transform of the radial basis function as a weight.

Therefore, we have to include a polynomial from the nullspace of the semi-norm to the approximant which becomes

$$s(x) = \sum_{\xi \in \Xi} \lambda_\xi \varphi(\|x - \xi\|) + q(x), \quad x \in \mathbb{R}^n,$$

where $q \in \mathbb{P}_n^k$ and the side conditions on the coefficients

$$\sum_{\xi \in \Xi} \lambda_\xi p(\xi) = 0, \quad \forall\, p \in \mathbb{P}_n^k.$$

If we consider the regularization network problem with the smoothing parameter μ again, then we have to solve the linear system with a smoothing parameter μ

$$(A + \mu I)\lambda + P^{\mathrm{T}} b = f, \quad P\lambda = 0,$$

where $P = (p_i(\xi))_{i=1,\ldots,L,\xi \in \Xi}$, and p_i form a basis of $\mathbb{P}_n^k$, b_i being the components of b, and $q(x) = \sum_{i=1}^L b_i p_i(x)$ is the expression of the polynomial added to the radial basis function sum. So in particular P is a matrix with as many rows as the dimension $L = \binom{n+k}{n}$ of $\mathbb{P}_n^k$ is and $|\Xi|$ columns.

In all cases, the radial basis functions composed with the Euclidean norm can be regarded as reproducing kernels in the semi-Hilbert spaces defined by the set X of distributions g for which $\|g\|_\varphi$ is finite and the semi-inner product

$$(h, g) = \frac{1}{(2\pi)^n} \int_{\mathbb{R}^n} \frac{1}{\hat{\varphi}(\|x\|)} \hat{h}(x) \overline{\hat{g}(x)}\, \mathrm{d}x, \quad h, g, \in X.$$

In particular, $\|g\|_\varphi^2 = (g, g)$. If the evaluation functional is continuous (bounded) on that space X, there exists a reproducing kernel, i.e., there is a $K : X \times X \to \mathbb{R}$ such that

$$g(x) = (g, K(\cdot, x)), \quad \forall\, x \in \mathbb{R}^n,\ g \in X,$$

see, for example, Wahba (1990). If the semi-inner product is actually an inner product, then the reproducing kernel is unique. The kernel gives rise to positive definite matrices $\{K(\xi,\zeta)\}_{\xi,\zeta\in\Xi}$ if and only if it is a positive operator. For the spaces X defined by our radial basis functions, it turns out that $K(x,y) := \varphi(\|x-y\|)$, see, e.g., the overview in Buhmann (2003). Then the matrices A are positive definite if $\hat{\varphi}(\|\cdot\|)$ is well-defined and positive, but if it has a singularity at zero, the A may be only conditionally positive definite. Note here that $\hat{\varphi}(\|\cdot\|)$ denotes the n-variate Fourier transform of $\varphi(\|\cdot\|)$, both being radially symmetric.

Advantages of the Approach

Why are we interested in using radial basis functions for networks? The radial basis functions have many excellent approximation properties which make them useful as general tools for approximation. Among them are the variety of more or less smoothness as required (e.g., multiquadrics is C^∞ for positive c and just continuous for $c = 0$), the fast evaluation and computation methods available (see, e.g., Beatson & Powell, 1994), the aforementioned nonsingularity properties and their connection with the theory of reproducing kernel Hilbert spaces, and finally their excellent convergence properties (see, e.g., Buhmann, 2003). Generally, neural networks are a tried and tested approach to approximation, modeling, and smoothing by methods from learning theory.

Limitations

The number of applications where the radial basis function approach has been used is vast. Also, the solutions may be computed efficiently by far field expansions, approximated Lagrange functions, and multipole methods. However, there are still some limitations with these important computational methods when the dimension n is large. So far, most of the multipole and far field methods have been implemented only for medium-sized dimensions.

Cross References

►Neural Networks
►Regularization
►Support Vector Machines

Recommended Reading

Beatson, R. K., & Powell, M. J. D. (1994). An iterative method for thin plate spline interpolation that employs approximations to Lagrange functions. In D. F. Griffiths & G. A. Watson (Eds.), *Numerical analysis 1993* (pp. 17–39). Burnt Mill: Longman.

Broomhead, D., & Lowe, D. (1988). Radial basis functions, multivariable functional interpolation and adaptive networks, *Complex Systems, 2*, 321–355.

Buhmann, M. D. (1990). Multivariate cardinal-interpolation with radial-basis functions. *Constructive Approximation, 6*, 225–255.

Buhmann, M. D. (1998). Radial functions on compact support. *Proceedings of the Edinburgh Mathematical Society, 41*, 33–46.

Buhmann, M. D. (2003). *Radial basis functions: Theory and implementations*. Cambridge: Cambridge University Press.

Duchon, J. (1976). Interpolation des fonctions de deux variables suivant le principe de la flexion des plaques minces. *RAIRO, 10*, 5–12.

Evgeniou, T., Poggio, T., & Pontil, M. (2000). Regularization networks and support vector machines. *Advances in Computational Mathematics, 13*, 1–50.

Hardy, R. L. (1990). Theory and applications of the multiquadric-biharmonic method. *Computers and Mathematics with Applications, 19*, 163–208.

Micchelli, C. A. (1986). Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation, 1*, 11–22.

Pinkus, A. (1996). TDI-subpaces of $C(\mathbb{R}^d)$ and some density problems from neural networks. *Journal of Approximation Theory, 85*, 269–287.

Schoenberg, I. J. (1938). Metric spaces and completely monotone functions. *Annals of Mathematics, 39*, 811–841.

Tichonov, A. N., & Arsenin, V. Y. (1977). *Solution of ill-posed problems*. Washington, DC: V.H. Winston.

Vapnik, V. N. (1996). *Statistical learning theory*. New York: Wiley.

Wahba, G. (1985). A comparison of GCV and GML for choosing the smoothing parameter in the generalized splines smoothing problem. *Annals of Statistics, 13*, 1378–1402.

Wahba, G. (1990). *Spline models for observational data. Series in applied mathematics* (Vol. 59). Philadelphia: SIAM.

Radial Basis Function Neural Networks

►Radial Basis Function Networks

Random Decision Forests

►Random Forests

Random Forests

Synonyms
Random decision forests

Definition
Random Forests is an ►ensemble learning technique. It is a hybrid of the ►Bagging algorithm and the ►random subspace method, and uses ►decision trees as the base classifier. Each tree is constructed from a bootstrap sample from the original dataset. An important point is that the trees are not subjected to pruning after construction, enabling them to be partially overfitted to their own sample of the data. To further diversify the classifiers, at each branch in the tree, the decision of which feature to split on is restricted to a *random subset* of size n, from the full feature set. The random subset is chosen anew for each branching point. n is suggested to be $\log_2(N+1)$, where N is the size of the whole feature set.

Random Subspace Method

Synonyms
Random subspaces; RSM

Definition
The random subspace method is an ►ensemble learning technique. The principle is to increase diversity between members of the ensemble by restricting classifiers to work on different random subsets of the full ►feature space. Each classifier learns with a subset of size n, chosen uniformly at random from the full set of size N. Empirical studies have suggested good results can be obtained with the rule-of-thumb to choose $n = N/2$ features. The method is generally found to perform best when there are a large number of features (large N), and the discriminative information is spread across them. The method can underperform in the converse situation, when there are few informative features, and a large number of noisy/irrelevant features. ►Random Forests is an algorithm combining RSM with the ►Bagging algorithm, which can provide significant gains over each used separately.

Random Subspaces

►Random Subspace Method

Randomized Decision Rule

►Markovian Decision Rule

Rank Correlation

Definition
Rank correlation measures the correspondence between two rankings τ and τ' of a set of m objects. Various proposals for such measures have been made, especially in the field of statistics. Two of the best-known measures are Spearman's Rank Correlation and Kendall's tau:
Spearman's Rank correlation calculates the sum of squared rank distances and is normalized such that it evaluates to −1 for reversed and to +1 for identical rankings. Formally, it is defined as follows:

$$(\tau, \tau') \mapsto 1 - \frac{6 \sum_{i=1}^{m} (\tau(i) - \tau'(i))^2}{m(m^2 - 1)} \tag{1}$$

Kendall's tau is the number of pairwise rank inversions between τ and τ', again normalized to the range $[-1, +1]$:

$$(\tau, \tau') \mapsto 1 - \frac{4 \left| \{(i,j) \mid i < j,\ \tau(i) < \tau(j) \wedge \tau'(i) > \tau'(j)\} \right|}{m(m-1)} \tag{2}$$

Cross References
►Preference Learning
►ROC Analysis

Ratio Scale

A **ratio** measurement scale possesses all the characteristics of interval measurement, and there exists a *zero* that, the same as arithmetic *zero*, means "nil" or "nothing." See ►Measurement Scales.

Real-Time Dynamic Programming

Real-Time Dynamic Programming (RTDP) is the same as ►Adaptive Real-Time Dynamic Programming (ARTDP) without the system identification component. It is applicable when an accurate model of the problem is available. It converges to an optimal policy of a stochastic optimal path problem under suitable conditions. RTDP was introduced by Barto, Bradtke, and Singh (1995) in their paper Learning to Act Using RTDP.

Recall

Recall is a measure of information retrieval performance. Recall is the total number of documents retrieved that are elevant/Total number of relevant documents in the database. See ►Precision and Recall.

Cross References

►Sensitivity

Receiver Operating Characteristic Analysis

►ROC Analysis

Recognition

►Classification

Recommender Systems

Prem Melville, Vikas Sindhwani
IBM T. J. Watson Research Center
Yorktown Heights, NY, USA

Definition

The goal of a recommender system is to generate meaningful recommendations to a collection of users for items or products that might interest them. Suggestions for books on Amazon, or movies on Netflix, are real-world examples of the operation of industry-strength recommender systems. The design of such recommendation engines depends on the domain and the particular characteristics of the data available. For example, movie watchers on Netflix frequently provide ratings on a scale of 1 (disliked) to 5 (liked). Such a data source records the quality of interactions between users and items. Additionally, the system may have access to user-specific and item-specific profile attributes such as demographics and product descriptions, respectively. Recommender systems differ in the way they analyze these data sources to develop notions of affinity between users and items, which can be used to identify well-matched pairs. ►Collaborative Filtering systems analyze historical interactions alone, while ►Content-based Filtering systems are based on profile attributes; and hybrid techniques attempt to combine both of these designs. The architecture of recommender systems and their evaluation on real-world problems is an active area of research.

Motivation and Background

Obtaining recommendations from trusted sources is a critical component of the natural process of human decision making. With burgeoning consumerism buoyed by the emergence of the web, buyers are being presented with an increasing range of choices while sellers are being faced with the challenge of personalizing their advertising efforts. In parallel, it has become common for enterprises to collect large volumes of transactional data that allows for deeper analysis of how a customer base interacts with the space of product offerings. Recommender systems have evolved to fulfill the natural dual need of buyers and sellers by automating the generation of recommendations based on data analysis.

The term "collaborative filtering" was introduced in the context of the first commercial recommender system, called Tapestry (Goldberg, Nichols, Oki, & Terry, 1992), which was designed to recommend documents drawn from newsgroups to a collection of users. The motivation was to leverage social collaboration in order to prevent users from getting inundated by a large volume of streaming documents. Collaborative filtering, which analyzes usage data across users

to find well-matched user-item pairs, has since been juxtaposed against the older methodology of content filtering, which had its original roots in information retrieval. In content filtering, recommendations are not "collaborative" in the sense that suggestions made to a user do not explicitly utilize information across the entire user-base. Some early successes of collaborative filtering on related domains included the GroupLens system (Resnick, Neophytos, Bergstrom, Mitesh, & Riedl, 1994b).

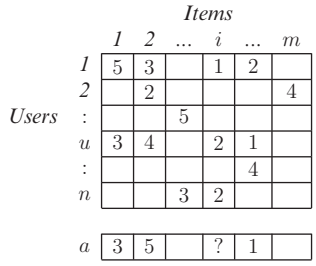
As noted in Billsus and Pazzani (1998), initial formulations for recommender systems were based on straightforward correlation statistics and predictive modeling, not engaging the wider range of practices in statistics and machine learning literature. The collaborative filtering problem was mapped to classification, which allowed dimensionality reduction techniques to be brought into play to improve the quality of the solutions. Concurrently, several efforts attempted to combine content-based methods with collaborative filtering, and to incorporate additional domain knowledge in the architecture of recommender systems.

Further research was spurred by the public availability of datasets on the web, and the interest generated due to direct relevance to e-commerce. Netflix, an online streaming video and DVD rental service, released a large-scale dataset containing 100 million ratings given by about half-a-million users to thousands of movie titles, and announced an open competition for the best collaborative filtering algorithm in this domain. Matrix Factorization (Bell, Koren, & Volinsky, 2009) techniques rooted in numerical linear algebra and statistical matrix analysis emerged as a state-of-the-art technique.

Currently, recommender systems remain an active area of research, with a dedicated ACM conference, intersecting several subdisciplines of statistics, machine learning, data mining, and information retrievals. Applications have been pursued in diverse domains ranging from recommending webpages to music, books, movies, and other consumer products.

Structure of Learning System

The most general setting in which recommender systems are studied is presented in Fig. 1. Known user preferences are represented as a matrix of n users and m items, where each cell $r_{u,i}$ corresponds to the rating given to item i by the user u. This *user ratings matrix* is typically sparse, as most users do not rate most items. The *recommendation task* is to predict what rating a user would give to a previously unrated item. Typically, ratings are predicted for all items that have not been observed by a user, and the highest rated items are presented as recommendations. The user under current consideration for recommendations is referred to as the *active user*.



Recommender Systems. Figure 1. User ratings matrix, where each cell $r_{u,i}$ corresponds to the rating of user u for item i. The task is to predict the missing rating $r_{a,i}$ for the active user a

The myriad approaches to recommender systems can be broadly categorized as:

- *Collaborative Filtering (CF)*: In CF systems, a user is recommended items based on the past ratings of all users collectively.
- *Content-based recommending*: These approaches recommend items that are similar in content to items the user has liked in the past, or matched to predefined attributes of the user.
- *Hybrid approaches*: These methods combine both collaborative and content-based approaches.

Collaborative Filtering

Collaborative filtering (CF) systems work by collecting user feedback in the form of ratings for items in a given domain and exploiting similarities in rating behavior amongst several users in determining how to recommend an item. CF methods can be further subdivided into *neighborhood-based* and *model-based* approaches. Neighborhood-based methods are

also commonly referred to as *memory-based* approaches (Breese, Heckerman, & Kadie, 1998).

Neighborhood-based Collaborative Filtering In neighborhood-based techniques, a subset of users are chosen based on their similarity to the active user, and a weighted combination of their ratings is used to produce predictions for this user. Most of these approaches can be generalized by the algorithm summarized in the following steps:

1. Assign a weight to all users with respect to similarity with the active user.
2. Select k users that have the highest similarity with the active user – commonly called the *neighborhood*.
3. Compute a prediction from a weighted combination of the selected neighbors' ratings.

In step 1, the weight $w_{a,u}$ is a measure of similarity between the user u and the active user a. The most commonly used measure of similarity is the Pearson correlation coefficient between the ratings of the two users (Resnick, Iacovou, Sushak, Bergstrom, & Reidl, 1994a), defined below:

$$w_{a,u} = \frac{\sum_{i \in I} (r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i \in I} (r_{a,i} - \bar{r}_a)^2 \sum_{i \in I} (r_{u,i} - \bar{r}_u)^2}} \quad (1)$$

where I is the set of items rated by both users, $r_{u,i}$ is the rating given to item i by user u, and $\bar{r}_u$ is the mean rating given by user u.

In step 3, predictions are generally computed as the weighted average of deviations from the neighbor's mean, as in:

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in K} (r_{u,i} - \bar{r}_u) \times w_{a,u}}{\sum_{u \in K} w_{a,u}} \quad (2)$$

where $p_{a,i}$ is the prediction for the active user a for item i, $w_{a,u}$ is the similarity between users a and u, and K is the neighborhood or set of most similar users.

Similarity based on Pearson correlation measures the extent to which there is a linear dependence between two variables. Alternatively, one can treat the ratings of two users as a vector in an m-dimensional space, and compute similarity based on the cosine of the angle between them, given by:

$$w_{a,u} = \cos(\boldsymbol{r}_a, \boldsymbol{r}_u) = \frac{\boldsymbol{r}_a \cdot \boldsymbol{r}_u}{\|\boldsymbol{r}_a\|_2 \times \|\boldsymbol{r}_u\|_2} = \frac{\sum_{i=1}^{m} r_{a,i} r_{u,i}}{\sqrt{\sum_{i=1}^{m} r_{a,i}^2} \sqrt{\sum_{i=1}^{m} r_{u,i}^2}} \quad (3)$$

When computing cosine similarity, one cannot have negative ratings, and unrated items are treated as having a rating of zero. Empirical studies (Breese et al., 1998) have found that Pearson correlation generally performs better. There have been several other similarity measures used in the literature, including *Spearman rank correlation*, *Kendall's τ correlation*, *mean squared differences*, *entropy*, and *adjusted cosine similarity* (Herlocker, Konstan, Borchers, & Riedl, 1999; Su & Khoshgoftaar, 2009).

Several extensions to neighborhood-based CF, which have led to improved performance are discussed below.

Item-based Collaborative Filtering: When applied to millions of users and items, conventional neighborhood-based CF algorithms do not scale well, because of the computational complexity of the search for similar users. As a alternative, Linden, Smith, and York (2003) proposed *item-to-item* collaborative filtering where rather than matching similar users, they match a user's rated items to similar items. In practice, this approach leads to faster online systems, and often results in improved recommendations (Linden et al., 2003; Sarwar, Karypis, Konstan, & Reidl, 2001).

In this approach, similarities between pairs of items i and j are computed off-line using Pearson correlation, given by:

$$w_{i,j} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_j)^2}} \quad (4)$$

where U is the set of all users who have rated both items i and j, $r_{u,i}$ is the rating of user u on item i, and $\bar{r}_i$ is the average rating of the ith item across users.

Now, the rating for item i for user a can be predicted using a simple weighted average, as in:

$$p_{a,i} = \frac{\sum_{j \in K} r_{a,j} w_{i,j}}{\sum_{j \in K} |w_{i,j}|} \quad (5)$$

where K is the neighborhood set of the k items rated by a that are most similar to i.

For item-based collaborative filtering too, one may use alternative similarity metrics such as *adjusted cosine similarity*. A good empirical comparison of variations of item-based methods can be found in Sarwar et al. (2001).

Significance Weighting: It is common for the active user to have highly correlated neighbors that are based on very few co-rated (overlapping) items. These neighbors based on a small number of overlapping items tend to be bad predictors. One approach to tackle this problem is to multiply the similarity weight by a *significance weighting* factor, which devalues the correlations based on few co-rated items (Herlocker et al., 1999).

Default Voting: An alternative approach to dealing with correlations based on very few co-rated items is to assume a default value for the rating for items that have not been explicitly rated. In this way one can now compute correlation (Eq. 1) using the union of items rated by users being matched as opposed to the intersection. Such a *default voting* strategy has been shown to improve collaborative filtering by Breese et al. (1998).

Inverse User Frequency: When measuring the similarity between users, items that have been rated by all (and universally liked or disliked) are not as useful as less common items. To account for this Breese et al. (1998) introduced the notion of *inverse user frequency*, which is computed as $f_i = \log n/n_i$, where n_i is the number of users who have rated item i out of the total number of n users. To apply inverse user frequency while using similarity-based CF, the original rating is transformed for i by multiplying it by the factor f_i. The underlying assumption of this approach is that items that are universally loved or hated are rated more frequently than others.

Case Amplification: In order to favor users with high similarity to the active user, Breese et al. (1998) introduced *case amplification* which transforms the original weights in Eq. (2) to

$$w'_{a,u} = w_{a,u} \cdot |w_{a,u}|^{\rho-1}$$

where ρ is the amplification factor, and $\rho \geq 1$.

Other notable extensions to similarity-based collaborative filtering include *weighted majority prediction* (Nakamura & Abe, 1998) and *imputation-boosted CF* (Su, Khoshgoftaar, Zhu, & Greiner, 2008).

Model-based Collaborative Filtering Model-based techniques provide recommendations by estimating parameters of statistical models for user ratings. For example, Billsus and Pazzani (1998) describe an early approach to map CF to a classification problem, and build a classifier for each active user representing items as features over users and available ratings as labels, possibly in conjunction with dimensionality reduction techniques to overcome data sparsity issues. Other predictive modeling techniques have also been applied in closely related ways.

More recently, ▶latent factor and matrix factorization models have emerged as a state-of-the-art methodology in this class of techniques (Bell et al., 2009). Unlike neighborhood based methods that generate recommendations based on statistical notions of similarity between users, or between items, latent factor models assume that the similarity between users and items is simultaneously induced by some hidden lower-dimensional structure in the data. For example, the rating that a user gives to a movie might be assumed to depend on few implicit factors such as the user's taste across various movie genres. Matrix factorization techniques are a class of widely successful latent factor models where users and items are simultaneously represented as unknown feature vectors (column vectors) $w_u, h_i \in \mathbb{R}^k$ along k latent dimensions. These feature vectors are learnt so that inner products $w_u^T h_i$ approximate the known preference ratings $r_{u,i}$ with respect to some loss measure. The squared loss is a standard choice for the loss function, in which case the following objective function is minimized,

$$J(W,H) = \sum_{(u,i)\in L} \left(r_{u,i} - w_u^T h_i\right)^2 \qquad (6)$$

where $W = [w_1 \ldots w_n]^T$ is an $n \times k$ matrix, $H = [h_1 \ldots h_m]$ is a $k \times m$ matrix, and L is the set of user-item pairs for which the ratings are known. In the impractical limit where all user-item ratings are known, the above objective function is $J(W,H) = \|R - WH\|^2_{fro}$

where R denotes the $n \times m$ fully known user-item matrix. The solution to this problem is given by taking the truncated SVD of R, $R = UDV^T$ and setting $W = U_k D_k^{\frac{1}{2}}, H = D_k^{\frac{1}{2}} V_k^T$ where U_k, D_k, V_k contain the k largest singular triplets of R. However, in the realistic setting where the majority of user-item ratings are unknown and insufficient number of matrix entries are observed, such a nice globally optimal solution cannot in general be directly obtained, and one has to explicitly optimize the non-convex objective function $J(W,H)$. Note that in this case, the objective function is a particular form of weighted loss, that is, $J(W,H) = \|S \odot (R - WH)\|_{fro}^2$ where $\odot$ denotes elementwise products, and S is a binary matrix that equals one over known user-item pairs L, and 0 otherwise. Therefore, weighted low-rank approximations are pertinent to this discussion (Srebro & Jaakkola, 2003). Standard optimization procedures include gradient-based techniques, or procedures like alternating least squares where H is solved keeping W fixed and vice versa until a convergence criterion is satisfied. Note that fixing either W or H turns the problem of estimating the other into a weighted ►linear regression task. In order to avoid learning a model that overfits, it is common to minimize the objective function in the presence of ►regularization terms, $J(W,H) + \gamma\|W\|^2 + \lambda\|H\|^2$, where γ, λ are regularization parameters that can be determined by cross-validation. Once W, H are learnt, the product WH provides an approximate reconstruction of the rating matrix from where recommendations can be directly read off.

Different choices of loss functions, regularizers, and additional model constraints have generated a large body of literature on matrix factorization techniques. Arguably, for discrete ratings, the squared loss is not the most natural loss function. The maximum margin matrix factorization (Rennie & Srebro, 2005) approach uses margin-based loss functions such as the hinge loss used in ►SVM classification, and its ordinal extensions for handling multiple ordered rating categories. For ratings that span over K values, this reduces to finding $K-1$ thresholds that divide the real line into consecutive intervals specifying rating bins to which the output is mapped, with a penalty for insufficient margin of separation. Rennie and Srebro (2005) suggest a nonlinear conjugate gradient algorithm to minimize a smoothed version of this objective function.

Another class of techniques is the nonnegative matrix factorization popularized by the work of Lee and Seung (1999) where nonnegativity constraints are imposed on W, H. There are weighted extensions of NMF that can be applied to recommendation problems. The rating behavior of each user may be viewed as being a manifestation of different roles, for example, a composition of prototypical behavior in clusters of users bound by interests or community. Thus, the ratings of each user are an additive sum of basis vectors of ratings in the item space. By disallowing subtractive basis, nonnegativity constraints lend a "part-based" interpretation to the model. NMF can be solved with a variety of loss functions, but with the generalized KL-divergence loss defined as follows,

$$J(W,H) = \sum_{u,i \in L} r_{u,i} \log \frac{r_{u,i}}{w_u^T h_i} - r_{u,i} + w_u^T h_i$$

NMF is in fact essentially equivalent to probabilistic latent semantic analysis (pLSA) which has also previously been used for collaborative filtering tasks (Hofmann, 2004).

The recently concluded million-dollar Netflix competition has catapulted matrix factorization techniques to the forefront of recommender technologies in collaborative filtering settings (Bell et al., 2009). While the final winning solution was a complex ensemble of different models, several enhancements to basic matrix factorization models were found to lead to improvements. These included:

1. The use of additional user-specific and item-specific parameters to account for systematic biases in the ratings such as popular movies receiving higher ratings on average.
2. Incorporating temporal dynamics of rating behavior by introducing time-dependent variables.

In many settings, only implicit preferences are available, as opposed to explicit like–dislike ratings. For example, large business organizations, typically, meticulously record transactional details of products purchased by their clients. This is a one-class setting since the business domain knowledge for negative examples – that a client has no interest in buying a product ever in the future – is typically not available explicitly in

corporate databases. Moreover, such knowledge is difficult to gather and maintain in the first place, given the rapidly changing business environment. Another example is recommending TV shows based on watching habits of users, where preferences are implicit in what the users chose to see without any source of explicit ratings. Recently, matrix factorization techniques have been advanced to handle such problems (Pan & Scholz, 2009) by formulating confidence weighted objective function, $J(W,H) = \sum_{(u,i)} c_{u,i} \left(r_{u,i} - w_u^T h_i\right)^2$, under the assumption that unobserved user-item pairs may be taken as negative examples with a certain degree of confidence specified via $c_{u,i}$.

The problem of recovering missing values in a matrix from a small fraction of observed entries is also known as the Matrix Completion problem. Recent work by Candès & Tao (2009) and Recht (2009) has shown that under certain assumptions on the singular vectors of the matrix, the matrix completion problem can be solved exactly by a convex optimization problem provided with a sufficient number of observed entries. This problem involves finding among all matrices consistent with the observed entries, the one with the minimum nuclear norm (sum of singular values).

Content-based Recommending

Pure collaborative filtering recommenders only utilize the user ratings matrix, either directly, or to induce a collaborative model. These approaches treat all users and items as atomic units, where predictions are made without regard to the specifics of individual users or items. However, one can make a better personalized recommendation by knowing more about a user, such as demographic information (Pazzani, 1999), or about an item, such as the director and genre of a movie (Melville, Mooney, & Nagarajan, 2002). For instance, given movie genre information, and knowing that a user liked "Star Wars" and "Blade Runner," one may infer a predilection for science fiction and could hence recommend "Twelve Monkeys." Content-based recommenders refer to such approaches, that provide recommendations by comparing representations of content describing an item to representations of content that interests the user. These approaches are sometimes also referred to as *content-based filtering*.

Much research in this area has focused on recommending items with associated *textual* content, such as web pages, books, and movies; where the web pages themselves or associated content like descriptions and user reviews are available. As such, several approaches have treated this problem as an information retrieval (IR) task, where the content associated with the user's preferences is treated as a query, and the unrated documents are scored with relevance/similarity to this query (Balabanovic & Shoham, 1997). In NewsWeeder (Lang, 1995), documents in each rating category are converted into *tf-idf* word vectors, and then averaged to get a prototype vector of each category for a user. To classify a new document, it is compared with each prototype vector and given a predicted rating based on the cosine similarity to each category.

An alternative to IR approaches, is to treat recommending as a classification task, where each example represents the content of an item, and a user's past ratings are used as labels for these examples. In the domain of book recommending, Mooney and Roy (2000) use text from fields such as the title, author, synopses, reviews, and subject terms, to train a multinomial ►naïve Bayes classifier. Ratings on a scale of 1 to k can be directly mapped to k classes (Melville et al., 2002), or alternatively, the numeric rating can be used to weight the training example in a probabilistic binary classification setting (Mooney & Roy, 2000). Other classification algorithms have also been used for purely content-based recommending, including ►k-nearest neighbor, ►decision trees, and ►neural networks (Pazzani & Billsus, 1997).

Hybrid Approaches

In order to leverage the strengths of content-based and collaborative recommenders, there have been several hybrid approaches proposed that combine the two. One simple approach is to allow both content-based and collaborative filtering methods to produce separate ranked lists of recommendations, and then merge their results to produce a final list (Cotter & Smyth, 2000). Claypool, Gokhale, and Miranda (1999) combine the two predictions using an adaptive weighted average, where the weight of the collaborative component increases as the number of users accessing an item increases.

Melville et al. (2002) proposed a general framework for *content-boosted collaborative filtering*, where content-based predictions are applied to convert a

sparse user ratings matrix into a full ratings matrix, and then a CF method is used to provide recommendations. In particular, they use a Naïve Bayes classifier trained on documents describing the rated items of each user, and replace the unrated items by predictions from this classifier. They use the resulting *pseudo ratings matrix* to find neighbors similar to the active user, and produce predictions using Pearson correlation, appropriately weighted to account for the overlap of actually rated items, and for the active user's content predictions. This approach has been shown to perform better than pure collaborative filtering, pure content-based systems, and a linear combination of the two. Within this content-boosted CF framework, Su, Greiner, Khoshgoftaar, and Zhu (2007) demonstrated improved results using a stronger content-predictor, TAN-ELR, and unweighted Pearson collaborative filtering.

Several other hybrid approaches are based on traditional collaborative filtering, but also maintain a content-based profile for each user. These content-based profiles, rather than co-rated items, are used to find similar users. In Pazzani's approach (Pazzani, 1999), each user-profile is represented by a vector of weighted words derived from positive training examples using the Winnow algorithm. Predictions are made by applying CF directly to the matrix of user-profiles (as opposed to the user-ratings matrix). An alternative approach, Fab (Balabanovic & Shoham, 1997), uses ►relevance feedback to simultaneously mold a personal filter along with a communal "topic" filter. Documents are initially ranked by the topic filter and then sent to a user's personal filter. The user's relevance feedback is used to modify both the personal filter and the originating topic filter. Good et al. (1999) use collaborative filtering along with a number of personalized information filtering agents. Predictions for a user are made by applying CF on the set of other users and the active user's personalized agents.

Several hybrid approaches treat recommending as a classification task, and incorporate collaborative elements in this task. Basu, Hirsh, and Cohen (1998) use *Ripper*, a ►rule induction system, to learn a function that takes a user and movie and predicts whether the movie will be liked or disliked. They combine collaborative and content information, by creating features such as *comedies liked by user* and *users who liked movies of genre X*. In other work, Soboroff and Nicholas (1999) multiply a *term-document matrix* representing all item content with the user-ratings matrix to produce a *content-profile matrix*. Using latent semantic Indexing, a rank-*k* approximation of the content-profile matrix is computed. Term vectors of the user's relevant documents are averaged to produce a user's profile. Then, new documents are ranked against each user's profile in the LSI space.

Some hybrid approaches attempt to directly combine content and collaborative data under a single probabilistic framework. Popescul, Ungar, Pennock, and Lawrence (2001) extended Hofmann's *aspect model* (Hofmann, 1999) to incorporate a three-way co-occurrence data among users, items, and item content. Their generative model assumes that users select latent topics, and documents and their content words are generated from these topics. Schein, Popescul, Ungar, and Pennock (2002) extend this approach, and focus on making recommendations for items that have not been rated by any user.

Evaluation Metrics

The quality of a recommender system can be evaluated by comparing recommendations to a test set of known user ratings. These systems are typical measured using *predictive accuracy metrics* (Herlocker, Konstan, Terveen, & Riedl, 2004), where the predicted ratings are directly compared to actual user ratings. The most commonly used metric in the literature is ►Mean Absolute Error (MAE) – defined as the average absolute difference between predicted ratings and actual ratings, given by:

$$MAE = \frac{\sum_{\{u,i\}} |p_{u,i} - r_{u,i}|}{N} \quad (7)$$

Where $p_{u,i}$ is the predicted rating for user u on item i, $r_{u,i}$ is the actual rating, and N is the total number of ratings in the test set.

A related commonly used metric, ►Root Mean Squared Error (RMSE), puts more emphasis on larger absolute errors, and is given by:

$$RMSE = \sqrt{\frac{\sum_{\{u,i\}} (p_{u,i} - r_{u,i})^2}{N}} \quad (8)$$

Predictive accuracy metrics treat all items equally. However, for most recommender systems the primary

concern is accurately predict the items a user will like. As such, researchers often view recommending as predicting *good*, that is, items with high ratings versus *bad* or poorly rated items. In the context of information retrieval (IR), identifying the good from the background of bad items can be viewed as discriminating between "relevant" and "irrelevant" items; and as such, standard IR measures, like ►Precision, ►Recall and ►Area Under the ROC Curve (AUC) can be utilized. These, and several other measures, such as *F1-measure*, *Pearson's product-moment correlation*, *Kendall's* τ, *mean average precision*, *half-life utility*, and *normalized distance-based performance measure* are discussed in more detail by Herlocker et al. (2004).

Challenges and Limitations

This section, presents some of the common hurdles in deploying recommender systems, as well as some research directions that address them.

Sparsity: Stated simply, most users do not rate most items and, hence, the user ratings matrix is typically very sparse. This is a problem for collaborative filtering systems, since it decreases the probability of finding a set of users with similar ratings. This problem often occurs when a system has a very high item-to-user ratio, or the system is in the initial stages of use. This issue can be mitigated by using additional domain information (Melville et al., 2002; Su et al., 2007) or making assumptions about the data generation process that allows for high-quality imputation (Su et al., 2008).

The Cold-Start Problem: New items and new users pose a significant challenge to recommender systems. Collectively these problems are referred to as the *cold-start problem* (Schein et al., 2002). The first of these problems arises in collaborative filtering systems, where an item cannot be recommended unless some user has rated it before. This issue applies not only to new items, but also to obscure items, which is particularly detrimental to users with eclectic tastes. As such the *new-item problem* is also often referred to as the *first-rater problem*. Since content-based approaches (Mooney & Roy, 2000; Pazzani & Billsus, 1997) do not rely on ratings from other users, they can be used to produce recommendations for *all* items, provided attributes of the items are available. In fact, the content-based predictions of similar users can also be used to further improve predictions for the active user (Melville et al., 2002).

The *new-user problem* is difficult to tackle, since without previous preferences of a user it is not possible to find similar users or to build a content-based profile. As such, research in this area has primarily focused on effectively selecting items to be rated by a user so as to rapidly improve recommendation performance with the least user feedback. In this setting, classical techniques from ►active learning can be leveraged to address the task of item selection (Harpale & Yang, 2008; Jin & Si, 2004).

Fraud: As recommender systems are being increasingly adopted by commercial websites, they have started to play a significant role in affecting the profitability of sellers. This has led to many unscrupulous vendors engaging in different forms of fraud to game recommender systems for their benefit. Typically, they attempt to inflate the perceived desirability of their own products (*push attacks*) or lower the ratings of their competitors (*nuke attacks*). These types of attack have been broadly studied as *shilling attacks* (Lam & Riedl, 2004) or *profile injection attacks* (Burke, Mobasher, Bhaumik, & Williams, 2005). Such attacks usually involve setting up dummy profiles, and assume different amounts of knowledge about the system. For instance, the *average attack* (Lam & Riedl, 2004) assumes knowledge of the average rating for each item; and the attacker assigns values randomly distributed around this average, along with a high rating for the item being *pushed*. Studies have shown that such attacks can be quite detrimental to predicted ratings, though *item-based* collaborative filtering tends to be more robust to these attacks (Lam & Riedl, 2004). Obviously, content-based methods, which only rely on a users past ratings, are unaffected by profile injection attacks.

While pure content-based methods avoid some of the pitfalls discussed above, collaborative filtering still has some key advantages over them. Firstly, CF can perform in domains where there is not much content associated with items, or where the content is difficult for a computer to analyze, such as ideas, opinions, etc. Secondly, a CF system has the ability to provide

serendipitous recommendations, that is, it can recommend items that are relevant to the user, but do not contain content from the user's profile.

Recommended Reading

Good surveys of the literature in the field can be found in Adomavicius and Tuzhilin (2005); Bell et al. (2009); Su and Khoshgoftaar (2009). For extensive empirical comparisons on variations of Collaborative Filtering refer to Breese (1998), Herlocker (1999), Sarwar et al. (2001).

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering, 17*(6), 734–749.

Balabanovic, M., & Shoham, Y. (1997). Fab: Content-based, collaborative recommendation. *Communications of the Association for Computing Machinery, 40*(3), 66–72.

Basu, C., Hirsh, H., & Cohen, W. (July 1998). Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the fifteenth national conference on artificial intelligence (AAAI-98), Madison, Wisconsin* (pp. 714–720).

Bell, R., Koren, Y., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer 42*(8): 30–37.

Billsus, D., & Pazzani, M. J. (1998). Learning collaborative information filters. In *Proceedings of the fifteenth international conference on machine learning (ICML-98), Madison, Wisconsin* (pp. 46–54). San Francisco: Morgan Kaufmann.

Breese, J. S., Heckerman, D., & Kadie, C. (July 1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the fourteenth conference on uncertainty in artificial intelligence, Madison, Wisconsin*.

Burke, R., Mobasher, B., Bhaumik, R., & Williams, C. (2005). Segment-based injection attacks against collaborative filtering recommender systems. In *ICDM '05: Proceedings of the fifth IEEE international conference on data mining* (pp. 577–580). Washington, DC: IEEE Computer Society. Houston, Texas.

Candès, E. J., & Tao, T. (2009). The power of convex relaxation: Near-optimal matrix completion. *IEEE Trans. Inform. Theory, 56*(5), 2053–2080.

Claypool, M., Gokhale, A., & Miranda, T. (1999). Combining content-based and collaborative filters in an online newspaper. In *Proceedings of the SIGIR-99 workshop on recommender systems: algorithms and evaluation*.

Cotter, P., & Smyth, B. (2000). PTV: Intelligent personalized TV guides. In *Twelfth conference on innovative applications of artificial intelligence, Austin, Texas* (pp. 957–964).

Goldberg, D., Nichols, D., Oki, B., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the Association of Computing Machinery, 35*(12), 61–70.

Good, N., Schafer, J. B., Konstan, J. A., Borchers, A., Sarwar, B., Herlocker, J., et al. (July 1999). Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the sixteenth national conference on artificial intelligence (AAAI-99), Orlando, Florida* (pp. 439–446).

Harpale, A. S., & Yang, Y. (2008). Personalized active learning for collaborative filtering. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval, Singapore* (pp. 91–98). New York: ACM.

Herlocker, J., Konstan, J., Borchers, A., & Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of 22nd international ACM SIGIR conference on research and development in information retrieval, Berkeley, California* (pp. 230–237). New York: ACM.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems, 22*(1), 5–53.

Hofmann, T. (1999). Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Stockholm, Sweden, July 30-August 1, 1999 Morgan Kaufmann.

Hofmann, T. (2004). Latent semantic analysis for collaborative filtering. *ACM Transactions on Information Systems, 22*(1), 89–115.

Jin, R., & Si, L. (2004). A Bayesian approach toward active learning for collaborative filtering. In *UAI '04: Proceedings of the 20th conference on uncertainty in artificial intelligence, Banff, Canada* (pp. 278–285). Arlington: AUAI Press.

Lam, S. K., & Riedl, J. (2004). Shilling recommender systems for fun and profit. In *WWW '04: Proceedings of the 13th international conference on World Wide Web, New York* (pp. 393–402). New York: ACM.

Lang, K. (1995). NewsWeeder: Learning to filter netnews. In *Proceedings of the twelfth international conference on machine learning (ICML-95)* (pp. 331–339). San Francisco. Tahoe City, CA, USA. Morgan Kaufmann, ISBN 1-55860-377-8.

Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature, 401*, 788.

Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing, 7*(1), 76–80.

Melville, P., Mooney, R. J., & Nagarajan, R. (2002). Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the eighteenth national conference on artificial intelligence (AAAI-02), Edmonton, Alberta* (pp. 187–192).

Mooney, R. J., & Roy, L. (June 2000). Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on digital libraries, San Antonio, Texas* (pp. 195–204).

Nakamura, A., & Abe, N. (1998). Collaborative filtering using weighted majority prediction algorithms. In *ICML '98: Proceedings of the fifteenth international conference on machine learning* (pp. 395–403). San Francisco: Morgan Kaufmann. Madison, Wisconsin.

Pan, R., & Scholz, M. (2009). Mind the gaps: Weighting the unknown in large-scale one-class collaborative filtering. In *15th ACM SIGKDD conference on knowledge discovery and data mining (KDD), Paris, France*.

Pazzani, M. J. (1999). A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review, 13*(5–6), 393–408.

Pazzani, M. J., & Billsus, D. (1997). Learning and revising user profiles: The identification of interesting web sites. *Machine Learning, 27*(3), 313–331.

Popescul, A., Ungar, L., Pennock, D. M., & Lawrence, S. (2001). Probabilistic models for unified collaborative and content-based

recommendation in sparse-data environments. In *Proceedings of the seventeenth conference on uncertainity in artificial intelligence*. University of Washington, Seattle.

Recht, B. (2009). A Simpler Approach to Matrix Completion. Benjamin Recht. (to appear in Journal of Machine Learning Research).

Rennie, J., & Srebro, N. (2005). Fast maximum margin matrix factorization for collaborative prediction. In *International conference on machine learning, Bonn, Germany*.

Resnick, P., Iacovou, N., Sushak, M., Bergstrom, P., & Reidl, J. (1994a). GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 computer supported cooperative work conference, New York*. New York: ACM.

Resnick, P., Neophytos, I., Bergstrom, P., Mitesh, S., & Riedl, J. (1994b). Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW94 – Conference on computer supported cooperative work, Chapel Hill* (pp. 175–186). Addison-Wesley.

Sarwar, B., Karypis, G., Konstan, J., & Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the tenth international conference on World Wide Web* (pp. 285–295). New York: ACM. Hong Kong.

Schein, A. I., Popescul, A., Ungar, L. H., & Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 253–260). New York: ACM. Tampere, Finland.

Soboroff, I., & Nicholas, C. (1999). Combining content and collaboration in text filtering. In T. Joachims (Ed.), *Proceedings of the IJCAI'99 workshop on machine learning in information filtering* (pp. 86–91).

Srebro, N., & Jaakkola, T. (2003). Weighted low-rank approximations. In *International conference on machine learning (ICML)*. Washington DC.

Su, X., Greiner, R., Khoshgoftaar, T. M., & Zhu, X. (2007). Hybrid collaborative filtering algorithms using a mixture of experts. In *Web intelligence* (pp. 645–649).

Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence, 2009*, 1–20.

Su, X., Khoshgoftaar, T. M., Zhu, X., & Greiner, R. (2008). Imputation-boosted collaborative filtering using machine learning classifiers. In *SAC '08: Proceedings of the 2008 ACM symposium on applied computing* (pp. 949–950). New York: ACM.

Record Linkage

►Entity Resolution

Recurrent Associative Memory

►Hopfield Network

Recursive Partitioning

►Divide-and-Conquer Learning

Reference Reconciliation

►Entity Resolution

Regression

Novi Quadrianto, Wray L. Buntine
RSISE, ANU and SML, NICTA, Canberra, Australia

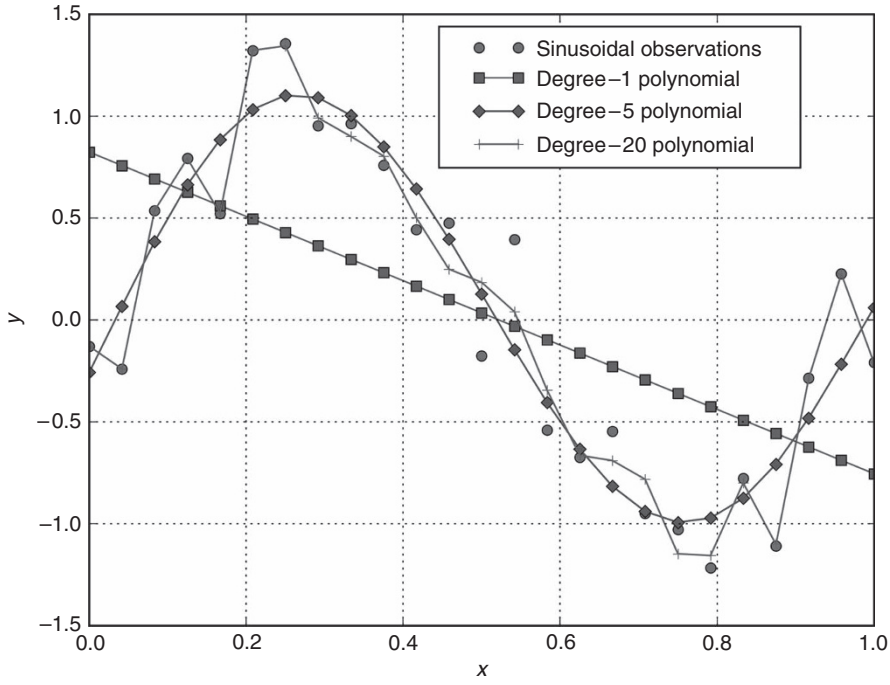
Definition

Regression is a fundamental problem in statistics and machine learning. In regression studies, we are typically interested in inferring a real-valued function (called a regression function) whose values correspond to the mean of a dependent (or response or output) variable conditioned on one or more independent (or input) variables. Many different techniques for estimating this regression function have been developed, including parametric, semi-parametric, and nonparametric methods.

Motivation and Background

Assume that we are given a set of data points sampled from an underlying but unknown distribution, each of which includes input x and output y. An example is given in Fig. 1. The task of regression is to learn a hidden functional relationship between x and y from observed and possibly noisy data points. In Fig. 1, the input–output relationship is a Gaussian corrupted sinusoidal relationship, that is $y = \sin(2\pi x) + \epsilon$ where ϵ is normally distributed noise. Various lines show the inferred relationship based on a linear parametric regression model with polynomial basis functions. The higher the degree of the polynomial, the more complex is the inferred relationship, as shown in Fig. 1, as the function tries to better fit the observed data points.

While the most complex polynomial here is an almost perfect reconstruction of observed data points (it has "low bias"), it gives a very poor representation of



Regression. Figure 1. 25 data points (one-dimensional input *x* and output *y* variables) with a Gaussian corrupted sinusoidal input–output relationship, $y = \sin(2\pi x) + \epsilon$ where ϵ is normally distributed noise. The task is to learn the functional relationship between *x* and *y*. Various lines show the inferred relationship based on a linear regression model with polynomial basis functions having various degrees

the true underlying function $\sin(2\pi x)$ that can change significantly with the change of a few data points (it has "high variance"). This phenomemon is called the ►*bias-variance dilemma*, and selecting a complex model with too high a variance is called ►*over-fitting*. Complex parametric models (like polynomial regression) lead to low bias estimators with a high variance, while simple models lead to low variance estimators with high bias. To sidestep the problem of trying to estimate or select the model complexity represented for instance by the degree of the polynomial, so-called nonparametric methods allow a rich variety of functions from the outset (*i.e.*, a function class not finitely parameterizable) and usually provide a hyperparameter that tunes the regularity, curvature, or complexity of the function.

Theory/Solution

Formally, in a regression problem, we are interested in recovering a functional dependency $y_i = f(x_i) + \epsilon_i$ from N observed training data points $\{(x_i, y_i)\}_{i=1}^N$, where $y_i \in \mathbb{R}$ is the noisy observed output at input location $x_i \in \mathbb{R}^d$. For ►Linear Regression, we represent the regression function $f()$ by a parameter $w \in \mathbb{R}^H$ in the form $f(x_i) := \langle \phi(x_i), w \rangle$ for H fixed basis functions $\{\phi_h(x_i)\}_{h=1}^H$. With general basis functions such as polynomials, exponentials, sigmoids, or even more sophisticated Fourier or wavelets bases, we can obtain a regression function which is nonlinear with regard to the input variables although still linear with regard to the parameters.

In regression, many more methods are possible. Some variations on these standard linear models are piecewise linear models, trees, and splines (roughly, piecewise polynomial models joined up smoothly) (Hastie, Tibshirani, & Friedman, 2003). These are called semi-parametric models, because they have a linear parametric component as well as a nonparametric component.

Fitting In general, regression fits a model to data using an objective function or quality criterion in a form such as

$$E(f) = \sum_{i=1}^{N} \epsilon(y_i, f(x_i)),$$

where smaller $E(f)$ implies better quality. This might be derived as an error/loss function, or as a negative log likelihood or log probability. The squared error function is the most convenient (leading to a least squares calculation), but many possibilities exist. In general, methods are distinguished by three aspects, (1) the representation of the function $f()$, (2) the form of the term $\epsilon(y_i, f(x_i))$, and (3) the penalty term discussed next.

Regularized/Penalized Fitting The issue of over-fitting, as mentioned already in the section Motivation and Background, is usually addressed by introducing a regularization or penalty term to the objective function. The regularized objective function is now in the form of

$$E_{\text{reg}} = E(f) + \lambda R(f). \tag{1}$$

Here, $E(f)$ measures the quality of the solution for $f()$ on the observed data points, $R(f)$ penalizes complexity of $f()$, and λ is called the regularization parameter which controls the relative importance between the two. Measures of function curvature, for instance, can be used for $R(f)$. In standard ▶Support Vector Machines, the term $E(f)$ measures the hinge loss, and penalty $R(f)$ is the sum of squares of the parameters, also used in ridge regression (Hastie et al., 2003).

Bias-Variance Dilemma

As we have seen in the previous section, the introduction of the regularization term can help avoid over-fitting. However, this raises the question of determining an optimal value for the regularization parameter λ. The specific choice of λ controls the bias-variance tradeoff (Geman, Bienenstock, & Doursat, 1992).

Recall that we try to infer a latent regression function $f(x)$ based on N observed training data points $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$. The notation $f(x; \mathcal{D})$ explicitly shows the dependence of f on the data $\mathcal{D}$. The mean squared error (MSE) which measures the effectiveness of f as a predictor of y is

$$\mathbf{E}[(y - f(x;\mathcal{D}))^2 | x, \mathcal{D}] = \mathbf{E}[(y - \mathbf{E}[y|x])^2 | x, \mathcal{D}] + (f(x;\mathcal{D}) - \mathbf{E}[y|x])^2 \tag{2}$$

where $\mathbf{E}[.]$ means expectation with respect to a conditional distribution $p(y|x)$. The first term of (2) does not depend on $f(x; \mathcal{D})$ and it represents the intrinsic noise on the data. The MSE of f as an estimator of the regression $\mathbf{E}[y|x]$ is

$$\mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}[y|x])^2] \tag{3}$$

where $\mathbf{E}_{\mathcal{D}}$ means expectation with respect to the training set $\mathcal{D}$. The estimation error in (3) can be decomposed into a bias and a variance terms, that is

$$\begin{aligned}
&\mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}[y|x])^2] \\
&= \mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})] + \mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})] \\
&\quad - \mathbf{E}[y|x])^2] \\
&= \mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})])^2] + (\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})] \\
&\quad - \mathbf{E}[y|x])^2 \\
&\quad + 2\mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})])](\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})] \\
&\quad - \mathbf{E}[y|x]) \\
&= \mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})])^2] + (\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})] \\
&\quad - \mathbf{E}[y|x])^2 \\
&= \text{variance} + \text{bias}^2.
\end{aligned}$$

The bias term measures the difference between the average predictor over all datasets and the desired regression function. The variance term measures the adaptability of the predictor to a particular dataset. There is a tradeoff between the bias and variance contributions to the estimation error, with very flexible models having low bias but high variance (over-fitting) and relatively rigid models having low variance but high bias (under-fitting). Typically, variance is reduced through "smoothing," that is an introduction of the regularization term. This, however, will introduce bias as peaks and valleys of the regression function will be blurred. To achieve an optimal predictive capability, an estimator with the best balance between bias and variance is chosen by varying the regularization parameter λ. It is crucial to note that bias-variance decomposition albeit powerful is based on averages of datasets, however in practice only a single dataset is observed. In this regard, a Bayesian treatment of regression, such as Gaussian process regression which will avoid over-fitting problem of maximum likelihood and which will also lead

to automatic methods of determining model complexity using the training data alone could be an attractive alternative.

Nonparametric Regression

In the parametric approach, an assumption on the mathematical form of the functional relationship between input x and output y such as linear, polynomial, exponential, or combination of them needs to be chosen a priori. Subsequently, parameters are placed on each of the chosen forms and the optimal values learnt from the observed data. This is restrictive both in the fixed functional form and in the ability to vary the model complexity. Nonparametric approaches try to derive the functional relationship directly from the data, that is, they do not parameterize the regression function.

▶Gaussian Processes for regression, for instance, is well-developed. Another approach is the *kernel method*, of which a rich variety exists (Hastie et al., 2003). These can be viewed as a regression variant of nearest neighbor classification where the function is made up of a local element for each data point:

$$f(x) = \frac{\sum_i y_i K_\lambda(x_i, x)}{\sum_i K_\lambda(x_i, x)},$$

where the function $K_\lambda(x_i,)$ is a nonnegative "bump" in x space centered at its first argument with diameter approximately given by λ. Thus, the function has a variable contribution from each data point, and λ controls the bias-variance tradeoff.

Generalized Linear Models

The previous discussion about regression focuses on continuous output/dependent variables. While this type of regression problem is ubiquitous, there are however some interests in cases of *restricted* output variables:

1. The output variable consists of two categories (called *binomial* regression).
2. The output variable consists of more than two categories (called *multinomial* regression).
3. The output variable consists of more than two categories which can be ordered in a meaningful way (called *ordinal* regression). and
4. The output variable is a count of the repetition of the occurrence of an event (called *poisson* regression).

Nelder and Wedderburn (1972) introduced the generalized linear model (GLM) by allowing the linear model to be related to the output variables via a link function. This is a way to unify different cases of response variables under one framework, each only differs in the choice of the link function. Specifically, in GLM, each output variable is assumed to be generated from the exponential family of distributions. The mean of this distribution depends on the input variables through

$$\mathbf{E}[y] = g(\mu) = w_0 + w_1\phi_1(x_i) + \ldots + w_D\phi_D(x_i), \quad (4)$$

where $g(\mu)$ is the link function (Table 1). The parameters of the generalized linear model can then be estimated by the maximum likelihood method, which can be found by iterative re-weighted least squares (IRLS), an instance of the expectation maximization (EM) algorithm.

Other Variants of Regression

So far, we have focused on the problem of predicting a single output variable y from an input variable x. Some studies look at predicting multiple output variables simultaneously. The simplest approach for the *multiple outputs* problem would be to model each output variable with a different set of basis functions. The more common approach uses the same set of basis functions to model all of the output variables. Not surprisingly,

Regression. Table 1 A table of Various Link Functions Associated with the Assumed Distribution on the Output Variable

Distribution of Dependent Variable	Name	Link Function
Gaussian	Identity link	$g(\mu) = \mu$
Poisson	Log link	$g(\mu) = \log(\mu)$
Binomial Multinomial	Logit link	$g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$
Exponential Gamma	Inverse link	$g(\mu) = \mu^{-1}$
Inverse Gaussian	Inverse squared link	$g(\mu) = \mu^{-2}$

the solution to the multiple outputs problem decouples into independent regression problems with shared basis functions.

For some other studies, the focus of regression is on computing several regression functions corresponding to various percentage points or quantiles (instead of the mean) of the conditional distribution of the dependent variable given the independent variables. This type of regression is called *quantile* regression (Koenker, 2005). Sum of tilted absolute loss (called pinball loss) is being optimized for this type of regression. Quantile regression has many important applications within econometrics, data mining, social sciences, and ecology, among other domains.

Instead of inferring one regression function corresponding to the mean of a response variable, k regression functions can be computed with the assumption that the response variable is generated by a mixture of k components. This is called the *mixture of regressions* problem (Gaffney & Smyth, 1999). Applications include trajectory clustering, robot planning, and motion segmentation.

Another important variant is the *heteroscedastic* regression model where the noise variance on the data is a function of the input variable x. The Gaussian process framework can be used conveniently to model this noise-dependent case by introducing a second Gaussian process to model the dependency of noise variance on the input variable (Goldberg, Williams, & Bishop, 1998). There are also attempts to make the regression model more robust to the presence of a few problematic data points called outliers. Sum of absolute loss (instead of sum of squared loss) or student's t-distribution (instead of Gaussian distribution) can be used for *robust* regression.

Cross References

►Gaussian Processes
►Linear Regression
►Support Vector Machines

Recommended Reading

Machine learning textbooks such as Bishop (2006), among others, introduce different regression models. For a more statistical introduction including an extensive overview of the many different semi-parametric methods and non-parametric methods such as kernel methods see Hastie et al. (2003). For a coverage of key statistical issues including nonlinear regression, identifiability, measures of curvature, autocorrelation, and such, see Seber and Wild (1989). For a large variety of built-in regression techniques, refer to R (http://www.r-project.org/).

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer.
Gaffney, S., & Smyth, P. (1999). Trajectory clustering with mixtures of regression models. In *ACM SIGKDD* (Vol. 62, pp. 63–72). ACM
Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*, 1–58.
Goldberg, P., Williams, C., & Bishop, C. (1998). Regression with input-dependent noise: A Gaussian process treatment. In *Neural information processing systems* (Vol. 10). The MIT Press
Hastie, T., Tibshirani, R., & Friedman, J. (2003). *The elements of statistical learning: Data mining, inference, and prediction* (Corrected ed.). Springer.
Koenker, R. (2005). *Quantile regression*. Cambridge University Press.
Nelder, J. A., & Wedderburn, R. W. M. (1972). Generalized linear models. *Journal of the Royal Statistical Society: Series A, 135*, 370–384.
Seber, G., & Wild, C. (1989). *Nonlinear regression*. New York: Wiley.

Regression Trees

Luís Torgo
University of Porto, Rua Campo Alegre, Porto, Portugal

Synonyms

Decision trees for regression; Piecewise constant models; Tree-based regression

Definition

Regression trees are supervised learning methods that address multiple regression problems. They provide a tree-based approximation $\hat{f}$, of an unknown regression function $Y = f(\mathbf{x}) + \varepsilon$ with $Y \in \mathbb{R}$ and $\varepsilon \approx N(0, \sigma^2)$, based on a given sample of data $D = \{\langle x_{i,1}, \ldots, x_{i,p}, y_i \rangle\}_{i=1}^{n}$. The obtained models consist of a hierarchy of logical tests on the values of any of the p predictor variables. The terminal nodes of these trees, known as the leaves, contain the numerical predictions of the model for the target variable Y.

Motivation and Background

Work on regression trees goes back to the AID system by Morgan and Sonquist Morgan (1963).

Nonetheless, the seminal work is the book *Classification and Regression Trees* by Breiman and colleagues (Breiman, Friedman, Olshen, & Stone, 1984). This book has established several standards in many theoretical aspects of tree-based regression, including over-fitting avoidance by post-pruning, the notion of surrogate splits for handling unknown variable, and estimating variable importance.

Regression trees have several features that make them a very interesting approach to several multiple regression problems. For example, regression trees provide: (a) automatic variable selection making them highly insensitive to irrelevant variables; (b) computational efficiency that allows addressing large problems; (c) handling of unknown variable values; (d) handling of both numerical and nominal predictor variables; (e) insensitivity to predictors' scales; and (f) interpretable models for most domains. In spite of all these advantages, regression trees have poor prediction accuracy in several domains because of the piecewise constant approximation they provide.

Structure of Learning System

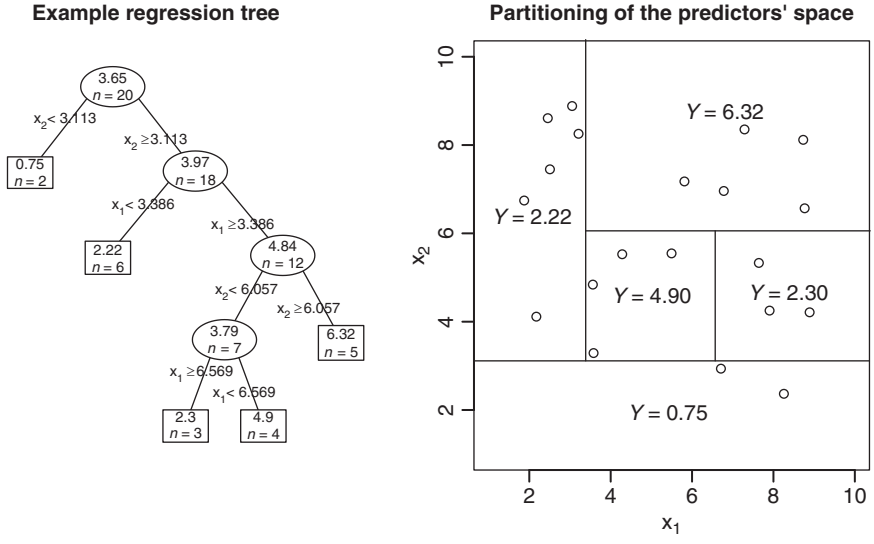
The most common regression trees are binary with logical tests in each node (an example is given on the left graph of Fig. 1). Tests on numerical variables usually take the form $x_i < \alpha$, with $\alpha \in \mathbb{R}$, while tests on nominal variables have the form $x_j \in \{v_1, \ldots, v_m\}$. Each path from the root (top) node to a leaf can be seen as a logical assertion defining a region on the predictors' space. Any regression tree provides a full mutually exclusive partition of the predictor space into L regions with boundaries that are parallel to the predictors' axes due to the form of the tests. Figure 1 illustrates these ideas with a tree and the respective partitioning on the right side of the graph. All observations in a partition are predicted with the same constant value, and that is the reason for regression trees sometimes being referred to as piecewise constant models.

Using a regression tree for obtaining predictions for new observations is straightforward. For each new observation a path from the root node to a leaf is followed, selecting the branches according to the variable values of the observation. The leaf contains the prediction for the observation.

Learning a Regression Tree

A binary regression tree is obtained by a very efficient algorithm known as recursive partitioning (Algorithm 1).

If the termination criterion is not met by the input sample D, the algorithm selects the best logical test on one of the predictor variables according to some criterion. This test divides the current sample into two partitions: the one with the cases satisfying the test, and the remaining. The algorithm proceeds by recursively



Regression Trees. Figure 1. A regression tree and the partitioning it provides

Algorithm 1 Recursive Partitioning.

```
1: function RecursivePartitioning(D)
   Input :  D, a sample of cases, {⟨x_{i,1}, ..., x_{i,p}, y_i⟩}
   Output :  t, a tree node

2:   if <termination criterion> then
3:     Return a leaf node with <constant k>
4:   else
5:     t ← new tree node
6:     t.split ← <Find the best test on one of the variables>
7:     t.leftNode ← RecursivePartitioning(x ∈ D : x ⊨ t.split)
8:     t.rightNode ← RecursivePartitioning(x ∈ D : x ⊭ t.split)
9:     Return the node t
10:  end if
11: end function
```

applying the same method to these two partitions to obtain the left and right branches of the node. Algorithm 1 has three main components that characterize the type of regression tree we are obtaining: (a) the *termination criterion*; (b) the *constant k*; and (c) the method to find the *best test on one of the predictors*. The choices for these components are related to the preference criteria that are selected to build the trees. The most common criterion is the minimization of the sum of the square errors, known as the least squares (LS) criterion. Using this criterion it can be easily proven (e.g., Breiman et al., 1984) that the constant k should be the average target variable value of the cases in the leaf. With respect to the *termination criterion*, usually very relaxed settings are selected so that an overly large tree is grown. The reasoning is that the trees will be pruned afterward with the goal of overcoming the problem of over-fitting of the training data.

According to the LS criterion the error in a given node is given by,

$$Err(t) = \frac{1}{n_t} \sum_{\langle \mathbf{x}_i, y_i \rangle \in D_t} (y_i - k_t)^2 \qquad (1)$$

where D_t is the sample of cases in node t, n_t is the cardinality of this set and k_t is the average target variable value of the cases in D_t.

Any logical test s divides the cases in D_t into two partitions, D_{t_L} and D_{t_R}. The resulting pooled error is given by,

$$Err(t,s) = \frac{n_{t_L}}{n_t} \times Err(t_L) + \frac{n_{t_R}}{n_t} \times Err(t_R) \qquad (2)$$

where n_{t_L}/n_t (n_{t_R}/n_t) is the proportion of cases going to the left (right) branch of t.

In this context, we can estimate the value of the split s by the respective error reduction, and this can be used to evaluate all candidate split tests for a node,

$$\Delta(s,t) = Err(t) - Err(t,s) \qquad (3)$$

Finding the best split test for a node t involves evaluating all possible tests for this node using Eq. (3). For each predictor of the problem one needs to evaluate all possible splits in that variable. For continuous variables this requires a sorting operation on the values of this variable occurring in the node. After this sorting, a fast incremental algorithm can be used to find the best cut-point value for the test (e.g. Torgo, 1999). With respect to nominal variables, Breiman et al. (1984) have proved a theorem that avoids trying all possible combinations of values, reducing the computational complexity of this task from $O(2^{v-1}-1)$ to $O(v-1)$, where v is the number of values of the nominal variable.

Departures from the standard learning procedure described above include, among others: the use of multivariate split nodes (e.g., Breiman et al., 1984; Gama, 2004; Li, Lue, & Chen, 2000) to overcome the axis parallel representation limitation of partitions; the use of different criteria to find the best split node (e.g., Buja & Lee, 2001; Loh, 2002; Robnik-Sikonja & Kononenko, 1996); the use of different preference criteria to guide the tree growth (e.g., Breiman et al., 1984; Buja & Lee, 2001; Torgo, 1999; Torgo & Ribeiro, 2003); and the use of both regression and split nodes (e.g., Lubinsky, 1995; Malerba, Esposito, Ceci, & Appice, 2004).

Pruning Regression Trees

As most nonparametric modeling techniques, regression trees may ▶over-fit the training data which will inevitably lead to poor out-of-the-sample predictive performance. The standard procedure to fight this undesirable effect is to grow an overly large tree and then to use some reliable error estimation procedure to

find the "best" sub-tree of this large model. This procedure is known as post-pruning a tree (Breiman et al., 1984). An alternative is to stop tree growth sooner in a process known as pre-pruning, which again needs to be guided by reliable error estimation to know when over-fitting is starting to occur. Although more efficient in computational terms, this latter alternative may lead to stop tree growth too soon even with look-ahead mechanisms.

Post-pruning is usually carried out in a three stages procedure: (a) a set of sub-trees of the initial tree is generated; (b) some reliable error estimation procedure is used to obtain estimates for each member of this set; and (c) some method based on these estimates is used to select one of these trees as the final tree model. Different methods exist for each of these steps. A common setup (e.g., Breiman et al., 1984) is to use error-complexity pruning to generate a sequence of nested sub-trees, whose error is then estimated by cross validation. The final tree is selected using the x-SE rule, which starts with the lowest estimated error sub-tree and then selects the smallest tree within the interval of x standard errors of the lowest estimated error tree (a frequent setting is to use one standard error).

Variations on the subject of pruning regression trees include, among others: pre-pruning alternatives (e.g., Breiman & Meisel, 1976; Friedman, 1979); the use of different tree error estimators (see Torgo (1998) for a comparative study and references to different alternatives); and the use of the Minimum Description Length (MDL) principle to guide the pruning (Robnik-Sikonja & Kononenko, 1998).

Cross References

►Model Trees
►Out-of-the-Sample
►Random Forests
►Regression
►Supervised Learning
►Training Sample

Recommended Reading

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees. Statistics/probability series*. Wadsworth & Brooks/Cole Advanced Books & Software.

Breiman, L., & Meisel, W. S. (1976). General estimates of the intrinsic variability of data in nonlinear regression models. *Journal of the American Statistical Association, 71*, 301–307.

Buja, A., & Lee, Y.-S. (2001). Data mining criteria for tree-based regression and classification. In *Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 27–36). San Francisco, California, USA.

Friedman, J. H. (1979). A tree-structured approach to nonparametric multiple regression. In T. Gasser & M. Rosenblatt (Eds.), *Smoothing techniques for curve estimation. Lecture notes in mathematics* (Vol. 757, pp. 5–22). Berlin: Springer.

Gama, J. (2004). Functional trees. *Machine Learning, 55*(3), 219–250.

Li, K. C., Lue, H., & Chen, C. (2000). Interactive tree-structured regression via principal Hessians direction. *Journal of the American Statistical Association, 95*, 547–560.

Loh, W. (2002). Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica, 12*, 361–386.

Lubinsky, D. (1995). Tree structured interpretable regression. In *Proceedings of the workshop on AI & statistics*.

Malerba, D., Esposito, F., Ceci, M., & Appice, A. (2004). Top-down induction of model trees with regression and splitting nodes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 26*(5), 612–625.

Morgan, J. N., & Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of American Statistical Association, 58*(302), 415–434.

Robnik-Sikonja, M., & Kononenko, I. (1996). Context-sensitive attribute estimation in regression. In *Proceedings of the ICML-96 workshop on learning in context-sensitive domains*. Brighton, UK.

Robnik-Sikonja, M., & Kononenko, I. (1998). Pruning regression trees with MDL. In *Proceedings of ECAI-98*. Brighton, UK.

Torgo, L. (1998). Error estimates for pruning regression trees. In C. Nedellec & C. Rouveirol (Eds.), *Proceedings of the tenth European conference on machine learning. LNAI* (Vol. 1398). London, UK: Springer-Verlag.

Torgo, L. (1999). *Inductive learning of tree-based regression models*. PhD thesis, Department of Computer Science, Faculty of Sciences, University of Porto.

Torgo, L., & Ribeiro, R. (2003). Predicting outliers. In N. Lavrac, D. Gamberger, L. Todorovski, & H. Blockeel (Eds.), *Proceedings of principles of data mining and knowledge discovery (PKDD'03). LNAI* (Vol. 2838, pp. 447–458). Berlin/Heidelberg: Springer-Verlag.

Regularization

Xinhua Zhang
Australian National University, NICTA London Circuit Canberra, Australia

Definition

Regularization plays a key role in many machine learning algorithms. Exactly fitting a model to the training data is generally undesirable, because it will fit the noise

in the training examples (►overfitting), and is doomed to predict (generalize) poorly on unseen data. In contrast, a simple model that fits the training data well is more likely to capture the regularities in it and generalize well. So a regularizer is introduced to quantify the complexity of a model, and many successful machine learning algorithms fall in the framework of regularized risk minimization:

$$\text{(How well the model fits the training data)} \quad (1)$$
$$+\lambda \cdot \text{(complexity/regularization of the model)}, \quad (2)$$

where the positive real number λ controls the tradeoff.

There is a variety of regularizers, which yield different statistical and computational properties. In general, there is no universally best regularizer, and a regularization approach must be chosen depending on the dataset.

Motivation and Background

The main goal of machine learning is to induce a model from the observed data, and use this model to make predictions and decisions. This is also largely the goal of general natural science, and is commonly called inverse problems ("forward problem" means using the model to generate observations). Therefore, it is no surprise that regularization had been well studied before the emergence of machine learning.

Inverse problems are typically ill-posed, e.g., having only a finite number of samples drawn from an uncountable space, or having a finite number of measurements in an infinite dimensional space. In machine learning, we often need to induce a classifier for the whole feature-label space, while only a finite number of feature-label pairs are available for training. In practice, the set of candidate models is often flexible enough to precisely fit all the training examples. However, this can lead to significant overfitting when the training data is noisy, and the real challenge is how to generalize well on the unseen data in the whole feature-label space.

Many techniques have been proposed to tackle ill-posed inverse problems. Almost all of them introduce an additional measure on how much a model is preferred *a priori* (i.e., without observing the training data). This extra belief on the desirable form of the model reflects the external knowledge of the model designer. It cannot be replaced by the data itself according to the "no free lunch theorem," which states that if there is no assumption on the mechanism of labeling, then it is impossible to generalize and any model can be inferior to another on some distribution of the feature-label pair (Devroye, Gyor, & Lugosi, 1996).

A commonly used prior is the so-called ►Occam's razor, which prefers "simple" models. It asserts that among all the models which fit the training data well, the simplest one is more likely to capture the "regularities" in it and hence has a larger chance to generalize well to the unseen data. Then an immediate question is how to quantify the complexity of a model, which is often called a regularizer. Intuitively, a regularizer can encode preference for a sparse model (few features are relevant for prediction), a large margin model (two classes have a wide margin), or a smooth model with weak high-frequency components. A general framework of regularization was given by Tikhonov (1943).

Theory

Suppose n feature-label pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ are drawn *iid* from a certain joint distribution P on $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the spaces of feature and label respectively. Let the marginal distribution on $\mathcal{X}$ and $\mathcal{Y}$ be P_x and P_y respectively. For convenience, let $\mathcal{X}$ be $\mathbb{R}^p$ (Euclidean space). Denote $X := (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and $\mathbf{y} := (y_1, \ldots, y_n)^\top$.

An Illustrative Example: Ridge Regression

Ridge regression is illustrative of the use of regularization. It tries to fit the label y by a linear model $\langle \mathbf{w}, \mathbf{x} \rangle$ (inner product). So we need to solve a system of linear equations in $\mathbf{w}$: $(\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top \mathbf{w} = \mathbf{y}$, which is equivalent to a linear least square problem: $\min_{\mathbf{w} \in \mathbb{R}^p} \|X^\top \mathbf{w} - \mathbf{y}\|^2$. If the rank of X is less than the dimension of $\mathbf{w}$, then it is overdetermined and the solution is not unique.

To approach this ill-posed problem, one needs to introduce additional assumptions on what models are preferred, i.e., the regularizer. One choice is to pick a matrix Γ and regularize $\mathbf{w}$ by $\|\Gamma \mathbf{w}\|^2$. As a result, we solve $\min_{\mathbf{w} \in \mathbb{R}^p} \|X^\top \mathbf{w} - \mathbf{y}\|^2 + \lambda \|\Gamma^\top \mathbf{w}\|^2$, and the solution has closed form $\mathbf{w}^* = (XX^\top + \lambda \Gamma \Gamma^\top) X \mathbf{y}$. Γ can be simply the identity matrix which encodes our preference for small norm models.

The use of regularization can also be justified from a Bayesian point of view. Treating $\mathbf{w}$ as a multi-variate random variable and the likelihood as $\exp\left(-\|X^\top \mathbf{w} - \mathbf{y}\|^2\right)$, then the minimizer of $\|X^\top \mathbf{w} - \mathbf{y}\|^2$

is just a maximum likelihood estimate of $\mathbf{w}$. However, we may also assume a prior distribution over $\mathbf{w}$, e.g., a Gaussian prior $p(\mathbf{w}) \sim \exp\left(-\lambda \left\|\Gamma^\top \mathbf{w}\right\|^2\right)$. Then the solution of the ridge regression is simply the maximum a posterior estimate of $\mathbf{w}$.

Examples of Regularization

A common approach to regularization is to penalize a model by its complexity measured by some real-valued function, e.g., a certain "norm" of $\mathbf{w}$. We list some examples below:

***L_1 regularization*:** L_1 regularizer, $\|\mathbf{w}\|_1 := \sum_i |w_i|$, is a popular approach to finding sparse models, i.e., only a few components of $\mathbf{w}$ are nonzero and only the corresponding small number of features are relevant to the prediction. A well-known example is the LASSO algorithm (Tibshirani, 1996), which uses a L_1 regularized least square:

$$\min_{\mathbf{w}\in\mathbb{R}^p} \left\|X^\top \mathbf{w} - \mathbf{y}\right\|^2 + \lambda \|\mathbf{w}\|_1 .$$

***L_2 regularization*:** The L_2 regularizer, $\frac{1}{2}||w||_2^2 := \frac{1}{2}\sum_i w_i^2$, is popular due to its self-dual properties. In all L_p spaces, only the L_2 space is Hilbertian and self-adjoint, so it affords much convenience in studying and exploiting the dual properties of the L_2 regularized models. A well-known example is the support vector machines (SVMs), which minimize the L_2 regularized hinge loss:

$$\frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - \langle \mathbf{w}, \mathbf{x}_i\rangle\} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 .$$

***L_p regularization*:** In general, all L_p norms $\|\mathbf{w}\|_p := \left(\sum_i |w_i|^p\right)^{1/p}$ ($p \geq 1$) can be used for regularization. When $p < 1$, $\left(\sum_i |w_i|^p\right)^{1/p}$ is no longer convex. A specially interesting case is when $p = 0$, and $\|\mathbf{w}\|_0$ is defined as the number of nonzero elements in $\mathbf{w}$ (the sparseness of $\mathbf{w}$). But explicitly optimizing the L_0 norm leads to a combinatorial problem which is hard to solve. In some cases, the L_1 regularizer can approximately recover the solution of L_0 regularization (Candes & Tao, 2005).

***$L_{p,q}$ regularizer*:** The $L_{p,q}$ regularizer is popular in the context of multi-task learning (Tropp, 2006). Suppose there are T tasks, and each training example $\mathbf{x}_i$ has a label vector $\mathbf{y}_i \in \mathbb{R}^T$ with each component corresponding to a task. For each task t, we seek for a linear regressor $\langle \mathbf{w}_t, \mathbf{x}\rangle$ such that for each training example $\mathbf{x}_i$, $\langle \mathbf{w}_t, \mathbf{x}_i\rangle$ fits the tth component of $\mathbf{y}_i$. Of course, the $\mathbf{w}_t$ could be determined independently from each other. But in many applications, the T tasks are somehow related, and it will be advantageous to learn them as a whole. Stack $\mathbf{w}_t$'s into a matrix $W := (\mathbf{w}_1, \ldots, \mathbf{w}_T)$ where each column corresponds to a task and each row corresponds to a feature. Then the intuition of multitask learning can be concretized by regularizing W with the $L_{p,q}$ compositional norm ($p, q \geq 1$):

$$\|W\|_{p,q} := \left(\sum_i \left(\sum_t |w_{it}|^p\right)^{\frac{q}{p}}\right)^{\frac{1}{q}},$$

where w_{it} is the ith component of $\mathbf{w}_t$. When $q = 1$, it becomes the L_1 norm of the L_p norm of the rows, and the sparse inducing property of L_1 norm encourages the rows to have L_p norm 0, i.e., the corresponding feature is not used by *any* task. Other choices of p and q are also possible.

***Entropy regularizer*:** The entropy regularizer is useful in boosting, and it works in a slightly different way from the above regularizers. Boosting aims to find a convex combination of hypotheses, such that the training data is accurately classified by the ensemble. At each step, the boosting algorithm maintains a distribution $\mathbf{d}$ ($d_i > 0$ and $\sum_i d_i = 1$) over the training examples, feeds $\mathbf{d}$ to an oracle which returns a new hypothesis, and then updates $\mathbf{d}$ and go on. As a "simple" ensemble means a small number of weak hypotheses, the boosting algorithm is expected to find an accurate ensemble by taking as few steps as possible. This can be achieved by exponentiated gradient descent (Kivinen & Warmuth, 1997), which stems from the relative entropy regularizer $\sum_i d_i \log \frac{d_i}{1/n}$ applied at each step. It also attracts $\mathbf{d}$ toward the uniform distribution, which helps avoid overfitting the noise, i.e., trying hard to match the (incorrect) label of a few training examples.

***Miscellaneous*:** Instead of using a function that directly measures the complexity of the model $\mathbf{w}$, regularization

can also be achieved by penalizing the complexity of the *output* of the model on the training data. This is called value regularization (Rifkin & Lippert, 2007). It not only yields neat derivations of standard algorithms, but also provides much convenience in studying the learning theory and optimization.

Furthermore, the regularized risk minimization framework in (1) is not the only approach to regularization. For example, in ▶online learning where the model is updated iteratively, early stopping is an effective form of regularization and it has been widely used in training neural networks. Suppose the available dataset is divided into a training set and a validation set, and the model is learned online from the training set. Then the algorithm terminates when the performance of the model on the validation set stops improving.

Measuring the Capacity of Model Class

Besides penalizing the complexity of the model, we can restrict the complexity of the model class $\mathcal{F}$ in the first place. For example, ▶linear regression is intrinsically "simpler" than quadratic regression. ▶Decision stumps are "simpler" than linear classifiers. In other words, regularization can be achieved by restricting the capacity of the model class, and the key question is how to quantify this capacity. Some commonly used measures in the context of binary classification are the following.

***VC dimension*:** The Vapnik–Chervonenkis dimension (▶VC dimension) quantifies how many data points can be arbitrarily labeled by using the functions in $\mathcal{F}$ (Vapnik & Chervonenkis, 1971). $\mathcal{F}$ is said to *shatter* a set of data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ if for any assignment of labels to these points, there exists a function $f \in \mathcal{F}$ which yields this labeling. The VC dimension of $\mathcal{F}$ is the maximum n such that there exist n data points that can be shattered by $\mathcal{F}$. For example, decision stumps have VC dimension 2, and linear classifiers (with bias) in a p-dimensional space have VC dimension $p + 1$.

***Covering number*:** The idea of covering number (Guo, Bartlett, Shawe-Taylor & Williamson 1999) is to characterize the inherent "dimension" of $\mathcal{F}$, in a way that follows the standard concept of vector dimension. Given n data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, we may endow the model class $\mathcal{F}$ with the following metric:

$$d_n(f,g) := \frac{1}{n}\sum_{i=1}^{n} \delta(f(\mathbf{x}_i) \neq g(\mathbf{x}_i)), \quad \forall f,g \in \mathcal{F},$$

where $\delta(\cdot) = 1$ if $\cdot$ is true and 0 otherwise. A set of functions $f_1, \ldots, f_m$ is said to be a cover of $\mathcal{F}$ at radius ϵ if for any function $f \in \mathcal{F}$, there exists an f_i such that $d_n(f, f_i) < \epsilon$. Then the covering number of $\mathcal{F}$ at radius $\epsilon > 0$ with respect to d_n is the minimum size of a cover of radius ϵ.

To understand the motivation of the definition, consider the unit ball in $\mathbb{R}^p$. To cover it by ϵ radius balls, one needs order $N(\epsilon, p) = \epsilon^{-p}$ balls. Then the dimension p can be estimated from the rate of growth of $\log N(\epsilon, p) = -p \log \epsilon$ with respect to ϵ. The covering number is an analogy of $N(\epsilon, p)$, and the dimension of $\mathcal{F}$ can be estimated in the same spirit.

***Rademacher average*:** The Rademacher average is a soft variant of the VC dimension. Instead of requiring the model class to shatter n data points, it allows that the labels be violated at some cost. Let $\sigma_i \in \{-1, 1\}$ be an arbitrary assignment of the labels, and assume all functions in $\mathcal{F}$ range in $\{-1, 1\}$ (this restriction can be relaxed). Then a model $f \in \mathcal{F}$ is considered as the most consistent with $\{\sigma_i\}$ if it maximizes $\frac{1}{n}\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)$. This term equals 1 if $\mathcal{F}$ does contain a model consistent with $\{\sigma_i\}$. Then we take an average over all possible assignments of σ_i, i.e., treating σ_i as a binary random variable with $P(\sigma_i = 1) = P(\sigma_i = -1) = 0.5$, and calculating the expectation of the best compatibility over $\{\sigma_i\}$:

$$\mathcal{R}_n(\mathcal{F}) = \mathop{\mathbb{E}}_{\sigma}\left[\sup_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)\right].$$

Rademacher complexity often leads to tighter generalization bounds than VC dimension thanks to its dependency on the observed data. Furthermore, we may take expectation over the samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ drawn independently and identically from P_x:

$$\mathcal{R}(\mathcal{F}) = \mathop{\mathbb{E}}_{\mathbf{x}_i \sim P_x} \mathop{\mathbb{E}}_{\sigma}\left[\sup_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)\right].$$

Therefore, similar to VC dimension, the Rademacher average is high if the model class $\mathcal{F}$ is "rich," and can match most assignments of $\{\sigma_i\}$.

Applications

In many applications such as bioinformatics, the training examples are expensive and the number of features p is much higher than the number of labeled

examples n. In such cases, regularization is crucial, (e.g., Zhang, Zhang, & Wells, 2008).

L_1 regularization has gained much popularity recently in the field of compressed sensing, and it has been widely used in imaging for radar, astronomy, medical diagnosis, and geophysics. See an ensemble of publications at http://dsp.rice.edu/cs

The main spirit of regularization, namely a preference for models with lower complexity, has been used by some ►model evaluation techniques. Examples include Akaike information criterion (AIC), Bayesian information criterion (BIC), ►minimum description length (MDL), and the ►minimum message length (MML).

Cross References

►Minimum Description Length
►Model Evaluation
►Occam's Razor
►Overfitting
►Statistical Learning Theory
►Support Vector Machines
►VC Dimension

Recommended Reading

Regularization lies at the heart of statistical machine learning, and it is indispensable in almost every learning algorithm. A comprehensive statistical analysis from the computational learning theory perspective can be found in Bousquet, Boucheron, & Lugosi (2005) and Vapnik (1998). Abundant resources on compressed sensing including both theory and applications are available at http://dsp.rice.edu/cs. Regularizations related to SVMs and kernel methods are discussed in detail by Schölkopf & Smola (2002) and Shawe-Taylor & Cristianini (2004). Anthony & Bartlett (1999) provide in-depth theoretical analysis for neural networks.

Anthony, M., & Bartlett, P. L. (1999). *Neural network learning: Theoretical foundations*. Cambridge: Cambridge University Press.

Bousquet, O., Boucheron, S., & Lugosi, G. (2005). Theory of classification: A survey of recent advances. *ESAIM: Probability and Statistics, 9*, 323–375.

Candes, E., & Tao, T. (2005). Decoding by linear programming. *IEEE Transactions on Information Theory, 51*(12), 4203–4215.

Devroye, L., Györ, L., & Lugosi, G. (1996). *A probabilistic theory of pattern recognition, vol. 31 of applications of mathematics*. New York: Springer.

Guo, Y., Bartlett, P. L., Shawe-Taylor, J., & Williamson, R. C. (1999). Covering numbers for support vector machines. In *Proceedings of the Annual Conference Computational Learning Theory*.

Kivinen, J., & Warmuth, M. K. (1997). Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation, 132*(1), 1–64.

Rifkin, R. M., & Lippert, R. A. (2007). Value regularization and Fenchel duality. *Journal of Machine Learning Research, 8*, 441–479.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels*. Cambridge: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2004). Kernel methods for pattern analysis. Cambridge: Cambridge University Press.

Tibshirani, R. (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society. Series B. Statistical Methodology, 58*, 267–288.

Tikhonov, A. N. (1943). On the stability of inverse problems. *Doklady Akademii nauk SSSR, 39*(5), 195–198.

Tropp, J. A. (2006). Algorithms for simultaneous sparse approximation, part ii: Convex relaxation. *Signal Processing, 86*(3), 589C–602.

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley: New York

Vapnik, V., & Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications, 16*(2), 264–281.

Zhang, M., Zhang, D., & Wells, M. T. (2008). Variable selection for large p small n regression models with incomplete data: Mapping QTL with epistases. *BMC Bioinformatics, 9*, 251.

Regularization Networks

►Radial Basis Function Networks

Reinforcement Learning

Peter Stone
The University of Texas at Austin, Austin, TX, USA

Reinforcement learning describes a large class of learning problems characteristic of autonomous agents interacting in an environment: sequential decision-making problems with delayed reward. Reinforcement learning algorithms seek to learn a policy (mapping from states to actions) that maximize the reward received over time.

Unlike in ►supervised learning problems, in reinforcement-learning problems, there are no labeled examples of correct and incorrect behavior. However, unlike ►unsupervised learning problems, a reward signal can be perceived.

Many different algorithms for solving reinforcement-learning problems are covered in other entries. This

entry provides just a brief high-level classification of the algorithms.

Perhaps the most well-known approach to solving reinforcement-learning problems, as covered in detail by Sutton and Barto (1998), is based on learning a value function, which represents the long-term expected reward of each state the agent may encounter, given a particular policy. This approach typically assumes that the environment is a ►Markov decision process in which rewards are discounted over time, though it is also possible to optimize for average reward per time step as in ►average-reward reinforcement learning. If a complete model of the environment is available, ►dynamic programming, or specifically ►value iteration, can be used to compute an optimal value function, from which an optimal policy can be derived.

If a model is not available, an optimal value function can be learned from experience via model-free techniques such as ►temporal difference learning, which combine elements of dynamic programming with Monte Carlo estimation. Partly due to Watkins' elegant proof that ►Q-learning converges to the optimal value function (Watkins, 1989), temporal difference methods are currently among the most commonly used approaches for reinforcement-learning problems.

Watkins' convergence proof relies on executing a policy that visits every state infinitely often. In practice, Q-learning does converge in small, discrete domains. However in larger, and particularly in continuous domains, the learning algorithm must generalize the value function across states, a process known as ►value function approximation. Examples include ►instance-based reinforcement learning, ►Gaussian process reinforcement learning, and ►relational reinforcement learning.

Even when combined with value function approximation, the most basic value-free methods, such as Q-learning and SARSA are very inefficient with respect to experience: they are not sample-efficient. With the view that experience is often more costly than computation, much research has been devoted to making more efficient use of experience, for instance via ►hierarchical reinforcement learning, ►reward shaping, or ►model-based reinforcement learning in which the experience is used to learn a domain model, which can then be solved via dynamic programming.

Though these methods make efficient use of the experience that is presented to them, the goal of optimizing sample efficiency also motivates the study of ►efficient exploration in reinforcement learning. The study of exploration methods can be isolated from the full reinforcement-learning problem by removing the notion of temporally delayed reward as is done in ►associative reinforcement learning or by removing the notion of states altogether as is done in ►k-armed bandits. k-Armed bandit algorithms focus entirely on the exploration versus exploitation challenge, without having to worry about generalization across states or delayed rewards. Back in the context of the full RL problem, ►Bayesian reinforcement learning enables optimal exploration given prior distributions over the parameters of the learning problem. However, its computational complexity has limited its use so far to very small domains.

Although most of the methods above revolve around learning a value function, reinforcement-learning problems can also be solved without learning value functions, by directly searching the space of potential policies via policy search. Effective ways of conducting such a search include ►policy gradient reinforcement learning, ►least squares reinforcement learning methods, and evolutionary reinforcement learning.

As typically formulated, the goal of a reinforcement-learning algorithm is to learn an optimal (or high-performing) policy based on knowledge of, or experience of, a reward function (and state transition function). However, it is also possible to take the opposite perspective that of trying to learn the reward function based on observation of the optimal policy. This problem formulation is known as ►inverse reinforcement learning.

Leveraging this large body of theory and algorithms, a current focus in the field is deploying large-scale, successful applications of reinforcement learning. Two such applications treated herein are ►autonomous helicopter flight using reinforcement learning and ►robot learning.

Cross References

►Associative Reinforcement Learning
►Autonomous Helicopter Flight Using Reinforcement Learning

►Average-Reward Reinforcement Learning
►Bayesian Reinforcement Learning
►Dynamic Programming
►Efficient Exploration in Reinforcement Learning
►Gaussian Process Reinforcement Learning
►Hierarchical Reinforcement Learning
►Instance-Based Reinforcement Learning
►Inverse Reinforcement Learning
►Least Squares Reinforcement Learning Methods
►Model-Based Reinforcement Learning
►Policy Gradient Methods
►Q-Learning
►Relational Reinforcement Learning
►Reward Shaping
►Symbolic Dynamic Programming
►Temporal Difference Learning
►Value Function Approximation

Recommended Reading

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, UK.

Reinforcement Learning in Structured Domains

►Relational Reinforcement Learning

Relational

The adjective *relational* can have two different meanings in machine learning. The ambiguity comes from an ambiguity in database terminology.

►Relational Data Mining refers to relational database, and is sometimes denoted *multi-relational* data mining. Indeed a relational database typically involves several relations (a relation is the formal name of a table). Those tables are often linked to each other, but the "relational" adjective does not refer to those relationships.

On the other hand, ►Relational Learning focuses on those relationships and intends to learn whether a relationship exists between some given entities.

Cross References

►Propositionalization
►Relational Data Mining
►Relational Learning

Relational Data Mining

►Inductive Logic Programming

Relational Dynamic Programming

►Symbolic Dynamic Programming

Relational Learning

Jan Struyf, Hendrik Blockeel
Katholieke Universiteit Leuven, Heverlee, Belgium

Problem Definition

Relational learning refers to learning in a context where there may be relationships between learning examples, or where these examples may have a complex internal structure (i.e., consist of multiple components and there may be relationships between these components). In other words, the "relational" may refer to both an internal or external relational structure describing the examples. In fact, there is no essential difference between these two cases, as it depends on the definition of an example whether relations are internal or external to it. Most methods, however, are clearly set in one of these two contexts.

Learning from Examples with External Relationships

This setting considers learning from a set of examples where each example itself has a relatively simple description, for instance in the attribute-value format, and relationships may be present among these examples.

Example 1. Consider the task of web-page classification. Each web-page is described by a fixed set of attributes, such as a bag of words representation of the page. Web-pages may be related through hyperlinks, and the class label of a given page typically depends on the labels of the pages to which it links.

Example 2. Consider the Internet Movie Database (www.imdb.com). Each movie is described by a fixed set of attributes, such as its title and genre. Movies are related to other entity types, such as *Studio*, *Director*, *Producer*, and *Actor*, each of which is in turn described by a different set of attributes. Note that two movies can be related through the other entity types. For example, they can be made by the same studio or star the same well-known actor. The learning task in this domain could be, for instance, predicting the opening weekend box office receipts of the movies.

If relationships are present among examples, then the examples may not be independent and identically distributed (i.i.d.), an assumption made by many learning algorithms. Relational data that violates this assumption can be detrimental to learning performance as Jensen and Neville (2002) show. Relationships among examples can, on the other hand, also be exploited by the learning algorithm. ►Collective classification techniques (Jensen, Neville, & Gallagher, 2004), for example, take the class labels of related examples into account when classifying a new instance.

Learning from Examples with a Complex Internal Structure

In this setting, each example may have a complex internal structure, but no relationships exist that relate different examples to one another. Learning algorithms typically use individual-centered representations in this setting, such as logical interpretations or strongly typed terms (Lloyd, 2003), which store together all data of a given example. An important advantage of individual-centered representations is that they scale better to large datasets. Special cases of this setting include applications where the examples can be represented as graphs, trees, or sequences.

Example 3. Consider a database of candidate chemical compounds to be used in drugs. The molecular structure of each compound can be represented as a graph where the vertices are atoms and the edges are bonds. Each atom is labeled with its element type and the bonds can be single, double, triple, or aromatic bonds. Compounds are classified as active or inactive with regard to a given disease and the goal is to build models that are able to distinguish active from inactive compounds based on their molecular structure. Such models can, for instance, be used to gain insight in the common substructures, such as binding sites, that determine a compound's activity.

Approaches to Relational Learning

Many different kinds of learning tasks have been defined in relational learning, and an even larger number of approaches have been proposed for tackling these tasks. We give an overview of different learning settings that can be considered instances of relational learning.

Inductive Logic Programming

In ►inductive logic programming (ILP), the input and output knowledge of a learner are described in (variants of) first-order predicate logic. Languages based on first-order logic are highly expressive from the point of view of knowledge representation, and indeed, a language such as Prolog (Bratko, 2000) can be used without adaptations to represent objects and the relationships between them, as well as background knowledge that one may have about the domain.

Example 4. This example is based on the work by Finn, Muggleton, Page, and Srinivasan (1998). Consider a data set that describes chemical compounds. The active compounds in the set are ACE inhibitors, which are used in treatments for hypertension. The molecular structure of the compounds is represented as a set of Prolog facts, such as: atom(m1, a1, o).

```
atom(m1, a2, c).
...
bond(m1, a1, a2, 1).
...
coord(m1, a1, 5.91, −2.44, 1.79).
coord(m1, a2, 0.57, −2.77, 0.33).
...
```

which states that molecule m1 includes an oxygen atom a1 and a carbon atom a2 that are single bonded. The coord/5 predicate lists the 3D coordinates of the

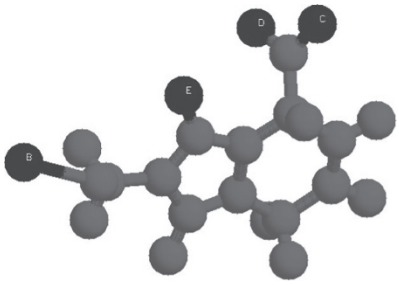
a

```
ACE_inhibitor(A) :-
     zincsite(A, B),
     hacc(A, C),
     dist(A, B, C, 7.9, 1.0),
     hacc(A, D),
     dist(A, B, D, 8.5, 1.0),
     dist(A, C, D, 2.1, 1.0),
     hacc(A, E),
     dist(A, B, E, 4.9, 1.0),
     dist(A, C, E, 3.1, 1.0),
     dist(A, D, E, 3.8, 1.0).
```

b

Molecule A is an ACE inhibitor if:
molecule A can bind to zinc at site B, and
molecule A contains a hydrogen acceptor C, and
the distance between B and C is 7.9 ± 1.0Å, and
molecule A contains a hydrogen acceptor D, and
the distance between B and D is 8.5 ± 1.0Å, and
the distance between C and D is 2.1 ± 1.0Å, and
molecule A contains a hydrogen acceptor E, and
the distance between B and E is 4.9 ± 1.0Å, and
the distance between C and D is 3.1 ± 1.0Å, and
the distance between D and E is 3.8 ± 1.0Å.

c



Relational Learning. Figure 1. (a) Prolog clause modeling the concept of an ACE inhibitor in terms of the background knowledge predicates zincsite/2, hacc/2, and dist/5. (b) The inductive logic programming system Progol automatically translates (a) into the "Sternberg English" rule, which can be easily read by human experts. (c) A molecule with the active site indicated by the atoms B, C, D, and E. (Image courtesy of Finn et al. (1998).)

atoms in the given conformer. Background knowledge, such as the concepts zinc site, hydrogen donor, and the distance between atoms, are defined by means of Prolog clauses. Figure 1 shows a clause learned by the inductive logic programming system Progol (Džeroski & Lavrač, 2001, Ch. 7) that makes use of these background knowledge predicates. This clause is the description of a pharmacophore, that is, a submolecular structure that causes a certain observable property of a molecule.

More details on the theory of inductive logic programming and descriptions of algorithms can be found in the entry on ▶inductive logic programming in this encyclopedia, or in references (De Raedt, 2008; Džeroski & Lavrač, 2001).

Learning from Graphs

A graph is a mathematical structure consisting of a set of nodes V and a set of edges $E \subseteq V^2$ between those nodes. The set of edges is by definition a binary relation defined over the nodes. Hence, for any learning problem where the relationships between examples can be described using a single binary relation, the training set can be represented straightforwardly as a graph. This setting covers a wide range of relational learning tasks, for example, web mining (the set of links between pages is a binary relation), social network analysis, etc. Non-binary relationships can be represented as hypergraphs; in a hypergraph, edges are defined as subsets of V of arbitrary size, rather than elements of V^2.

In graph-based learning systems, there is a clear distinction between approaches that learn from examples with external relationships, where the whole data set is represented as a single graph and each node is an example, and individual-centered approaches, where each example by itself is a graph. In the first kind of approaches, the goal is often to predict properties of existing nodes or edges, to predict the existence or non-existence of edges ("▶link discovery"), to predict whether two nodes actually refer to the same object ("node identification"), detection of subgraphs that frequently occur in the graph, etc. When learning from

multiple graphs, a typical goal is to learn a model for classifying the graphs, to find frequent substructures (where frequency is defined as the number of graphs a subgraphs occurs in), etc.

Compared to other methods for relational learning, graph-based methods typically focus more on the structure of the graph, and less on properties of single nodes. They may take node and edge labels into account, but typically do not allow for more elaborate information to be associated with each node.

►Graph mining methods are often more efficient than other relational mining methods because they avoid certain kinds of overhead, but are typically still NP-complete, as they generally rely on subgraph isomorphism testing. Nevertheless, researchers have been able to significantly improve efficiency or even avoid NP-completeness by looking only for linear or tree-shaped patterns, or by restricting the graphs analyzed to a relatively broad subclass. As an example, Horváth et al. (2006) show that a large majority of molecules belong to the class of outerplanar graphs, and propose an efficient algorithm for subgraph isomorphism testing in this class.

More information about mining graph data can be found in the ►graph mining entry in this encyclopedia, or in (Cook & Holder, 2007; Washio & Motoda, 2003).

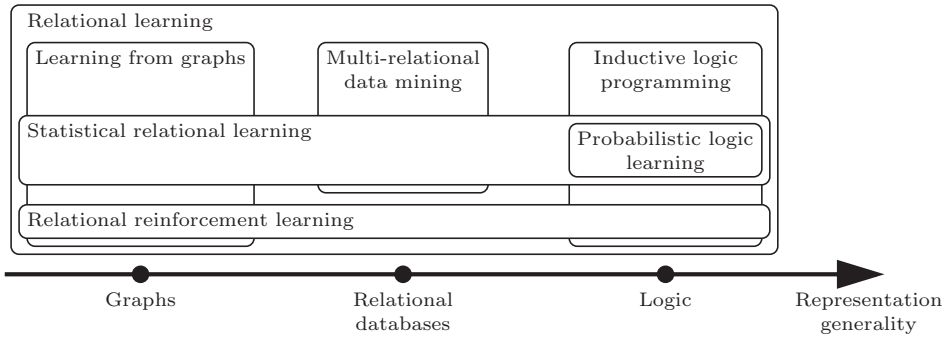
Multi-relational Data Mining

Multi-relational data mining approaches relational learning from the relational database point of view. The term "multi-relational" refers to the fact that from the database perspective, one learns from information spread over multiple tables or relations, as opposed to ►attribute-value learning, where one learns from a single table.

Multi-relational data mining systems tightly integrate with relational databases. Mainly rule and decision tree learners have been developed in this setting. Because practical relational databases may be huge, most of these systems pay much attention to efficiency and scalability, and use techniques such as sampling and pre-computation (e.g., materializing views). An example of a scalable and efficient multi-relational rule learning system is CrossMine (Yin, Han, Yang, & Yu, 2006).

An alternative approach to relational learning and multi-relational data mining is ►propositionalization. Propositionalization consists of automatically converting the relational representation into an attribute-value representation and then using attribute-value data mining algorithms on the resulting representation. An important line of research within multi-relational data mining investigates how database approaches can be used to this end. Database oriented propositionalization creates a view in which each example is represented by precisely one row. Information from related entities is incorporated into this row by adding derived attributes, computed by means of aggregation. In the movie database (Example 2), the view representing movies could include aggregated attributes such as the number of actors starring in the movie. A comparison of propositionalization approaches is presented by Krogel et al. (2003), and a discussion of them is also included in this volume.

Finally, note that most inductive logic programming systems are directly applicable to multi-relational data mining by representing each relational table as a predicate. This is possible because the relational representation is essentially a subset of first-order logic (known as datalog). Much research on multi-relational



data mining was developed within the ILP community (Džeroski & Lavrač, 2001).

Statistical Relational Learning/Probabilistic Logic Learning

Research on relational learning, especially in the beginning, has largely focused on how to handle the relational structure of the data, and ignored aspects such as uncertainty. Indeed, the databases handled in multi-relational data mining, or the knowledge assumed given in inductive logic programming, are typically assumed to be deterministic. With the rise of probabilistic representations and algorithms within machine learning has come an increased interest in enabling relational learners to cope with uncertainty in the input data. This goal has been approached from at least two different directions: statistical learning approaches have been extended toward the relational setting, giving rise to the area of ►statistical relational learning, whereas inductive logic programming researchers have investigated how to extend their knowledge representation and learning algorithms to cater for probabilistic information, referring to this research area as ►probabilistic logic learning. While there are some differences in terminology and approaches, both research areas essentially address the same research question, namely how to integrate relational and probabilistic learning.

Among the best known approaches for statistical relational learning is the learning of probabilistic relational models (PRMs, Džeroski & Lavrač, 2001, Chap. 13). PRMs extend Bayesian networks to the relational representation used in relational databases. PRMs model the joint probability distribution over the non-key attributes in a relational database schema. Similar to Bayesian networks, PRMs are ►graphical models. Each attribute corresponds to a node and direct dependencies are modeled by directed edges. Such edges can connect attributes from different entity types that are (indirectly) related (such a relationship is called a "slot chain"). Inference in PRMs occurs by constructing a ►Bayesian network by instantiating the PRM with the data in the database and performing the inference in the latter. To handle 1:N relationships in the Bayesian network, PRMs make use of aggregation, similar to the propositionalization techniques mentioned above.

Bayesian logic programs (BLPs) (Kersting, 2006) aim at combining the inference power of Bayesian networks with that of first-order logic reasoning. Similar to PRMs, the semantics of a BLP is defined by translating it to a Bayesian network. Using this network, the probability of a given interpretation or the probability that a given query yields a particular answer can be computed.

The acyclicity requirement of Bayesian networks carries over to representations such as PRMs and BLPs. Markov logic networks (MLNs) (Richardson & Domingos, 2006) upgrade ►Markov networks to first-order logic and allow networks with cycles. MLNs are defined as sets of weighted first-order logic formulas. These are viewed as "soft" constraints on logical interpretations: the fewer formulas a given interpretation violates, the higher its probability. The weight determines the contribution of a given formula: the higher its weight, the greater the difference in log probability between an interpretation that satisfies the formula and one that does not, other things being equal. The Alchemy system implements structure and parameter learning for MLNs.

More specific statistical learning techniques such as Naïve Bayes and Hidden Markov Models have also been upgraded to the relational setting. More information about such algorithms and about statistical relational learning in general can be found in (Getoor & Taskar, 2007; Kersting, 2006).

In probabilistic logic learning, two types of semantics are distinguished (De Raedt & Kersting, 2003): the model theoretic semantics and the proof theoretic semantics. Approaches that are based on the model theoretic semantics define a probability distribution over interpretations and extend probabilistic attribute-value techniques, such as Bayesian networks and Markov networks, while proof theoretic semantics approaches define a probability distribution over proofs and upgrade, e.g., stochastic context free grammars.

Example 5. Consider the case where each example is a sentence in natural language. In this example, a model theoretic approach would define a probability distribution directly over sentences. A proof theoretic approach would define a probability distribution over "proofs," in this case possible parse trees of the sentence (each sentence may have several possible parses). Note that the proof theoretic view is more general in the sense that

the distribution over sentences can be computed from the distribution over proofs.

Stochastic logic programs (SLPs) (Muggleton, 1996) follow most closely the proof theoretic view and upgrade stochastic context free grammars to first-order logic. SLPs are logic programs with probabilities attached to the clauses such that the probabilities of clauses with the same head sum to 1.0. The probability of a proof is then computed as the product of the probabilities of the clauses that are used in the proof. PRISM (Sato & Kameya, 1997) follows a related approach where the probabilities are defined on ground facts.

Like with standard graphical models, learning algorithms may include both parameter learning (estimating the probabilities) and structure learning (learning the program). For most frameworks mentioned above, such techniques have been or are being developed.

For a more detailed treatment of statistical relational learning and probabilistic logic learning, we refer to the entry on statistical relational learning in this volume, and to several reference works (De Raedt & Kersting, 2003; Getoor & Taskar, 2007; Kersting, 2006; De Raedt, Frasconi, Kersting, & Muggleton, 2008).

Relational Reinforcement Learning

Relational reinforcement learning (RRL) (Džeroski, De Raedt, & Driessens, 2001; Tadepalli, Givan, & Driessens, 2004) is reinforcement learning upgraded to the relational setting. Reinforcement learning is concerned with how an agent should act in a given environment to maximize its accumulated reward. In RRL, both the state of the environment and the actions are represented using a relational representation, typically in the form of a logic program.

Much research in RRL focuses on Q-learning, which represents the knowledge of the agent by means of a Q-function mapping state–action pairs to real values. During exploration, the agent selects in each state the action that is ranked highest by the Q-function. The Q-function is typically represented using a relational regression technique. Several techniques, such as relational regression trees, relational instance based learning, and relational kernel based regression have been considered in this context. Note that the regression algorithms must be able to learn incrementally: each time the agent receives a new reward, the Q-function must be incrementally updated for the episode (sequence of state-action pairs) that led to the reward. Due to the use of relational regression techniques, the agent is able to generalize over states: it will perform similar actions in similar states and therefore scales better to large application domains.

More recent topics in RRL include how expert knowledge can be provided to the agent in the form of guidance, and how learned knowledge can be transferred to related domains ("transfer learning"). More details on these techniques and more specific information on the topic of relational reinforcement learning can be found in its corresponding encyclopedia entry and in the related entry on ▶symbolic dynamic programming, as well as in references (Džeroski et al., 2001; Tadepalli et al., 2004).

Cross References

▶Inductive Logic Programming
▶Multi-Relational Data Mining
▶Relational Reinforcement Learning

Recommended Reading

Most of the topics covered in this entry have more detailed entries in this encyclopedia, namely "Inductive Logic Programming," "Graph Mining," "Relational Data Mining," and "Relational Reinforcement Learning." These entries provide a brief introduction to these more specific topics and appropriate references for further reading.

Direct relevant references to the literature include the following. A comprehensive introduction to ILP can be found in De Raedt's book (De Raedt, 2008) on logical and relational learning, or in the collection edited by Džeroski and Lavrač (2001) on relational data mining. Learning from graphs is covered by Cook and Holder (2007). Džeroski and Lavrač (2001) is also a good starting point for reading about multi-relational data mining, together with research papers on multi-relational data mining systems, for instance, Yin et al. (2006), who present a detailed description of the CrossMine system. Statistical relational learning in general is covered in the collection edited by Getoor & Taskar (2007), while De Raedt & Kersting (2003) and De Raedt et al. (2008) present overviews of approaches originating in logic-based learning. An overview of relational reinforcement learning can be found in Tadepalli et al. (2004).

Bratko, I. (2000). *Prolog programming for artificial intelligence*. Reading, MA: Addison-Wesley (3rd ed.).

Cook, D. J., & Holder, L. B. (2007). *Mining graph data*. Hoboken, NJ: Wiley.

De Raedt, L. (2008). *Logical and relational learning*. Berlin: Springer.

De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (2008). *Probabilistic inductive logic programming*. Berlin: Springer.

De Raedt, L., & Kersting, K. (2003). Probabilistic logic learning. *SIGKDD Explorations, 5*(1), 31–48.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning, 43*, 7–52.
Džeroski, S., & Lavrač, N., (Eds.). (2001). *Relational data mining*. Berlin: Springer.
Finn, P., Muggleton, S., Page, D., & Srinivasan, A. (1998). Pharmacophore discovery using the inductive logic programming system PROGOL. *Machine Learning, 30*, 241–270.
Getoor, L., & Taskar, B. (2007). *Introduction to statistical relational learning*. Cambridge: MIT Press.
Horváth, T., Ramon, J., & Wrobel, S. (2006). Frequent subgraph mining in outerplanar graphs. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 197–206). New York: ACM.
Jensen, D., & Neville, J. (2002). Linkage and autocorrelation cause feature selection bias in relational learning. In *Proceeding of the 19th International Conference on Machine Learning*, University of New South Wales, Sydney (pp. 259–266). San Francisco, CA: Morgan Kaufmann.
Jensen, D., Neville, J., & Gallagher, B. (2004). Why collective inference improves relational classification. In *Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining*, Philadelphia, PA (pp. 593–598). New York: ACM.
Kersting, K. (2006). *An inductive logic programming approach to statistical relational learning*. Amsterdam: IOS Press.
Krogel, M.-A., Rawles, S., Železný, F., Flach, P., Lavrač, N., & Wrobel, S. (2003). Comparative evaluation of approaches to propositionalization. In *Proceedings of the 13th international conference on inductive logic programming*, Szeged, Hungary (pp. 194–217). Berlin: Springer-Verlag.
Lloyd, J. W. (2003). *Logic for learning*. Berlin: Springer.
Muggleton, S. (1996). Stochastic logic programs. In L. De Raedt (Ed.), *Advances in inductive logic programming* (pp. 254–264). Amsterdam: IOS Press.
Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning, 62*(1–2), 107–136.
Sato, T., & Kameya, Y. (1997). PRISM: A symbolic-statistical modeling language. In *Proceedings of the 15th International joint conference on artificial intelligence (IJCAI 97)*, Nagoya, Japan (pp. 1330–1335). San Francisco, CA: Morgan Kaufmann.
Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational reinforcement learning: An overview. In *Proceeding of the ICML'04 Workshop on relational reinforcement learning*, Banff, Canada (pp. 1–9).
Washio, T., & Motoda, H. (2003). State of the art of graph-based data mining. *SIGKDD Explorations, 5*(1), 59–68.
Yin, X., Han, J., Yang, J., & Yu, P. S. (2006). Efficient classification across multiple database relations: A CrossMine approach. *IEEE Transactions on Knowledge and Data Engineering, 18*(6), 770–783.

Relational Regression Tree

►First-Order Regression Tree

Relational Reinforcement Learning

Kurt Driessens
Universiteit Leuven, Celestijnenlaan, Belgium

Synonyms

Learning in worlds with objects; Reinforcement learning in structured domains

Definition

Relational reinforcement learning is concerned with learning behavior or control policies based on a numerical feedback signal, much like standard reinforcement learning, in complex domains where states (and actions) are largely characterized by the presence of objects, their properties, and the existing relations between those objects. Relational reinforcement learning uses approaches similar to those used for standard reinforcement learning, but extends these with methods that can abstract over specific object identities and exploit the structural information available in the environment.

Motivation and Background

►Reinforcement learning is a very attractive machine learning framework, as it tackles – in a sense – the whole artificial intelligence problem at a small scale: an agent acts in an unknown environment and has to learn how to behave optimally by reinforcement, i.e., through rewards and punishment. Reinforcement learning has produced some impressive and promising results. However, the applicability of reinforcement learning has been greatly limited by its problem of dealing with large problem spaces and its inability to generalize the learned knowledge to new but related problem domains.

While standard reinforcement learning methods represent the learning environment as a set of unrelated states or, when using ►attribute-value representations, as a vector space consisting of a fixed number of independent dimensions, humans tend to think about their environment in terms of objects, their properties, and the relations between them. Examples of objects in everyday life are chairs, people, streets, trees, etc. This representation allows people to treat or use most of

the new objects that they encounter correctly, without requiring training time to learn how to use them. For example, people are able to drink their coffee from any cup that will hold it, even if they have never encountered that specific cup before, because they already have experience with drinking their coffee from other cup-type objects. Standard reinforcement learning agents do not have this ability. Their state and action representations do not allow them to abstract away from specific object-identities and recognize them as a type of object they are already accustomed to.

Relational reinforcement learning tries to overcome this problem by representing states of the learning agent's environment as sets of objects, their properties, and the relationships between them, similar to the approaches used in ►relational learning and ►inductive logic programming. These structural representations make it possible for the relational reinforcement learning agent to abstract away from specific identities of objects and often also from the amount of objects present, the exact learning environment, or even the specific task to be performed.

The term "Relational reinforcement learning" was introduced by Džeroski, De Raedt, and Blockeel (1998) when they first teamed the Q-learning algorithm with a first-order regression algorithm. Since then, relational reinforcement learning has gained an increasing amount of interest.

Structure of the Learning System

In principle, the structure of a relational reinforcement learning system is very similar to that of standard reinforcement learning systems Fig. 3. At a high level, the learning agent interacts with an environment by performing actions that influence that environment, and the environment provides the learning agent with a description of its current state and a numerical feedback of the behavior of the agent. The goal of the agent is to maximize some cumulative form of this feedback signal. The major difference between standard reinforcement learning and relational reinforcement learning is the representation of the state–action–space. Relational reinforcement learning works on ►Markov decision processes where states and actions have been relationally factored, so-called relational Markov decision processes (RMDPs).

An RMDP can be defined as follows:

Definition 1 (Relational Markov Decision Process) *Let P_S be a set of state related predicates, P_A a set of action related predicates and C a set of constants in a logic Λ. Let $\mathcal{B}$ be a theory defined in that logic.*
An RMDP

is defined as $< S, A, T, R >$, where $S \equiv \{s \subset H^{P_S \cup C} | s \models \mathcal{B}\}$ represents the set of states, $A \equiv \{a \subset H^{P_A \cup C} | a \models \mathcal{B}\}$ represents the set of actions, in which H^X is the set of facts that can be constructed given the symbols in X, and T and R represent the transition probabilities and reward function respectively: $T : S \times A \times S \to [0,1]$ and $R : S \to \mathbb{R}$.

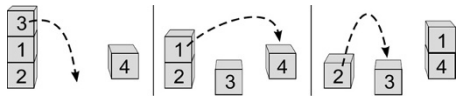
In less formal language, this means that the states and actions in an RMDP are represented using a set of constants C and a set of predicates P_S and P_A respectively and constrained by a background theory $\mathcal{B}$. This means that the background theory $\mathcal{B}$ defines which states are possible in the domain and which actions can be executed in which states.

The following example illustrates these concepts. Consider the blocks world depicted in Fig. 1. To represent this environment in first-order logic, one could use:

- State related predicates: P_S = {on/2, clear/1}
- Action related predicate: P_A = {move/2}
- Constants: C = {1,2,3,4,floor}

The set of facts $H^{P_S \cup C}$ would then include, for example: $on(1,2)$, $on(4,floor)$ and $clear(2)$ but also $on(3,3)$ and $on(floor,2)$. To constrain the possible states to those that actually make sense in a standard, i.e., real-world view of the blocks world, the theory $\mathcal{B}$ can include rules to make states that include these kinds of facts impossible. For example, to make sure that a block cannot be on top of itself, $\mathcal{B}$ could include the following rule:

$$\texttt{false} \leftarrow on(X,X).$$



Relational Reinforcement Learning. Figure 1. Example state–action pairs in the Blocks World

One can also include more extensive rules to define the exact physics of the blocks world that one is interested in. For example, including

$$\texttt{false} \leftarrow on(Y,X), on(Z,X), X \neq floor, Y \neq Z$$

as part of the theory $\mathcal{B}$, one can exclude states where two blocks are on top of the same block. The action space given by $H^{P_A \cup C}$ consists of facts such as $move(3,2)$ and $move(floor,1)$ and can be constrained by rules such as:

$$\texttt{false} \leftarrow move(floor,X).$$

which makes sure that the floor cannot be placed on top of a block.

The leftmost state–action pair of Fig. 1 can be fully specified by the following set of facts (state description on the left, action on the right):

on(2,floor).	clear(3).
on(1,2).	clear(4).
on(3,1).	clear(floor).
on(4,floor).	

One can easily generalize over specific states and create abstract states (or state–action pairs) that represent sets of states (or state–action pairs) by using variables instead of constants and by listing only those parts of states and actions that hold for each element of the abstract state (or state–action pair). For example, the abstract state "$on(1,2), on(2,floor)$" represents all states in which block 1 is on top of block 2, which in turn is on the floor. The abstract state does not specify the locations of any other blocks. Of the three states depicted in Fig. 1, the set of states represented by the abstract state would include the left and middle states. Abstract states can also be represented by using variables when one does not want to specify the location of any specific block, but wants to focus on structural aspects of the states and actions. The abstract state–action pair "$move(X,Y), on(Y,floor)$" represents all state–action pairs where a block is moved on top of another block that is on the floor, for example the middle and right state–action pairs of Fig. 1.

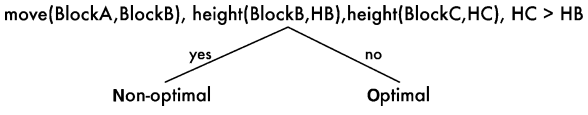
Added Benefits of Relational Reinforcement Learning

We already stated that the real world is made up out of interacting objects, or at least that humans often think about the real world as such. Relational reinforcement learning allows this same representation to be used by reinforcement learning agents, which in turn leads to more human-interpretable learning results.

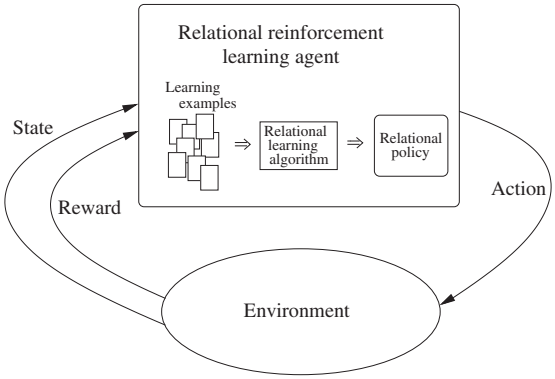
As a consequence of the used logical or relational representation of states and actions, the results learned by a relational reinforcement learning agent can be reused more easily when some of the parameters of the learning task change. Because relational reinforcement learning algorithms try to solve the problem at hand at an abstract level, the solutions will often carry over to different instantiations of that abstract problem. For example, the resulting policies learned by the RRL system (Driessens, 2004) discussed below, a very simple example of which is shown in Fig. 2, often generalize over domains with a varying number of objects. If only actions which lead to the "*optimal*" leaf are executed, the shown policy tree will organize any number of blocks into a single stack.

As another example of this, the relational approximate policy iteration approach (Fern, Yoon, & Givan, 2006), also discussed below, is able to learn task specific control knowledge from random walks in the environment. By treating the resulting state of such a random walk as a goal state and generalizing over the specifics of that goal (and the rest of the random walk) relational approximate policy iteration can learn domain specific, but goal independent policies. This generalization of the policy is accomplished by parametrization of the goal and focusing on the relations between objects in the goal, states and actions when representing the learned policy.

Another practical benefit of relational reinforcement learning lies in the field of inductive transfer. Transfer learning is concerned with the added benefits of having experience with a related task when being confronted with a new one. Because of the structural representation of learned results, the transfer of knowledge learned by relational reinforcement learning agents can be accomplished by recycling those parts



Relational Reinforcement Learning. Figure 2. Simple relational policy for stacking any number of blocks



Relational Reinforcement Learning. Figure 3. Structure of the RRL system

of the results that still hold valid information for the new task. Depending on the relation between the two tasks, this can yield substantial benefits concerning the required training experience.

The use of first-order logic as a representational language in relational reinforcement learning also allows the integration of reasoning methods with traditional reinforcement learning approaches. One example of this is ►Symbolic Dynamic Programming, which uses logical regression to compute necessary preconditions that allow an agent to reach certain goals. This same integration allows the use of search or planning knowledge as background information to extend the normal description of states and actions.

Example Relational Reinforcement Learning Approaches

Relational Q-Learning Relational reinforcement learning was introduced with the development of the RRL-system (Džeroski et al., 1998). This is a Q-learning system that employs a relational regression algorithm to generalize the Q-table used by standard Q-learning algorithms into a Q-function. The differences with a standard Q-learning agent are mostly located inside the learning agent. One important difference is the agent's representation of the current state. In relational reinforcement learning, this representation contains structural or relational information about the environment.

Inside the learning agent, the information consisting of encountered states, chosen actions, and the connected rewards is translated into learning examples. These examples are then processed by a relational learning system that produces a relational Q-function and/or policy as a result. The relational representation of the Q-function allows the RRL-system to use the structural properties of states and actions when assigning a Q-value to them.

Several relational regression approaches have been developed and applied in this context. While the original approach used an of-the-shelve relational regression algorithm that processed the learning examples in batch and had to be restarted to be able to process newly available learning experiences, a number of incremental algorithms have been developed for use in relational reinforcement learning since then. These include an incremental first-order regression tree algorithm, incremental relational instance based regression, kernel based regression that uses Gaussian processes, and graph-kernels and algorithms that include combinations of the above (Driessens, 2004).

It is possible to translate the learned Q-function approximations into a function that directly represents its policy. Using the values predicted by the learned Q-function, one can generate learning examples that represent state–action pairs and label them as either part of the learned policy or not. This results in a binary classification problem that can be handled by a supervised relational learning algorithms. This technique is known as P-learning (Džeroski, De Raedt, & Driessens, 2001). It exhibits better generalization performance across related learning problems than the Q-learning approach described above. Other than the first-order decision trees mentioned above, rule-based learners have also been applied to this kind of policy learning.

Non-parametric Policy Gradients Non-parametric policy gradients (Kersting & Driessens, 2008), also a model-free approach, apply Friedmann's gradient boosting (Friedman, 2001) in an otherwise standard policy gradient approach for reinforcement learning. To avoid having to represent policies using a fixed number of parameters, policies are represented as a weighted sum of regression models grown in a stage-wise optimization (This allows the number of parameters to grow as the experience of the learner increases, hence the name non-parametric.). While this does not make

non-parametric policy gradients a technique specifically designed for relational reinforcement learning, it allows, like the relational Q-learning approach described above, the use of relational regression models and is not constrained to the attribute-value setting of standard policy gradients.

The idea behind the approach is that instead of finding a single, highly accurate policy, it is easier to find many rough rules of thumb of how to change the way the agent currently acts. The learned policy is represented as

$$\pi(s,a) = \frac{e^{\Psi(s,a)}}{\sum_b e^{\Psi(s,b)}},$$

where instead of assuming a linear parameterization for Ψ as is done in standard policy gradients, it is assumed that Ψ will be represented by a linear combination of functions. Specifically, one starts with some initial function Ψ_0, e.g., based on the zero potential, and iteratively adds corrections $\Psi_m = \Psi_0 + \Delta_1 + \cdots + \Delta_m$. In contrast to the standard gradient approach, Δ_m here denotes the so-called functional gradient, which is sampled during interaction with the environment and then generalized by an off-the-shelf regression algorithm.

The advantages of policy gradients over value-function techniques are that they can learn non-deterministic policies and that convergence of the learning process can be guaranteed, even when using function approximation (Sutton, McAllester, Singh, & Mansour, 2000). Experimental results show that non-parametric policy gradients have the potential to significantly outperform relational Q-learning (Kersting & Driessens, 2008).

Relational Approximate Policy Iteration A different approach, which also directly learns a policy, is taken in relational approximate policy iteration (Fern et al., 2006). Like standard policy iteration, the approach iteratively improves its policy through interleaving evaluation and improvement steps. In contrast to standard policy iteration, it uses a policy language bias and a generalizing policy function.

Instead of building a value-function approximation for each policy evaluation step, relational approximate policy iteration evaluates the current policy and its closely related neighbors by sampling the state–action–space through a technique called policy roll-out. This technique generates a set of trajectories from a given state, by executing every possible action in that state and following the current policy for a number of steps afterward (It is also possible to improve convergence speed by following the next policy.). These trajectories and their associated costs result in number of learning examples – one for each possible action in each selected state – that can be used, together with the policy language bias to generate the next, improved policy.

Because every possible action in each sampled state needs to be evaluated, this approach does require a model or a resettable simulator of the environment. However, relational approximate policy iteration has been shown to work well for learning domain specific control knowledge and performs very well on planning competition problems.

Symbolic Dynamic Programming In contrast to the previous techniques, ▶symbolic dynamic programming (SDP) does not learn a policy through exploration of the environment. Instead, it is a model-based approach that uses knowledge about preconditions and consequences of actions to compute the fastest way to reach a given goal. Like other dynamic programming techniques, SDP starts from the goal the agent wants to reach and reasons backwards to find the policy that is needed to reach that goal. In contrast to other dynamic programming techniques, it does not solve specific instantiations of the problem domain, but instead solves the problem at an abstract level, thereby solving it for all possible instantiations of the problem at once.

SDP treats the required goal-conditions as an abstract state definition. Because pre- and post-conditions of actions are known, SDP can compute the necessary conditions that allow actions to reach the abstract goal state. These conditions define abstract states from which it is possible to reach a goal state in one step. Starting from these abstract states, the same approach can be used to discover abstract states that allow the goal to be reached in two steps and so on.

This approach was first proposed by Boutilier, Reiter, and Price (2001), implemented as a working system by Kersting, van Otterlo, and De Raedt (2004) and later improved upon by Sanner and Boutilier (2005). This last approach won 2nd place in the probabilistic programming competition at ICAPS in 2006.

Cross References

►Hierarchical Reinforcement Learning
►Inductive Logic Programming
►Model-Based Reinforcement Learning
►Policy Iteration
►Q-learning
►Reinforcement Learning
►Relational Learning
►Symbolic Dynamic Programming
►Temporal Difference Learning

Further Information

The field of relational reinforcement learning has given rise to a number of PhD dissertations in the last few years (Croonenborghs, 2009; Driessens, 2004; Sanner, 2008; van Otterlo, 2008). The dissertation of Martijn van Otterlo resulted in a book (van Otterlo, 2009) which provides a recent and reasonably complete overview of the relational reinforcement learning research field. Other publications that presents an overview of relational reinforcement learning research include the proceedings of the two workshops on representational issues in (relational) reinforcement learning at the International Conferences of Machine Learning in 2004 and 2005 (Driessens, Fern, & van Otterlo, 2005; Tadepalli, Givan, & Driessens, 2004).

Recommended Reading

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proceedings of the 17th international joint conference on artificial intelligence (IJCAI-2001)*, Seattle, WA (pp. 690–700).

Croonenborghs, T. (2009). *Model-assisted approaches for relational reinforcement learning*. PHD thesis, Department of Compute Science, Katholieke Universiteit Leuven.

Driessens, K. (2004). *Relational reinforcement learning*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven.

Driessens, K., Fern, A., & van Otterlo, M. (Eds.). (2005). *Proceedings of ICML-2005 workshop on rich representation for reinforcement learning*, Bonn, Germany.

Džeroski, S., De Raedt, L., & Blockeel, H. (1998). Relational reinforcement learning. In *Proceedings of the 15th international conference on machine learning (ICML-1998)* (pp. 136–143). San Francisco, CA: Morgan Kaufmann. Madison, WI, USA.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning, 43*, 7–52.

Fern, A., Yoon, S., & Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research, 25*, 85–118.

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics, 29*, 1189–1232.

Kersting, K., & Driessens, K. (2008). Non-parametric policy gradients: A unified treatment of propositional and relational domains. In A. McAllum & S. Roweis (Eds.), *Proceedings of the 25th international conference on machine learning (ICML 2008)*, Helsinki, Finland (pp. 456–463).

Kersting, K., van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. In *Proceedings of the twenty-first international conference on machine learning (ICML-2004)*, Banff, Canada (pp. 465–472).

Sanner, S. (2008). *First-order decision-theoretic planning in structured relational environments*. PhD thesis, Department of Compute Science, University of Toronto.

Sanner, S., & Boutilier, C. (2005). Approximate linear programming for first-order MDPs. In *Proceedings of the 21st conference on Uncertainty in AI (UAI)*, Edinburgh, Scotland.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems 12* (pp. 1057–1063). Cambridge: MIT Press.

Tadepalli, P., Givan, R., & Driessens, K. (Eds.). (2004). *Proceedings of the ICML-2004 workshop on relational reinforcement learning*, Banff, Canada.

van Otterlo, M. (2008). *The logic of adaptive learning*. PhD thesis, Centre for Telematics and Information Technology, University of Twente.

van Otterlo, M. (2009). The logic of adaptive behavior: Knowledge representation and algorithms for adaptive sequential decision making under uncertainty in first-order and relational domains. Amsterdam, The Netherlands: IOS Press.

Relational Value Iteration

►Symbolic Dynamic Programming

Relationship Extraction

►Link Prediction

Relevance Feedback

Relevance feedback provides a measure of the extent to which the results of a search match the expectations of the user who initiated the query. Explicit feedback require users to assess relevance by choosing one out of a number of choices, or to rank documents to reflect their perceived degree of relevance. Implicit feedback is obtained by monitoring user's behavior such as time spent browsing a document, amount of scrolling performed while browsing a document, number of times

a document is visited, etc. Relevance feedback is one the techniques used to support query reformulation and turn the search into an iterative and interactive process.

Cross References

►Search Engines: Applications of ML

Representation Language

►Hypothesis Language

Reservoir Computing

Risto Miikkulainen
The University of Texas at Austin, Austin, TX, USA

Synonyms

Echo state network; Liquid state machine

Definition

Reservoir computing is an approach to sequential processing where recurrency is separated from the output mapping (Jaeger, 2003; Maass, Natschlaeger, & Markram, 2002). The input sequence activates neurons in a recurrent neural network (a reservoir, where activity propagates as in a liquid). The recurrent network is large, nonlinear, randomly connected, and fixed. A linear output network receives activation from the recurrent network and generates the output of the entire machine. The idea is that if the recurrent network is large and complex enough, the desired outputs can likely be learned as linear transformations of its activation. Moreover, because the output transformation is linear, it is fast to train. Reservoir computing has been successful in particular in speech and language processing and vision and cognitive neuroscience.

Recommended Reading

Jaeger, H. (2003). Adaptive nonlinear system identification with echo state networks. In S. Becker, S. Thrun, & K. Obermayer (Eds.), *Advances in neural information processing systems* (Vol. 15, pp. 593–600). Cambridge, MA: MIT Press.

Maass, W., Natschlaeger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*, 2531–2560.

Resolution

►First-Order Logic

Resubstitution Estimate

Resubstitution estimates are estimates that are derived by applying a ►model to the ►training data from which it was learned. For example, *resubstitution error* is the error of a model on the training data.

Cross References

►Model Evaluation

Reward

In most *Markov decision process* applications, the decision-maker receives a *reward* each period. This reward can depend on the current *state*, the *action* taken, and the next state and is denoted by $r_t(s, a, s')$.

Reward Selection

►Reward Shaping

Reward Shaping

Eric Wiewiora
University of California, San Diego

Synonyms

Heuristic rewards; Reward selection

Definition

Reward shaping is a technique inspired by animal training where supplemental rewards are provided to make a problem easier to learn. There is usually an obvious natural reward for any problem. For games, this is usually a win or loss. For financial problems, the reward is usually profit. Reward shaping augments the natural reward signal by adding additional rewards for making progress toward a good solution.

Motivation and Background

Reward shaping is a method for engineering a reward function in order to provide more frequent feedback on appropriate behaviors. It is most often discussed in the ►reinforcement learning framework. Providing feedback is crucial during early learning so that promising behaviors are tried early. This is necessary in large domains, where reinforcement signals may be few and far between.

A good example of such a problem is chess. The objective of chess is to win a match, and an appropriate reinforcement signal should be based on this. If an agent were to learn chess without prior knowledge, it would have to search for a great deal of time before stumbling onto a winning strategy. We can speed up this process by rewarding the agent more frequently. One possibility is to reward the learner for capturing enemy pieces, and punish the learner for losing pieces. This new reward creates a much richer learning environment, but also runs the risk of distracting the agent from the true goal (winning the game).

Another domain where feedback is extremely important is in robotics and other real-world applications. In the real world, learning requires a large amount of interaction time, and may be quite expensive. Mataric noted that in order to mitigate "thrashing" (repeatedly trying ineffective actions) rewards should be supplied as often as possible (Mataric, 1994).

If a problem is inherently described by sparse rewards, it may be difficult to change the reward structure without disrupting progress to the original goal. The behavior that is optimal with a richer reward function may be quite different from the intended behavior, even if relatively small shaping rewards are added. A classic example of this is found in Randlov and Alsrom (1998). While training an agent to control a bicycle simulation, they rewarded an agent whenever it moved toward a target destination. In response to this reward, the agent learned to ride in a tight circle, receiving reward whenever it moved in the direction of the goal.

Theory

We assume a reinforcement learning framework. For every time step t, the learner observes state s_t, takes action a_t, and receives reward r_t. The goal of reinforcement learning is to find a policy $\pi(s)$ that produces actions that optimize some long-term measurement of reward. We define the value function for every state as the expected infinite horizon discounted reward

$$V(s) = \max_{\pi} \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_t = \pi(s_t)\right],$$

where γ is the discount rate. A reinforcement learner's goal is to learn a good estimate of $V(s)$, and to use this estimate to choose a good policy.

A natural reward source should be fairly obvious from the problem at hand. Financial problems should use net monetary gain or loss as reward. Games and goal-directed problems should reward winning the game or reaching the goal. It is usually advantageous to augment this natural reward with a shaping reward f_t. We define the augmented value function V' for the reinforcement learning problem with shaping rewards

$$V'(s) = \max_{\pi'} \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t (r_t + f_t) | s_0 = s, a_t = \pi'(s_t)\right].$$

Ideally, the policy that optimizes the augmented value function will differ much from the previous optimal policy.

Constructing an appropriate shaping reward system is inherently a problem-dependent task, though a line of research aids in the implementation of these reward signals. *Potential-based shaping* provides a formal framework for translating imperfect knowledge of the relative value of states and actions into a shaping reward.

Potential-Based Shaping

Ng et al. proposed a method for adding shaping rewards in a way that guarantees the optimal policy maintains its optimality (Ng, Harada, & Russell, 1999). They define a potential function $\Phi()$ over the states. The shaping reward f for transitioning from state s to s' is defined as the discounted change in this state potential:

$$f(s, s') = \gamma\Phi(s') - \Phi(s).$$

This potential-based shaping reward is added to the natural reward for every state transition the learner experiences. Call the augmented reward $r'_t = r_t + f(s_t, s_{t+1})$, and the value function based on this reward $V'(s)$. The potential-based shaping concept can also be applied to

actions as well as states. See Wiewiora, Cottrell, & Elkan (2003) for details.

It can be shown that the augmented value function is closely related to the original:

$$V'(s) = V(s) - \Phi(s).$$

An obvious choice for the potential function is $\Phi(s) \approx V(s)$, making $V'()$ close to zero. This intuition is strengthened by results presented by Wiewiora (2003). This paper shows that for most reinforcement learning systems, the potential function acts as an initial estimate of the natural value function $V()$.

However, even if the potential function used for shaping is very close to the true natural value function, learning may still be difficult. Koenig et al. have shown that initial estimates of the value function have a large influence on the efficiency of reinforcement learning (Koenig & Simmons, 1996). With an initial estimate of the value function set below the optimal value, many reinforcement learning algorithms could require learning time exponential in the state and action space in order to find a highly rewarding state. On the other hand, in non-random environments, an optimistic initialization the value function creates learning time that is polynomial in the state-action space before a goal is found.

Cross References

►Reinforcement Learning

Recommended Reading

Koenig, S., & Simmons, R. G. (1996). The effect of representation and knowledge on goal directed exploration with reinforcement-learning algorithms. *Machine Learning, 22*(1–3), 227–250.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *International conference on machine learning*, New Brunswick, NJ (pp. 181–189). San Francisco, CA: Morgan Kaufmann.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Machine learning, proceedings of the sixteenth international conference*, Bled, Slovenia (pp. 278–287). San Francisco, CA: Morgan Kaufmann.

Randlov, J., & Alstrom, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the fifteenth international conference on machine learning*, Madison, WI. San Francisco, CA: Morgan Kaufmann.

Wiewiora, E., Cottrell, G., & Elkan, C. (2003). Principled methods for advising reinforcement learning agents. In *Machine learning, proceedings of the twentieth international conference*, Washington, DC (pp. 792–799). Menlo Park, CA: AAAI Press.

Wiewiora, E. (2003). Potential-based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research, 19*, 205–208.

RIPPER

►Rule Learning

Robot Learning

Jan Peters[1], Russ Tedrake[2], Nicholas Roy[2], Jun Morimoto[3]
[1]Max Planck Institute for Biological Cybernetics, Germany
[2]Massachusetts Institute of Technology, Cambridge, MA, USA
[3]Advanced Telecommunication Research Institute International ATR, Kyoto, Japan

Definition

Robot learning consists of a multitude of machine learning approaches, particularly ►reinforcement learning, ►inverse reinforcement learning and ►regression methods. These methods have been adapted sufficiently to domain to achieve real-time learning in complex robot systems such as helicopters, flapping-wing flight, legged robots, anthropomorphic arms, and humanoid robots.

Robot Skill Learning Problems

In classical artificial intelligence-based robotics approaches, scientists attempted to manually generate a set of rules and models that allows the robot systems to sense and act in the real world. In contrast, ►robot learning has become an interesting problem in robotics as (1) it may be prohibitively hard to program a robot for many tasks, (2) not all situations, as well as goals, may be foreseeable, and (3) real-world environments are often nonstationary (Connell and Mahadevan, 1993). Hence, future robots need to be able to adapt to the real world.

In comparison to many other machine learning domains, robot learning suffers from a variety of complex real-world problems. The real-world training time is limited and, hence, only a few complete executions of a task can ever be generated. These episodes

are frequently perceived noisily, have a large variability in the executed actions, do not cover all possible scenarios, and often do not include all reactions to external stimuli. At the same time, high-dimensional data is obtained at a fast rate (e.g., proprioceptive information at 500 Hz to 5 kHz, vision at 30–200 Hz). Hence, domain-appropriate machine learning methods are often needed in this domain.

A straightforward way to categorize robot learning approaches is given by the type of feedback (Connell and Mahadevan, 1993). A scalar performance score such as a reward or cost will often result in a ►reinforcement learning approach. A presented desired action or predicted behavior allows supervised learning approaches such as model learning or direct imitation learning. Feedback in terms of an explanation has become most prominent in apprenticeship learning. These methods will be explained in more detail in the next section. Note that unsupervised learning problems, where no feedback is required can also be found in robotics, see Ham et al. (2005) and Jenkins et al. (2006) but only for special topics.

Note that this overview on ►robot learning focuses on general problems that need to be addressed to teach robots new skills or tasks. Hence, several important specific robotics problems in specialized domains such as ►simultaneous localization and map building (SLAM) for mobile robots (Thrun et al., 2005) and unsupervised sensor fusion approaches for robot perception (Apolloni et al., 2005; Jenkins et al., 2006) are considered beyond the scope of this article.

Robot Learning Systems

As learning has found many applications in robotics, this article can only scratch the surface. It focuses on the key problem of teaching a robot new abilities with methods such as (1) Model Learning, (2) Imitation and Apprenticeship Learning, and (3) Reinforcement Learning.

Model Learning

Model learning is the machine learning counterpart to classical system identification (Farrell and Polycarpou, 2006; Schaal et al., 2003). However, while the classical approaches heavily rely on the structure of physically based models, specification of the relevant state variables and hand-tuned approximations of unknown nonlinearities, model learning approaches avoid many of these labor-intensive steps and the entire process to be more easily automated. Machine learning and system identification approaches often assume an observable state of the system to estimate the mapping from inputs to outputs of the system. However, a learning system is often able to learn this mapping including the statistics needed to cope with unidentified state variables and can hence cope with a larger class of systems. Two types of models are commonly learned, i.e., forward models and inverse models.

Forward models predict the behavior of the system based either on the current state or a history of preceeding observations. They can be viewed as "learned simulators" that may be used for optimizing a policy or for predicting future information. Examples of the application of such learned simulators range from the early work in the late 1980s by Atkeson and Schaal in robot arm-based cartpole swing-ups to Ng's recent extensions for stabilizing an inverted helicopter. Most forward models can directly be learned by **regression**.

Conversely, *inverse models* attempt to predict the input to a system in order to achieve a desired output in the next step, i.e., it uses the model of the system to directly generate control signals. In traditional control, these are often called approximation-based control systems (Farrell and Polycarpou, 2006). Inverse model learning can be solved straightforwardly by **regression** if the system dynamics are uniquely invertible, e.g., as in inverse dynamics learning for a fully actuated system. However, for underactuated or redundantly actuated systems, operational space control, etc., such a unique inverses do not exist and additional optimization is needed.

Imitation and Apprenticeship Learning

A key problem in robotics is to ease the problem of programming a complex behavior. Traditional robot programming approaches rely on accurate, manual modeling of the task and removal of all uncertainities, so that they work well. In contrast to classical robot programming, learning from demonstration approaches aim at recovering the instructions directly from a human demonstration. Numerous unsolved problems exist in this context such as discovering the intent of the teacher or determing the mapping from the teacher's kinematics to the robot's kinematics (often called the

correspondence problem). Two different approaches are common in this area: direct imitation learning and apprenticeship learning.

In *imitation learning* (Schaal et al., 2003), also known as ►behavioral cloning, the robot system directly estimates a policy from a teacher's presentation, and, subsequently, the robot system reproduces the task using this policy. A key advantage of this approach is that it can often learn a task successfully from few demonstrations. In areas where human demonstrations are straightforward to obtain, e.g., for learning racket sports, manipulation, drumming on anthropomorphic systems, direct imitation learning often proved to be an appropriate approach. Its major shortcomings are that it cannot explain why the derived policy is a good one, and it may struggle with learning from noisy demonstrations.

Hence, *apprenticeship learning* (Coates et al., 2009) has been proposed as an alternative, where a reward function is used as an explanation of the teacher's behavior. Here, the reward function is chosen under which the teacher appears to act optimally, and the optimal policy for this reward function is subsequently computed as a solution. This approach transforms the problem of learning from demonstrations onto the harder problem of approximate optimal control or reinforcement learning, hence it is also known as inverse optimal control or ►inverse reinforcement learning. As a result, it is limited to problems that can be solved by current reinforcement learning methods. Additionally, it often has a hard time dealing with tasks, where only few demonstrations with low variance exist. Hence, inverse reinforcement learning has been particularly successful in areas where it is hard for a human to demonstrate the desired behavior such as for helicopter acrobatics or in robot locomotion.

Further information on learning by demonstration may be found in Coates et al. (2009) and Schaal et al. (2003).

Robot Reinforcement Learning

The ability to self-improve with respect to an arbitrary reward function, i.e., ►reinforcement learning, is essential for robot systems to become more autonomous. Here, the system learns about its policy by interacting with its environment and receiving scores (i.e., rewards or costs) for the quality of its performance. Few off-the-shelf reinforcement learning methods scale into the domain of robotics both in terms of dimensionality and the number of trials needed to obtain an interesting behavior. Three different but overlapping styles of reinforcement learning can be found in robotics: model-based reinforcement learning, model-free ►value function approximation methods, and direct policy search (see ►Markov Decision Process).

Model-based reinforcement learning relies upon a learned forward model used for simulation-based optimization as discussed before. While often highly efficient, it frequently suffers from the fact that learned models are imperfect and, hence, the optimization method can be guaranteed to be biased by the errors in the model. A full Bayesian treatment of model uncertainty appears to be a promising way for alleviating this shortcoming of this otherwise powerful approach.

Value function approximation methods have been the core approach used in reinforcement learning during the 1990s. These techniques rely upon approximating the expected rewards for every possible action in every visited state. Subsequently, the controller chooses the actions in accordance to this value. Such approximation requires a globally consistent value function, where the quality of the policy is determined by the largest error of the value function at any possible state. As a result, these methods have been problematic for anthropomorphic robotics as the high-dimensional domains often defy learning such a global construct. However, it has been highly sucessful in low-dimensional domains such as mobile vehicle control and robot soccer with wheeled robots as well as on well-understood test domains such as cart-pole systems.

Unlike the previous two approaches, *policy search* attempts to directly learn the optimal policy from experience without solving intermediary learning problems. Policies often have significantly fewer parameters than models or value functions. For example, for the control of a prismatic robot optimally with respect to a quadratic reward function, the number of policy parameters grows linearly in the number of state dimensions, while it grows quadratically in the size of the model and value function (this part is well-known as this problem is analytically tractable). In general cases, the number of parameters of value functions does often

even grow exponentially in the number of states (which is known as the "Curse of Dimensionality"). This insight has given rise to policy search methods, particularly, ►policy gradient methods and probabilistic approaches to policy search such as the reward-weighted regression or PoWER (Kober and Peters, 2009). To date, application results of direct policy search approaches range from gait optimization in locomotion (Tedrake et al., 2004) to various motor learning examples (e.g., Ball-in-a-Cup, T-Ball, or throwing darts, see e.g., Kober and Peters, 2009).

Further information on reinforcement learning for robotics may be found in Connell and Mahadevan (1993), Kober and Peters (2009), Riedmiller et al. (2009), and Tedrake et al. (2004).

Application Domains

The possible application domains for robot learning have not been fully explored; one could even aggressively state that a huge number of challenges remain to be fully addressed in order to solve the problem of robot learning. Nevertheless, robot learning has been successful in several application domains.

For accurate execution of desired trajectories, model learning has been scaled to learning the full inverse dynamics of a humanoid robot in real time more accurately than achievable with physical models (Schaal et al., 2002). Current work focuses mainly on improving the concurrent execution of tasks as well as control of redundant or underactuated systems.

Various approaches have been successful in task learning. Learning by demonstration approaches are moving increasingly toward industrial grade solutions, where fast training of complex tasks becomes possible. Skills ranging from motor toys, e.g., basic movements, paddling a ball, to complex tasks such as cooking a complete meal, basic table tennis strokes, helicopter acrobatics, or footplacement in locomotion have been learned from human teachers. Reinforcement learning has yielded better gaits in locomotion, jumping behaviors for legged robots, perching with fixed wing flight robots, forehands in table tennis as well as various applications of learning to control motor toys used for the motor development of children.

Cross References

►Behavioral Cloning
►Inverse Reinforcement Learning
►Policy Search
►Reinforcement Learning
►Value Function Approximation

Recommended Reading

Recently, several special issues (Morimoto et al., 2010; Peters and Ng, 2009) and books (Sigaud, 2010) have covered the domain of robot learning. The classical book (Connell and Mahadevan, 1993) is interesting nearly 20 years after its publication. Additional special topics are treated in Apolloni et al. (2005) and Thrun et al. (2005).

Apolloni, B., Ghosh, A., Alpaslan, F. N., Jain, L. C., & Patnaik, S. (2005). *Machine learning and robot perception. Studies in computational intelligence* (Vol. 7). Berlin: Springer.

Coates, A., Abbeel, P., & Ng, A. Y. (2009). Apprenticeship learning for helicopter control. *Communications of the ACM, 52*(7), 97–105.

Connell, J. H., & Mahadevan, S. (1993). *Robot learning*. Dordrecht: Kluwer Academic.

Farrell, J. A., & Polycarpou, M. M. (2006). *Adaptive approximation based control. Adaptive and learning systems for signal processing, communications and control series*. Hoboken: John Wiley.

Ham, J., Lin, Y., & Lee, D. D. (2005). Learning nonlinear appearance manifolds for robot localization. In *International conference on intelligent robots and Systems*, Takamatsu, Japan.

Jenkins, O., Bodenheimer, R., & Peters, R. (2006). Manipulation manifolds: Explorations into uncovering manifolds in sensory-motor spaces (8 pages). In *International conference on development and learning*, Bloomington, IN

Kober, J., & Peters, J. (2009). Policy search for motor primitives in robotics. In *Advances in neural information processing systems* 22. Cambridge: MIT Press.

Morimoto, J., Toussaint, M., & Jenkins, C. (2010). Special issue on robot learning in practice. *IEEE Robotics and Automation Magazine, 17*(2), 17–84.

Peters, J., & Ng, A. (2009). Special issue on robot learning. *Autonomous Robots, 27*(1–2):1–144.

Peters, J., & Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks, 21*(4): 682–697.

Riedmiller, M., Gabel, T., Hafner, R., & Lange, S. (July 2009). Reinforcement learning for robot soccer. *Autonomous Robots, 27*(1):55–73.

Schaal, S., Atkeson, C. G., & Vijayakumar, S. Scalable techniques from nonparameteric statistics for real-time robot learning. *Applied Intelligence, 17*(1):49–60.

Schaal, S., Ijspeert, A., & Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transaction of the Royal Society of London: Series B, Biological Sciences, 358*(1431):537–547.

Sigaud, O., & Peters, J. (2010). *From motor learning to interaction learning in robots. Studies in computational intelligence* (Vol. 264). Heidelberg: Springer.
Tedrake, R., Zhang, T. W., & Seung, H. S. (2004). Stochastic policy gradient reinforcement learning on a simple 3d biped. In *Proceedings of the IEEE international conference on intelligent robots and systems* (pp. 2849–2854). IROS 2004, Sendai, Japan.
Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. Cambridge: MIT Press.

ROC Analysis

Peter A. Flach
University of Bristol
Bristol, UK

Synonyms

Receiver operating characteristic analysis

Definition

ROC analysis investigates and employs the relationship between ►sensitivity and ►specificity of a binary classifier. *Sensitivity* or ►*true positive rate* measures the proportion of positives correctly classified; *specificity* or ►*true negative rate* measures the proportion of negatives correctly classified. Conventionally, the true positive rate (tpr) is plotted against the ►*false positive rate* (fpr), which is one minus true negative rate. If a classifier outputs a score proportional to its belief that an instance belongs to the positive class, decreasing the ►decision threshold – above which an instance is deemed to belong to the positive class – will increase both true and false positive rates. Varying the decision threshold from its maximal to its minimal value results in a piecewise linear curve from $(0,0)$ to $(1,1)$, such that each segment has a nonnegative slope (Fig. 1). This ROC curve is the main tool used in ROC analysis. It can be used to address a range of problems, including: (1) determining a decision threshold that minimizes ►error rate or misclassification cost under given class and cost distributions; (2) identifying regions where one classifier outperforms another; (3) identifying regions where a classifier performs worse than chance; and (4) obtaining calibrated estimates of the class posterior.

Motivation and Background

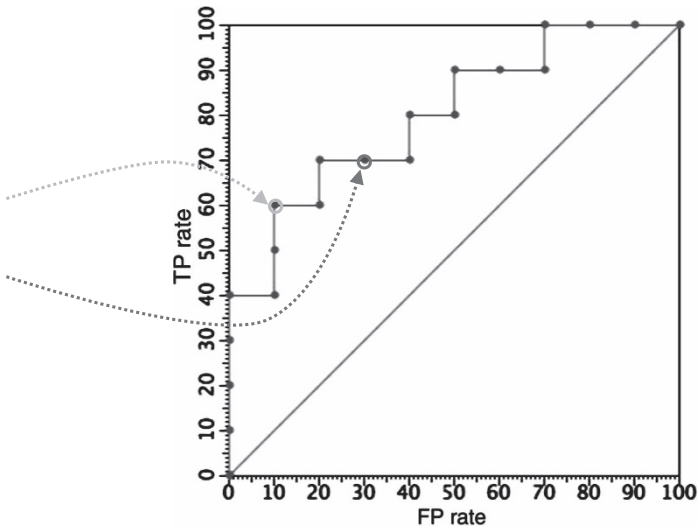
ROC analysis has its origins in *signal detection theory* (Egan, 1975). In its simplest form, a detection problem involves determining the value of a binary signal contaminated with random noise. In the absence of any other information, the most sensible decision threshold would be halfway between the two signal values. If the noise distribution is zero-centered and symmetric, sensitivity and specificity at this threshold have the same expected value, which means that the corresponding *operating point* on the ROC curve is located at the intersection with the descending diagonal tpr + fpr = 1. However, we may wish to choose different operating points, for instance because false negatives and false positives have different costs. In that case, we need to estimate the noise distribution.

A slight reformulation of the signal detection scenario clarifies its relevance in a machine learning setting. Instead of superimposing random noise on a deterministic signal, we can view the resulting noisy signal as coming from a ►mixture distribution consisting of two component distributions with different means. The detection problem is now to decide, given a received value, from which component distribution it was drawn. This is essentially what happens in a binary ►classification scenario, where the scores assigned by a trained classifier follow a mixture distribution with one component for each class. The random variations in the data are translated by the classifier into random variations in the scores, and the classifier's performance depends on how well the per-class score distributions are separated. Figure 2 illustrates this for both discrete and continuous distributions. In practice, empirical ROC curves and distributions obtained from a test set are discrete because of the finite resolution supplied by the test set. This resolution is further reduced if the classifier only assigns a limited number of different scores, as is the case with ►decision trees; the histogram example illustrates this.

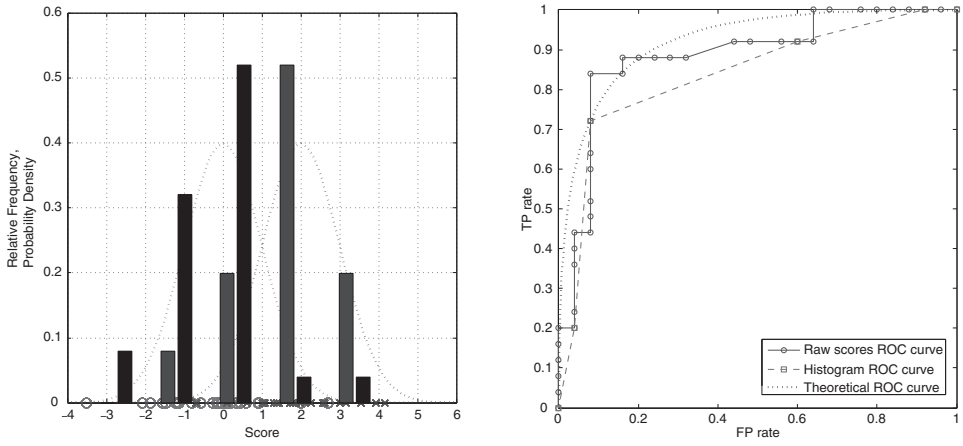
Solutions

For convenience we will assume henceforth that score distributions are discrete, and that decision thresholds always fall between actual scores (the results easily generalize to continuous distributions using probability

Class	Score
+	0.98
+	0.93
+	0.87
+	0.84
–	0.79
+	0.73
+	0.67
–	0.62
+	0.57
–	0.54
–	0.48
+	0.43
–	0.37
+	0.34
–	0.28
–	0.24
+	0.18
–	0.12
–	0.09
–	0.03



ROC Analysis. Figure 1. The table on the *left* gives the scores assigned by a classifier to ten positive and ten negative examples. Each threshold on the classifier's score results in particular true and false positive rates: e.g., thresholding the score at 0.5 results in three misclassified positives (tpr = 0.7) and three misclassified negatives (fpr = 0.3); thresholding at 0.65 yields tpr = 0.6 and fpr = 0.1. Considering all possible thresholds gives the ROC curve on the right; this curve can also be constructed without explicit reference to scores, by going down the examples sorted on decreasing score and making a step up (to the right) if the example is positive (negative)



ROC Analysis. Figure 2. (*left*) Artificial classifier "scores" for two classes were obtained by sampling 25 points each from two ▶Gaussian distributions with mean 0 and 2, and unit variance. The figure shows the raw scores on the *x*-axis and normalized histograms obtained by uniform five-bin discretization. (*right*) The jagged ROC curve was obtained by thresholding the raw scores as before. The histogram gives rise to a smoothed ROC curve with only five segments. The dotted line is the theoretical curve obtained from the true Gaussian distributions

density functions). There is a useful duality between thresholds and scores: decision thresholds correspond to operating points connecting two segments in the ROC curve, and actual scores correspond to segments of the ROC curve connecting two operating points. Let $f(s|+)$ and $f(s|-)$ denote the relative frequency of positive (negative) examples from a test set being assigned score s. (Note that s itself may be an estimate of the likelihood $p(x|+)$ of observing a positive example with feature vector x. We will return to this later.)

Properties of ROC Curves

The first property to note is that the true (false) positive rate achieved at a certain decision threshold t is the proportion of the positive (negative) score distribution to the right of the threshold; that is, $\text{tpr}(t) = \sum_{s>t} f(s|+)$ and $\text{fpr}(t) = \sum_{s>t} f(s|-)$. In Fig. 2, setting the threshold at 1 using the discretized scores gives a true positive rate of 0.72 and a false positive rate of 0.08, as can be seen by summing the bars of the histogram to the right of the threshold. Although the ROC curve does not display thresholds or scores, this allows us to reconstruct the range of thresholds yielding a particular operating point from the score distributions.

If we connect two distinct operating points on an ROC curve by a straight line, the slope of that line segment is equal to the ratio of positives to negatives in the corresponding score interval; that is,

$$\text{slope}(t_1, t_2) = \frac{\text{tpr}(t_2) - \text{tpr}(t_1)}{\text{fpr}(t_2) - \text{fpr}(t_1)} = \frac{\sum_{t_1<s<t_2} f(s|+)}{\sum_{t_1<s<t_2} f(s|-)}$$

Choosing the score interval small enough to cover a single segment of the ROC curve corresponding to score s, it follows that the segment has slope $f(s|+)/f(s|-)$.

This can be verified in Fig. 2: e.g., the top-right segment of the smoothed curve has slope 0 because the leftmost bin of the histogram contains only negative examples. For continuous distributions the slope of the ROC curve at any operating point is equal to the ratio of probability densities at that score.

It can happen that $\text{slope}(t_1, t_2) < \text{slope}(t_1, t_3) < \text{slope}(t_2, t_3)$ for $t_1 < t_2 < t_3$, which means that the ROC curve has a "dent" or *concavity*. This is inevitable when using raw classifier scores (unless the positives and negatives are perfectly separated), but can also be observed in the smoothed curve in the example: the rightmost bin of the histogram has a positive-to-negative ratio of 5, while the next bin has a ratio of 13. Consequently, the two leftmost segments of the ROC curve display a slight concavity. It means that performance can be improved by combining the two bins, leading to one large segment with slope 9. In other words, ROC curve concavities demonstrate locally suboptimal behavior of a classifier. An extreme case of suboptimal behavior occurs if the entire curve is concave, or at least below the ascending diagonal: in that case, performance can simply be improved by assigning all test instances the same score, resulting in an ROC curve that follows the ascending diagonal. A *convex* ROC curve is one without concavities.

The AUC Statistic

The most important statistic associated with ROC curves is the *area under* (ROC) *curve* or AUC. Since the curve is located in the unit square, we have $0 \leq \text{AUC} \leq 1$. $\text{AUC} = 1$ is achieved if the classifier scores every positive higher than every negative; $\text{AUC} = 0$ is achieved if every negative is scored higher than every positive. $\text{AUC} = 1/2$ is obtained in a range of different scenarios, including: (1) the classifier assigns the same score to all test examples, whether positive or negative, and thus the ROC curve is the ascending diagonal; (2) the per-class score distributions are similar, which results in an ROC curve close (but not identical) to the ascending diagonal; and (3) the classifier gives half of a particular class the highest scores, and the other half, the lowest scores. Note that, although a classifier with AUC close to 1/2 is often said to perform randomly, there is nothing random in the third classifier: rather, its excellent performance on some of the examples is counterbalanced by its very poor performance on some others. (Sometimes a linear rescaling 2AUC−1 called the Gini coefficient is preferred, which has a related use in the assessment of income or wealth distributions using Lorenz curves: a Gini coefficient close to 0 means that income is approximately evenly distributed. Notice that this Gini coefficient is often called the Gini index, but should not be confused with the impurity measure used in decision tree learning).

AUC has a very useful statistical interpretation: it is the expectation that a (uniformly) randomly drawn positive receives a higher score than a randomly drawn negative. It is a normalized version of the *Wilcoxon–Mann–Whitney sum of ranks test*, which tests the null hypothesis that two samples of ordinal measurements are drawn from a single distribution. The "sum of ranks" epithet refers to one method to compute this statistic, which is to assign each test example an integer rank according to decreasing score (the highest scoring example gets rank 1, the next gets rank 2, etc.); sum up the ranks of the n^- negatives, which have to be high; and subtract $\sum_{i=1}^{n^-} i = n^-(n^- + 1)/2$ to achieve 0 if all negatives are ranked first. The AUC statistic is then obtained by normalizing by the number of pairs of one positive

and one negative, n^+n^-. There are several other ways to calculate AUC: for instance, we can calculate, for each negative, the number of positives preceding it, which basically is a columnwise calculation and yields an alternative view of AUC as the expected true positive rate if the operating point is chosen just before a randomly drawn negative.

Identifying Optimal Points and the ROC Convex Hull

In order to select an operating point on an ROC curve, we first need to specify the objective function that we aim to optimize. In the simplest case this will be ►accuracy, the proportion of correctly predicted examples. Denoting the proportion of positives by pos, we can express accuracy as a weighted average of the true positive and true negative rates $\text{pos} \cdot \text{tpr} + (1 - \text{pos})(1 - \text{fpr})$. It follows that points with the same accuracy lie on a straight line with slope $(1 - \text{pos})/\text{pos}$; these parallel lines are the *isometrics* for accuracy (Flach, 2003). In order to find the optimal operating point for a given class distribution, we can start with an accuracy isometric through $(0,1)$ and slide it down until it touches the ROC curve in one or more points (Fig. 3 *(left)*). In the case of a single point this uniquely determines the operating point and thus, the threshold. If there are several points in common between the accuracy isometric and the ROC curve, we can make an arbitrary choice, or interpolate stochastically. We can read off the achieved accuracy by intersecting the accuracy isometric with the descending diagonal, on which $\text{tpr} = 1 - \text{fpr}$ and therefore the true positive rate at the intersection point is equal to the accuracy associated with the isometric.

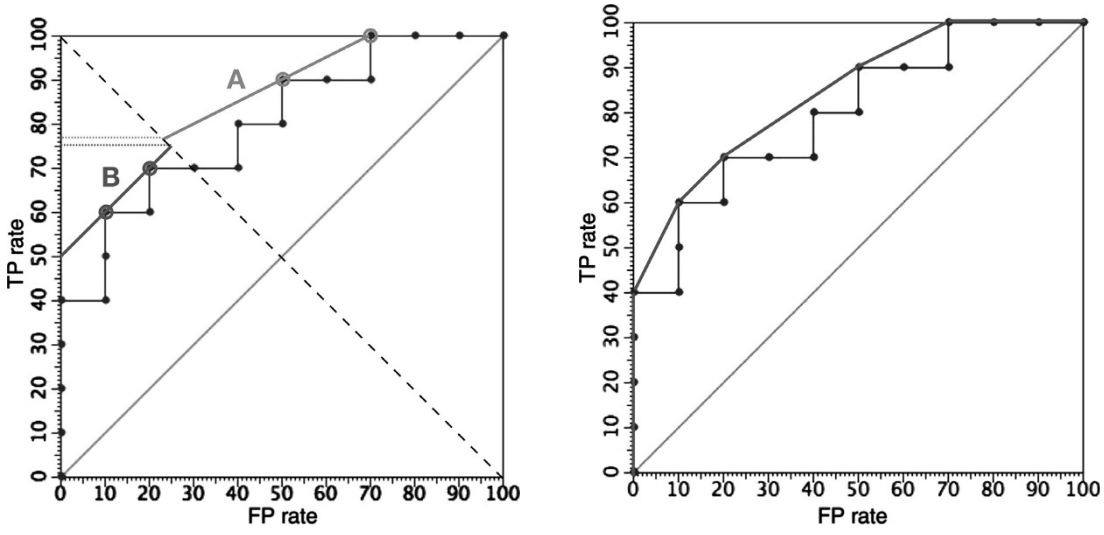
We can generalize this approach to any objective function that is a linear combination of true and false positive rates. For instance, let predicting class i for an instance of class j incur cost $\text{cost}(i|j)$, so for instance the cost of a false positive is $\text{cost}(+|-)$ (profits for correct predictions are modeled as negative costs, e.g., $\text{cost}(+|+) < 0$). Cost isometrics then have slope $(\text{cost}(+|-) - \text{cost}(-|-))/(\text{cost}(-|+) - \text{cost}(+|+))$. Nonuniform class distributions are simply taken into account by multiplying the class and cost ratio, giving a single *skew ratio* expressing the relative importance of negatives compared to positives.

This procedure of selecting an optimal point on an ROC curve can be generalized to select among points lying on more than one curve, or even an arbitrary set of points (e.g., points representing different categorical classifiers). In such scenarios, it is likely that certain points are never selected for any skew ratio; such points are said to be *dominated*. For instance, points on a concave region of an ROC curve are dominated. The nondominated points are optimal for a given closed interval of skew ratios, and can be joined to form the *convex hull* of the given ROC curve or set of ROC points (Fig. 3 *(right)*). (In multiobjective optimization, this concept is called the *Pareto front*.) This notion of the ROC convex hull (sometimes abbreviated as ROCCH) is extremely useful in a range of situations. For instance, if an ROC curve displays concavities, the convex hull represents a discretization of the scores which achieves higher AUC. Alternatively, the convex hull of a set of categorical classifiers can be interpreted as a hybrid classifier that can reach any point on the convex hull by stochastic interpolation between two neighboring classifiers (Provost & Fawcett, 2001).

Obtaining Calibrated Estimates of the Class Posterior

Recall that each segment of an ROC curve has slope $\text{slope}(s) = f(s|+)/f(s|-)$, where s is the score associated with the segment, and $f(s|+)$ and $f(s|-)$ are the relative frequencies of positives and negatives of assigned score s. Now consider the function $\text{cal}(s) = \text{slope}(s)/(\text{slope}(s) + 1) = f(s|+)/(f(s|+) + f(s|-))$: the *calibration map* $s \mapsto \text{cal}(s)$ adjusts the classifier's scores to reflect the empirical probabilities observed in the test set. If the ROC curve is convex, $\text{slope}(s)$ and $\text{cal}(s)$ are monotonically nonincreasing with decreasing s, and thus replacing the scores s with $\text{cal}(s)$ does not change the ROC curve (other than merging neighboring segments with different scores but the same slope into a single segment).

Consider decision trees as a concrete example. Once we have trained (and possibly pruned) a tree, we can obtain a score in each leaf l by taking the ratio of positive to negative training examples in that leaf: $\text{score}(l) = p(+|l)/p(-|l)$. These scores represent posterior odds, taking into account the class prior in the training set. Each leaf gives rise to a different segment of the ROC curve, which, by the nature of how the scores were calculated, will be convex. The calibrated scores $\text{cal}(\text{score}(l))$ then represent an adjusted estimate of the positive posterior that replaces the training set prior with a uniform prior. To see this, notice that duplicating

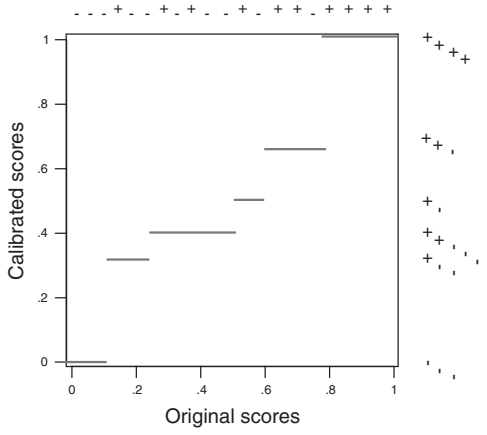


ROC Analysis. Figure 3. (*left*) **The slope of accuracy isometrics reflects the class ratio. Isometric A has slope 1/2: this corresponds to having twice as many positives as negatives, meaning that an increase in true positive rate of *x* is worth a 2*x* increase in false positive rate. This selects two optimal points on the ROC curve. Isometric B corresponds to a uniform class distribution, and selects optimal points which make fewer positive predictions. In either case, the achieved accuracy can be read off on the *y*-axis after intersecting the isometric with the descending diagonal (slightly higher for points selected by A).** (*right*) **The convex hull selects those points on an ROC curve which are optimal under some class distribution. The slope of each segment of the convex hull gives the class ratio under which the two end points of the segment yield equal accuracy. All points under the convex hull are nonoptimal**

all positive training examples would amplify all uncalibrated scores score(l) with a factor 2, but the ROC curve and therefore the calibrated probability estimates remain unchanged.

If the ROC curve is not convex, the mapping $s \mapsto \text{cal}(s)$ is not monotonic; while the scores cal(s) would lead to improved performance on the data from which the ROC curve was derived, this is very unlikely to generalize to other data, and thus leads to ►overfitting. This is why, in practice, a less drastic calibration procedure involving the convex hull is applied (Fawcett & Niculescu-Mizil, 2007). Let s_1 and s_2 be the scores associated with the start and end segments of a concavity, i.e., $s_1 > s_2$ and $\text{slope}(s_1) < \text{slope}(s_2)$. Let $\text{slope}(s_1 s_2)$ denote the slope of the line segment of the convex hull that repairs this concavity, which implies $\text{slope}(s_1) < \text{slope}(s_1 s_2) < \text{slope}(s_2)$. The calibration map will then map any score in the interval $[s_1, s_2]$ to $\text{slope}(s_1 s_2)/(\text{slope}(s_1 s_2) + 1)$ (Fig. 4).

This ROC-based calibration procedure, which is also known as *isotonic regression* (Zadrozny & Elkan,



ROC Analysis. Figure 4. The piecewise constant calibration map derived from the convex hull in Fig. 3. The original score distributions are indicated at the top of the figure, and the calibrated distributions are on the right. We can clearly see the combined effect of binning the scores and redistributing them over the interval $[0,1]$

2002), not only produces calibrated probability estimates but also improves AUC. This is in contrast with other calibration procedures such as logistic calibration which do not bin the scores and therefore do not change the ROC curve. ROC-based calibration can be shown to achieve lowest *Brier score* (Brier, 1950), which measures the mean squared error in the probability estimates as compared with the ideal probabilities (1 for a positive and 0 for a negative), among all probability estimators that do not reverse pairwise rankings. On the other hand, being a nonparametric method it typically requires more data than parametric methods in order to estimate the bin boundaries reliably.

Future Directions

ROC analysis in its original form is restricted to binary classification, and its extension to more than two classes gives rise to many open problems. c-class ROC analysis requires $c(c-1)$ dimensions, in order to distinguish each possible misclassification type. Srinivasan proved that basic concepts such as the ROC polytope and its linearly interpolated convex hull generalize to the c-class case (Srinivasan, 1999). In theory, the volume under the ROC polytope can be employed for assessing the quality of a multiclass classifier (Ferri, Hernández-Orallo, & Salido, 2003), but this volume is hard to compute as – unlike the two-class case, where the segments of an ROC curve can simply be enumerated in $O(n \log n)$ time by sorting the n examples on their score (Fawcett, 2006; Flach, 2004) – there is no simple way to enumerate the ROC polytope. Mossman considers the special case of 3-class ROC analysis, where for each class the two possible misclassifications are treated equally (the so-called *one-versus-rest* scenario) (Mossman, 1999). Hand and Till propose the average of all one-versus-rest AUCs as an approximation of the area under the ROC polytope (Hand & Till, 2001). Various algorithms for minimizing a classifier's misclassification costs by reweighting the classes are considered in Bourke, Deng, Scott, Schapire, and Vinodchandran (2008) and Lachiche and Flach (2003).

Other research directions include the explicit visualization of misclassification costs (Drummond & Holte, 2006), and using ROC analysis to study the behavior of machine learning algorithms and the relations between machine learning metrics (Fuernkranz & Flach, 2005).

Cross References

- ▶Accuracy
- ▶Class Imbalance Problem
- ▶Classification
- ▶Confusion Matrix
- ▶Cost-Sensitive Learning
- ▶Error Rate
- ▶False Negative
- ▶False Positive
- ▶Gaussian Distribution
- ▶Posterior Probability
- ▶Precision
- ▶Prior Probability
- ▶Recall
- ▶Sensitivity
- ▶Specificity
- ▶True Negative
- ▶True Positive

Recommended Reading

Bourke, C., Deng, K., Scott, S., Schapire, R., & Vinodchandran, N. V. (2008). On reoptimizing multi-class classifiers. *Machine Learning, 71*(2–3), 219–242.

Brier, G. (1950). Verification of forecasts expressed in terms of probabilities. *Monthly Weather Review, 78*, 1–3.

Drummond, C., & Holte, R. (2006). Cost curves: An improved method for visualizing classifier performance. *Machine Learning, 65*(1), 95–130.

Egan, J. (1975). *Signal detection theory and ROC analysis. Series in cognitition and perception*. New York: Academic Press.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*(8), 861–874.

Fawcett, T., & Niculescu-Mizil, A. (2007). PAV and the ROC convex hull. *Machine Learning, 68*(1), 97–106.

Ferri, C., Hernández-Orallo, J., & Salido, M. (2003). Volume under the ROC surface for multi-class problems. In *Proceedings of the fourteenth (ECML 2003)* (pp. 108–120). Lecture Notes in Computer Science 2837. Berlin: Springer.

Flach, P. (2003). The geometry of ROC space: Understanding machine learning metrics through ROC isometrics. In *Proceedings of the twentieth international conference on machine learning (ICML 2003)* (pp. 194–201). Washington, DC: AAAI Press.

Flach, P. (2004). *The many faces of ROC analysis in machine learning*. ICML-04 Tutorial. http://www.cs.bris.ac.uk/flach/ICML04tutorial/. Accessed on 16 December 2009.

Fuernkranz, J., & Flach, P. (2005). ROC 'n' Rule learning – towards a better understanding of covering algorithms. *Machine Learning, 58*(1), 39–77.

Hand, D., & Till, R. (2001). A simple generalization of the area under the ROC curve to multiple class classification problems. *Machine Learning, 45*(2), 171–186.

Lachiche, N., & Flach, P. (2003). Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC

curves. In *Proceedings of the twentieth international conference on machine learning (ICML'03)* (pp. 416–423). Washington, DC: AAAI Press.
Mossman, D. (1999). Three-way ROCs. *Medical Decision Making, 19*, 78–89.
Provost, F., & Fawcett, T. (2001). Robust classification for imprecise environments. *Machine Learning, 42*(3), 203–231.
Srinivasan, A. (1999). *Note on the location of optimal classifiers in n-dimensional ROC space.* Technical report PRG-TR-2-99. Oxford University Computing Laboratory, Oxford.
Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the 8th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 694–699). New York: ACM.

ROC Convex Hull

The convex hull of an ▶ROC curve is a geometric construction that selects the points on the curve that are optimal under some class and cost distribution. It is analogous to the Pareto front in multiobjective optimization. See ▶ROC Analysis.

ROC Curve

The ROC curve is a plot depicting the trade-off between the ▶true positive rate and the ▶false positive rate for a classifier under varying decision thresholds. See ▶ROC Analysis.

Rotation Forests

Rotation Forests is an ▶ensemble learning technique. It is similar to the ▶Random Forests approach to building decision tree ensembles. In the first step, the original feature set is split randomly into K disjoint subsets. Next, ▶principal components analysis is used to extract n principal component dimensions from each of the K subsets. These are then pooled, and the original data projected linearly into this new feature space. A tree is then built from this data in the usual manner. This process is repeated to create an ensemble of trees, each time with a different random split of the original feature set.

As the tree learning algorithm builds the classification regions using hyperplanes parallel to the feature axes, a small rotation of the axes may lead to a very different tree. The effect of rotating the axes is that classification regions of high accuracy can be constructed with far fewer trees than in ▶Bagging and ▶Adaboost.

RSM

▶Random Subspace Method

Rule Learning

Johannes Fürnkranz
Fachbereich Informatik, Darmstadt, Germany

Synonyms

AQ; Covering algorithm; CN2; Foil; Laplace estimate; *m*-estimate; OPUS; RIPPER

Definition

Inductive rule learning solves a ▶classification problem via the induction of a ▶rule set or a ▶decision list. The principal approach is the so-called *separate-and-conquer* or *covering* algorithm, which learns one rule at a time, successively removing the covered examples. Individual algorithms within this framework differ primarily in the way they learn single rules. A more extensive survey of this family of algorithms can be found in Fürnkranz (1999).

The Covering Algorithm

Most covering algorithms operate in a ▶concept learning framework, that is, they assume a set of positive and negative training examples. Adaptations to the multi-class case are typically performed via ▶class binarization, transforming the original problem into a set of binary problems. Some algorithms, most notably CN2 (Clark & Niblett, 1989; Clark & Boswell, 1991), learn multi-class rules directly by optimizing over all possible classes in the head of the rule. In this case, the resulting theory is interpreted as a decision list. In the following, a two-class problem with a positive and a negative class will be assumed.

```
procedure COVERING (Examples, Classifier)
Input: Examples, a set of positive and negative
                 examples for a class c.
// initialize the classifier
Classifier = ∅
// loop until no more positive examples are covered
while POSITIVE (Examples) ≠ ∅ do
    // find the best rule for the current examples
    Rule = FINDBESTRULE (Examples)
    // check if we need more rules
    if RULESTOPPINGCRITERION (Classifier,
                              Rule, Examples)
    then  break while
    // remove covered examples and add rule to rule set
    Examples = Examples \ COVER (Rule, Examples)
    Classifier = Classifier ∪ Rule
endwhile
// post-process the rule set (e.g., pruning)
Classifier = POSTPROCESSING (Classifier)
Output: Classifier
```

```
procedure FINDBESTRULE (Examples, BestRule)
Input: Examples, a set of positive and negative
                 examples for a class c.
InitRule = INITIALIZERULE (Examples)
InitVal = EVALUATERULE (InitRule)
BestRule = <InitVal, InitRule>
Rules = {BestRule}
while Rules ≠ ∅ do
  Candidates = SELECTCANDIDATES(Rules, Examples)
  Rules = Rules \ Candidates
  for Candidate ∈ Candidates do
    Refinements = REFINERULE(Candidate, Examples)
      for Refinement ∈ Refinements do
        Evaluation = EVALUATERULE (Refinement,
                                   Examples)
          if STOPPINGCRITERION(Refinement,
                               Examples)
          then  next Refinement
          NewRule = <Evaluation, Refinement>
          Rules = INSERTSORT(NewRule, Rules)
          if NewRule > BestRule
          then  BestRule = NewRule
      endfor
  endfor
  Rules = FILTERRULES(Rules, Examples)
endwhile
Output: BestRule
```

The COVERING algorithm starts with an empty theory. If there are any positive examples in the training set it calls the subroutine FINDBESTRULE for learning a single rule that will cover a subset of the positive examples (and possibly some negative examples as well). All covered examples are then separated from the training set, the learned rule is added to the theory, and another rule is learned from the remaining examples. Rules are learned in this way until no positive examples are left or until the RULESTOPPINGCRITERION fires. In the simplest case, the stopping criterion is a check whether there are still remaining positive examples that need to be covered. The resulting theory may also undergo some POSTPROCESSING, for example, a separate pruning and re-induction phase as in RIPPER (Cohen, 1995).

In the following, these components will be discussed in more detail.

Finding the Best Rule

Single rules are typically found by searching the space of possible rules for a rule that optimizes a given quality criterion defined in EVALUATERULE. The value of this heuristic function is the higher the more positive and the less negative examples are covered by the candidate rule. FINDBESTRULE maintains *Rules*, a sorted list of candidate rules, which is initialized by the procedure INITIALIZERULE. New rules will be inserted in appropriate places (INSERTSORT), so that *Rules* will always be sorted in decreasing order of the heuristic evaluations of the rules. At each cycle, SELECTCANDIDATES selects a subset of these candidate rules, which are then refined using the ▶refinement operator REFINERULE. Each refinement is evaluated and inserted into the sorted *Rules* list unless the STOPPINGCRITERION prevents this. If the evaluation of the *NewRule* is better than the best rule found previously, *BestRule* is set to *NewRule*. FILTERRULES selects the subset of the ordered rule list that will be used in subsequent iterations. When all candidate rules have been processed, the best rule is returned.

Different choices of these functions allow the definition of different biases for the separate-and-conquer learner. The ▶search bias is defined by the choice of a search strategy (InitializeRule and RefineRule), a search algorithm (SelectCandidates and FilterRules), and a search heuristic (EvaluateRule). The refinement operator RefineRule constitutes the ▶language bias of the algorithm. An overfitting avoidance bias can be implemented via some StoppingCriterion and/or in a post-processing phase.

For example, InitializeRule and RefineRule may be defined so that they realize a top-down (general-to-specific), a bottom-up (specific-to-general) or a bidirectional search. Exhaustive ▶breadth-first, ▶depth-first, or best-first searches can be realized by appropriate choices of EvaluateRule, and no filtering or candidate selection. FilterRules can, for example, be used to realize a ▶hill-climbing or ▶beam search by maintaining only the best or the *BeamWidth* best rules. Evolutionary algorithms and stochastic local search can also be easily realized.

The most common algorithm for finding the best rule is a top-down hill-climbing algorithm. It basically constructs a rule by consecutively adding conditions to the rule body so that a given quality criterion is greedily optimized. This constitutes a simple greedy hill-climbing algorithm for finding a local optimum in the hypothesis space defined by the feature set. InitializeRule will thus return the most general rule, the rule with the body `{true}`, and RefineRule will return all possible extensions of the rule by a single condition. FilterRules will only let the best refinement pass for the next iteration, so that SelectCandidates will always have only one choice. The search heuristic, the stopping criterion, and the post-processing are discussed in the next sections.

Rule Learning Heuristics

The covering algorithm tries to find a rule set that is as *complete* and *consistent* as possible. Thus, each rule should cover as many positive examples and as few negative examples as possible. The exact trade-off between these two objectives is realized via the choice of a rule learning heuristic. A few important ones are (assume that p out of P positive examples and n out of N negative examples are covered by the rule):

Laplace estimate $\left(\text{Lap} = \frac{p+1}{p+n+2}\right)$ computes the fraction of positive examples in all covered examples, where each class is initialized with one virtual example in order to penalize rules with low coverage.

m-estimate $\left(\text{m} = \frac{p+m\cdot P/(P+N)}{p+n+m}\right)$ is a generalization of the Laplace estimate which uses m examples for initialization, which are distributed according to the class distribution in the training set (Cestnik, 1990).

Information gain $\left(\text{ig} = p\cdot\left(\log_2\frac{p}{p+n} - \log_2\frac{p'}{p'+n'}\right)\right)$, where p' and n' are the number of positive and negative examples covered by the rule's predecessor) is Quinlan's (1990) adaptation of the information gain heuristic used for decision tree learning. The main difference is that this only focuses on a single branch (a rule), whereas the decision tree version tries to optimize all successors of a node simultaneously.

Correlation and χ^2 $\left(\text{corr} = \frac{p(N-n)-(P-p)n}{\sqrt{PN(p+n)(P-p+N-n)}}\right)$ computes the four-field correlation of covered/uncovered positive/negative examples. It is equivalent to a χ^2 statistic ($\chi^2 = (P+N)\,\text{corr}^2$).

An exhaustive overview and theoretical comparison of various search heuristics in coverage space, a variant of ▶ROC space can be found in Fürnkranz and Flach (2005).

Overfitting Avoidance

It is trivial to find a rule set that is complete and consistent on the training data. To achieve this, one only needs to convert each positive example into a rule. Each of these rules is consistent (provided the data set is not inconsistent), and collectively they cover the entire example set (completeness). However, this is clearly a bad case of ▶overfitting because the theory will not generalize to new positive examples.

Overfitting is to some extent handled by the search heuristics described above, but most algorithms use additional ▶pruning techniques. One can discriminate between ▶*pre-pruning* techniques, where a separate criterion is used to filter out unpromising rules. For example, CN2 computes the *likelihood ratio statistic* $\text{lrs} = 2\cdot\left(p\log\frac{p}{e_p} + n\log\frac{n}{e_n}\right)$, where $e_p = (p+n)\frac{P}{P+N}$ and $e_n = (p+n)\frac{N}{P+N} = (p+n) - e_p$ are the number of positive and negative examples one could expect if the $p+n$ examples

covered by the rule were distributed in the same way as the $P + N$ examples in the full data set. This statistic follows a χ^2 distribution, which allows to filter out rules for which the distribution of the covered examples is not statistically and significantly different from the distribution of examples in the full data set. Other pre-pruning criteria are simple thresholds that define a minimum acceptable value for the search heuristic, or FOIL's ▶minimum description length criterion that relates the length of a rule to the number of examples it covers.

However, it can be shown experimentally that CN2 or FOIL still have a tendency to overfit the data. Instead, state-of-the-art algorithms ▶post-prune a rule right after it has been learned. For this purpose, one third of the training data are reserved for pruning. After a rule has been learned, it is greedily simplified on the pruning set. Simplifications can be the deletion of the last condition, a final sequence of conditions, or an arbitrary condition of the rule. If the simplification does not decrease the accuracy of the rule on the pruning set, it will be performed. This so-called *incremental reduced error pruning* algorithm (Fürnkranz & Widmer, 1994) is used in the rule learning algorithm RIPPER (Cohen, 1995).

A survey and experimental comparison of pruning techniques for rule learning can be found in Fürnkranz (1997).

Alternatives to Covering

An obvious generalization of covering is to not entirely remove covered examples but to reduce their example ▶weights, thus decreasing their importance in subsequent iterations (see, e.g., the SLIPPER algorithm; Cohen & Singer 1999).

Rules can also be learned by alternative strategies. There have been numerous proposals, only the most influential can be mentioned. Each path from the root to a leaf of a ▶decision tree corresponds to a rule and so rules can be learned by first learning a decision tree and then post-processing it (see, e.g., the C4.5RULES algorithm; Quinlan, 1993). It is also possible to use the ▶APriori algorithm for an exhaustive search for classification rules, and to use a subsequent covering algorithm to combine the rules into a rule set (see, e.g., the CBA algorithm; Liu, Hsu, & Ma, 1998). RISE (Domingos, 1996) combines bottom-up generalization with ▶nearest neighbor algorithms to learn a theory via "conquering without separating".

Well-known Rule Learning Algorithms

AQ can be considered as the original covering algorithm. Its original version was conceived by Ryszard Michalski in the 1960s (Michalski, 1969), and numerous versions and variants of the algorithm appeared subsequently in the literature. AQ uses a top-down beam search for finding the best rule. It does not search all possible specializations of a rule, but only considers refinements that cover a particular example, the so-called *seed example*. This idea is basically the same as the use of a ▶bottom clause in ▶inductive logic programming.

CN2 (Clark & Niblett, 1989; Clark & Boswell, 1991) employs a beam search guided by the Laplace estimate, and uses the likelihood ratio significance test to fight overfitting. It can operate in two modes, one for learning rule sets (by modeling each class independently), and one for learning decision lists.

FOIL (Quinlan, 1990) was the first relational learning algorithm that received attention beyond the field of inductive logic programming. It learns a concept with the covering loop and learns individual concepts with a top-down refinement operator, guided by information gain. The main difference to previous systems is that FOIL allowed the use of first-order background knowledge. Instead of only being able to use tests on single attributes, FOIL could employ tests that compute relations between multiple attributes, as well as introduce new variables in the body of a rule.

RIPPER (Cohen, 1995) was the first rule learning system that effectively countered the overfitting problem via *incremental reduced error pruning*, as described above. It also added a post-processing phase for optimizing a rule set in the context of other rules. The key idea is to remove one rule out of a previously learned rule set and try to relearn it not only in the context of previous rules (as would be the case in the regular covering rule), but in the context of a complete theory. RIPPER is still state-of-the-art in inductive rule learning. A freely accessible reimplementation can be found in the WEKA machine learning library under the name of JRIP.

OPUS (Webb, 1995) was the first rule learning algorithm to demonstrate the feasibility of a full exhaustive search through all possible rule bodies for finding a rule that maximizes a given quality criterion (or heuristic function). The key idea is the use of *ordered search* that prevents that a rule is generated multiple times. This means that even though there are $l!$ different orders of the conditions of a rule of length l, only one of them can be taken by the learner for finding this rule. In addition, OPUS uses several techniques that prune significant parts of the search space, so that this search method becomes feasible. Follow-up work has shown that this technique is also an efficient alternative for ►association rule discovery, provided that the database to mine fits into the memory of the learning system.

Cross References

►Apriori Algorithm
►Association Rule
►Decision List
►Decision Trees
►Subgroup Discovery

Recommended Reading

Cestnik, B. (1990). Estimating probabilities: A crucial task in machine learning. In L. Aiello (Ed.), *Proceedings of the ninth European conference on artificial intelligence (ECAI-90), Stockholm, Sweden* (pp. 147–150). Pitman, London.

Clark, P., & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. In *Proceedings of the fifth European working session on learning (EWSL-91), Porto, Portugal* (pp. 151–163). London: Springer.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*(4), 261–283.

Cohen, W. W. (1995). Fast effective rule induction. In A. Prieditis & S. Russell (Eds.), *Proceedings of the 12th international conference on machine learning (ML-95), Lake Tahoe, California* (pp. 115–123). Morgan Kaufmann, San Mateo, CA.

Cohen, W. W., & Singer, Y. (1999). A simple, fast, and effective rule learner. In *Proceedings of the 16th national conference on artificial intelligence (AAAI-99), Orlando* (pp. 335–342). Menlo Park: AAAI/MIT Press.

Domingos, P. (1996). Unifying instance-based and rule-based induction. *Machine Learning, 24*, 141–168.

Fürnkranz, J. (1997). Pruning algorithms for rule learning. *Machine Learning, 27*(2), 139–171.

Fürnkranz, J. (February 1999). Separate-and-conquer rule learning. *Artificial Intelligence Review, 13*(1), 3–54.

Fürnkranz, J., & Flach, P. (2005). ROC 'n' rule learning – Towards a better understanding of covering algorithms. *Machine Learning, 58*(1), 39–77.

Fürnkranz, J., & Widmer, G. (1994). Incremental reduced error pruning. In W. Cohen & H. Hirsh (Eds.), *Proceedings of the 11th international conference on machine learning (ML-94), New Brunswick, NJ* (pp. 70–77). Morgan Kaufmann, San Mateo, CA.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. In R. Agrawal, P. Stolorz, & G. Piatetsky-Shapiro (Eds.), *Proceedings of the fourth international conference on knowledge discovery and data mining (KDD-98), New York City, NY* (pp. 80–86).

Michalski, R. S. (1969). On the quasi-minimal solution of the covering problem. In *Proceedings of the fifth international symposium on information processing (FCIP-69), Bled, Yugoslavia. Switching circuits* (Vol. A3, pp. 125–128).

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo: Morgan Kaufmann.

Webb, G. I. (1995). OPUS: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research, 5*, 431–465.

S

Sample Complexity

►Generalization Bounds

Samuel's Checkers Player

Definition

Samuel's Checkers Player is the first machine learning system that received public recognition. It pioneered many important ideas in game playing and machine learning. The two main papers describing his research (Samuel, 1959, 1967) became landmark papers in Artificial Intelligence. In one game, the resulting program was able to beat one of America's best players of the time.

Description of the Learning System

Samuel's checkers player featured a wide variety of learning techniques. First, his checkers player remembered positions that it frequently encountered during play. This simple form of *rote learning* allowed it to save time, and to search deeper in subsequent games whenever a stored position was encountered on the board or in some line of calculation. Next, it featured the first successful application of what is now known as ►Reinforcement Learning for tuning the weights of its evaluation function. The program trained itself by playing against a stable copy of itself. After each move, the weights of the evaluation function were adjusted in a way that moved the evaluation of the root position after a quiescence search closer to the evaluation of the root position after searching several moves deep. This technique is a variant of what is nowadays known as ►Temporal-Difference Learning and commonly used in successful game-playing programs. Samuel's program not only tuned the weights of the evaluation but also employed on-line ►Feature Selection for constructing the evaluation function with the terms that seem to be the most significant for evaluating the current board situation. ►Feature Construction was recognized as the key problem that still needs to be solved. Later, Samuel changed his evaluation function from a linear combination of terms into a structure that closely resembled a 3-layer ►Neural Network. This structure was trained with ►Preference Learning from several thousand positions from master games.

Cross References

►Machine Learning and Game Playing

Recommended Reading

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 3*(3), 211–229.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. II – recent progress. *IBM Journal of Research and Development, 11*(6), 601–617.

Saturation

►Bottom Clause

SDP

►Symbolic Dynamic Programming

Search Bias

►Learning as Search

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Search Engines: Applications of ML

Eric Martin
University of New South Wales,
Sydney, NSW, Australia

Definition

Search engines provide users with Internet resources – links to web sites, documents, text snippets, images, videos, etc. – in response to queries. They use techniques that are part of the field of information retrieval, and rely on statistical and pattern matching methods. Search engines have to take into account many key aspects and requirements of this specific instance of the information retrieval problem. First, the fact is that they have to be able to process hundreds of millions of searches a day and answer queries in a matter of milliseconds. Second, the resources on the World Wide Web are constantly updated, with information being continuously added, removed or changed – the overall contents changing by up to 8% a week – in a pool consisting of billions of documents. Third, the users express possibly semantically complex queries in a language with limited expressive power, and often not make use or proper use of available syntactic features of that language – for instance, the boolean *or* operator occurs in less than 3% of queries.

Motivation and Background

Web searching is technically initiated by sending a query to a search engine but the whole search process starts earlier, in the mind of the person who conducts the search. To be successful, the process needs to provide users with words, text snippets, images, or movies that fulfill the users' quest for information. Thus, though a search is technically the implementation of a procedure that maps a query to some digital material, it spans a larger spectrum of activities, from a psychological trigger to a psychological reward. For a given set of digital material that, if provided, would be deemed perfectly satisfactory by a number of users looking for the same information, different users will issue different queries. This might be because they have varying skills at conveying what they are after in the form of a few words. This, in turn, might be because their understanding of the technology prompts them to formulate what they are after in a form that, rightly or wrongly, they consider appropriate for a computing device to process. That might be for a number of different reasons that all point to the fact that the quality of the search is not determined by its adequacy to the query, but by its adequacy to the psychological trigger that produced the query. This especially makes Web searching challenging and exciting area in the field of ▶information retrieval.

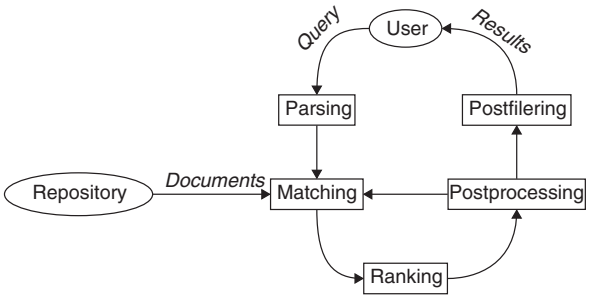
In Broder (2002), it is suggested that web queries can be classified in three classes:

- *Navigational queries* expect the search to return a particular url. For instance, http://www.cityrail.info is probably the expected result to the query `Cityrail` for a Sydneysider.
- *Transactional queries* expect the search to return links to sites that offer further interaction, for example for online shopping or to download music. For instance, http://www.magickeys.com/books/, where books for young children are available for download, is probably a good result to the query `children stories`.
- *Informational queries* expect the search to reveal the information, that is, the correct answer to a question. This information can be immediately provided in the page where the results of the search are displayed, for instance, `Bern` for the query `capital of switzerland`. Or it can be provided in the pages accessible from the first links returned by the search, as for instance `Italy` that is easily found in the web page accessed from the first link returned in response to the query `football world champion 1982`.

Answering an informational query with the information itself, rather than with links to documents where the information is to be found, is one of the most difficult challenges that search engines developers have started addressing.

Structure of the Learning System

The general structure of a search engine can be illustrated as follows:



A ▶string matching algorithm is applied to the parsed query issued by the user and to an indexed representation of a set of documents, resulting in a ranked subset of the latter. This ranked set of documents can be subjected to a postprocessing procedure whose aim is to improve the results by either refining the query or by analyzing the documents further, possibly over many iterations, until the results stabilize and can be returned to the user, following a postfiltering procedure to display the information appropriately.

Retrieval Methods

The existence of hyperlinks between documents distinguishes search engines from other information retrieval applications. All techniques developed in the field of information retrieval are potentially relevant for extracting information from the web, but benefits from a proper analysis of the cross-reference structure. That is, to measure the degree of relevance of a document to a given query, one can take advantage of a prior ranking of all documents independent of that query or any other, following a sophisticated version of the PageRank (Page, Brin, Motwani, & Winograd, 1999) link analysis algorithm. One of the simplest versions of the algorithm recursively defines the PageRank $\mathrm{PR}(T)$ of a page T which pages $T_1, \ldots, T_n$ point to, amongst the $c_1, \ldots, c_n$ pages $T_1, \ldots, T_n$ point to, respectively, as

$$\frac{1-d}{N} + d(T_1/c_1 + \cdots + T_n/c_n),$$

where N is the total number of pages and d, a *damping factor*, represents the probability that a user decides to follow a link rather than randomly visit another page; normalizing the solution so that the PageRanks of all pages add up to 1, $\mathrm{PR}(T)$ then represents the probability that a user visits T by clicking on a link.

Boolean retrieval is one of the simplest methods to retrieve a set of documents that match exactly a query expressed as a boolean combination of keywords. The match is facilitated by using an *inverted file* indexing structure which associates every possible keyword with links to the documents in which it occurs. If extra information is kept on the occurrences of keywords in documents (number of occurrences, part of the document in which they occur, font size and font type used for their display, etc.) then the results can also be ranked. But *best match* models, as opposed to *exact match* models, are better suited to producing ranked results. The *vector space* model is one of the earliest and most studied models of this kind. It represents documents and queries as vectors over a space each of whose dimensions represents a possible keyword, and measures the similarity between the vectors $\vec{q}$ and $\vec{d}$ whether it occurs at least once in query and document, respectively, record for each keyword as the cosine of the angle formed by $\vec{q}$ and $\vec{d}$, namely,

$$\frac{\vec{q}\vec{d}}{\|\vec{q}\|\|\vec{d}\|},$$

that is all the most closer to 1 that query and document have more in common. The *term-frequency-inverse-document-frequency* (tf-idf) model refines the encoding given by $\vec{d}$ by replacing a value of 1 in the ith dimension, indicating the existence of an occurrence of the ith keyword in $\vec{d}$, with

$$c_1 \log\left(\frac{N}{c_2}\right),$$

where c_1 is the number of occurrences of the ith keyword in the document, N is the total number of documents, and c_2 is the number of documents in the whole collection that contain at least one occurrence of the ith keyword; so more importance is given to keywords that occur more and that occur "almost exclusively" in the document under consideration. One of the most obvious issues with this approach is that the number of dimensions is huge and the vectors are sparse. Another important issue is that set of vectors determined by the set of keywords is not orthogonal, and not even linearly independent, because two given keywords can be synonyms (sick and ill), not semantically related (garlic and manifold), or more or less semantically related (wheel and tire).

The *extended vector space* model (Robertson, Walker, & Beaulieu, 1999b) addresses this issue assuming that the similarity between two keywords is captured by the symmetric difference between the set of documents that contain a keyword and the set of documents that contain the other, ranging from identical sets (similar keywords) to disjoint sets (unrelated keywords). Let $D_1, \ldots, D_{N'}$ be an enumeration of the quotient relation over the set of all documents such that two documents are equivalent if they contain precisely the same keywords (so N' is at most equal to N, the number of documents in the whole collection). Conceive of an N'-dimensional vector space S of which $D_1, \ldots, D_{N'}$ is a basis. Associate the ith keyword with the vector $\vec{v}_i$ of S defined as $1/\sqrt{w_1^2 + \cdots + w_{N'}^2}(w_1, \ldots, w_{N'})$ where for all nonzero $k \leq N'$, w_k is the number of occurrences of the ith keyword in all documents that belong to class D_k. Then associate a document with the vector $\vec{d}$ of S defined as $\alpha_1\vec{v}_1 + \cdots + \alpha_{N''}\vec{v}_{N''}$ where N'' is the number of keywords and for all nonzero $k \leq N''$, α_k is the number of occurrences of the ith keyword in that document, and associate a query with the vector $\vec{q}$ of S defined as $\beta_1\vec{v}_1 + \cdots + \beta_{N''}\vec{v}_{N''}$ where for all nonzero $k \leq N''$, β_k is equal to 1 if the ith keyword occurs in the query, and to 0 otherwise. The similarity between $\vec{q}$ and $\vec{d}$ is then measured as described for the simple vector space method.

The *topic-based vector space* model (Becker & Kuropka, 2003) also replaces the original vector space with a different vector space of a different dimension, addressing the issue of nonorthogonality between keywords thanks to *fundamental topics*, assumed to be pairwise independent, using ontologies; the fundamental topics then provide the vector basis which is a linear combination of a given keyword. So the topic-based vector space model conceives of the meaning of words as the semantic relationships that emerge from the common use of a language by the members of a given community, whereas the extended vector space model conceives of the meaning of words as the syntactic relationship of term co-occurrence with respect to the repository of documents being processed.

Probabilistic retrieval frameworks aim at estimating the probability that a given document is relevant to a given query. Given a keyword w, denote by p_w^+ the probability that w occurs in a document relevant to w, and denote by p_w^- the probability that w occurs in a document not relevant to w. Many probabilistic retrieval frameworks then define the relevance of a document to a query as follows, where $w_1, \ldots, w_n$ are the keywords that occur both in the query and in the document:

$$\sum_{i=1}^{n} \log\left(\frac{p_{w_i}^+(1-p_{w_i}^-)}{p_{w_i}^-(1-p_{w_i}^+)}\right).$$

This quantity increases all the more that the document contains more words that are more likely to occur in relevant documents, and more words less likely to occur in irrelevant documents. Different frameworks suggest different ways to evaluate the values of $p_{w_i}^+$ and $p_{w_i}^-$. For instance, p_i is sometimes assumed to be constant and $p_{w_i}^-$ defined as n_i/N where N is the total number of documents and n_i the number of documents in which w_i occurs, capturing the fact that a document containing a keyword appearing in few other documents is likely to be relevant to that keyword, in which case the previous formula can be rewritten

$$c\sum_{i=1}^{n} \log\left(\frac{N-n_i}{n_i}\right)$$

for some constant c. More sophisticated methods have been developed to better estimate the probabilities, such as the *Okapi weighting document score* (Robertson, Walker, & Beaulieu, 1999a) which defines the relevance of a document to a query as

$$\sum_{i=1}^{n} \log\left(\frac{N-n_i+0.5}{n_i+0.5}\right)\frac{(k_1+1)c_i}{(k_1(1-b)+b(l/\beta))+c_i} \times \frac{(k_3+1)d_i}{k_3+d_i},$$

where the notation is as above, with the addition of c_i to denote the number of occurrences of w_i in the document, d_i to denote the number of occurrences of w_i in the query, l to denote the number of bytes in the document, β to denote the average number of bytes in a document, and b, k_1, and k_3 to denote constants.

Query Classification

The development of effective methods of information retrieval from web resources requires a good understanding of users' needs and practice. In Markev (2007a), the following questions are identified as being especially relevant towards gaining such an understanding.

- What characterizes the queries that end users submit to online IR systems? What search features do people use? What features would enable them to improve on the retrievals they have in hand? What features are hardly ever used? What do end users do in response to the system's retrievals?

This chapter indicates that many of the basic features of information retrieval systems are poorly used. For instance, less than 15, 3, and 2% of queries make use of the *and*, *or*, and *not* boolean operators, respectively, and less than 15% of queries of enclosing quotes; the wrong syntax is often used, resulting in incorrect use of advanced search features in one third of the cases; less than 10% of queries take advantage of ►relevance feedback. Based on those findings, the second part (Markev, 2007b) of the article suggests *two dozen new research questions* for researchers in information retrieval, while noting that about 70% of users are satisfied with their search experience.

Evaluating search satisfaction has received lots of attention. In Fox, Karnawat, Mydland, Dumais, and White (2005), both explicit and implicit measures of satisfaction are collected. Explicit measures are obtained by prompting the user to evaluate a search result as satisfying, partially satisfying, or not satisfying, and similarly to evaluate satisfaction gained from a whole search session. Implicit measures are obtained by recording mouse and keyboard actions, time spent on a page, scrolling actions and durations, number of visits to a page, position of page in results list, number of queries submitted, number of results visited, etc. A Bayesian model can be used to infer the relationships between explicit and implicit measures of satisfaction. This chapter reports on two ►Bayesian networks that were built to predict satisfaction for individual page visits and satisfaction for entire search sessions – w.r.t. the feedback obtained from both kinds of prompts – with evidence that a combination of well chosen implicit satisfaction measures can be a good predictor of explicit satisfaction. Referring to the categorization of web queries in Broder (2002) as *user goals*, it is proposed in Lee, Liu, and Cho (2005) to build *click distributions* by sorting results to a query following the numbers of clicks they received from all users, and suggested that highly skewed distributions should correspond to navigational queries, while flat distributions should correspond to informational queries. The same kind of considerations are also applied to *anchor-link distributions*, the anchor-link distribution of a query being defined as the function that maps a URL to the number of times that URL is the destination of an anchor that has the same text as the query.

Finer techniques of query classification are proposed in Beitzel, Jensen, Lewis, Chowdhury, and Frieder (2007), where is a rule-based automatic classier is produced from *selectional preferences*. A query consisting of at least two keywords is split into a head x and a tail y, and then converted into a *forward* pair (x, u) and a *backward* pair (u, y), where u represents a category, that is, a generic term that refers to a list of semantically related words in a thesaurus. For instance, the query "interest rate" can (only) be split into (interest, rate) and converted to the forward pair (interest, personal finance) where "personal finance" denotes the list consisting of the terms "banks," "rates," "savings," etc.; so the first keyword – "interest" – provides context for the second one. Given a large query log, the *maximum likelihood estimate* (MLE) of $P(u/x)$, the probability that a query decomposed as (x, z) is such that z belongs to category u, is defined as the quotient between the number of queries in the log that have (x, u) as a forward pair and the number of queries in the log that can be decomposed as (x, z). This allows one to write a forward rule of the form "x Y classified as u with weight p," where p is the MLE of $P(u/x)$, provided that the *selectional preference strength* of x be above some given threshold. The rule can then be applied to incoming queries, such as "interest only loan" by matching a final or initial segment of the query – depending on whether forward or backward rules are under consideration – and suggest possible classifications; with the running example, "interest only loan" would then be classified as "personal finance with weight p" if a forward rule of the form "interest Y classified as personal finance with weight p" had been discovered. Such a classification can then be used to rewrite the query, or to send it to an appropriate database-backend if many domain-specific databases are available.

Cross References

►Bayesian Methods
►Classification

►Covariance Matrix
►Rule Learning
►Text Mining

Recommended Reading

Becker, J., & Kuropka, D. (2003). Topic-based vector space model. In W. Abramowicz & G. Klein (Eds.), *Proceedings of the sixth international conference on business information systems* (pp. 7–12). Colorado Springs, CO.

Beitzel, S. M., Jensen, E. C., Lewis, D. D., Chowdhury, A., & Frieder, O. (2007). Automatic classification of web queries using very large unlabeled query logs. *ACM Transactions on Information Systems, 25*(2), 9. ISSN: 1046-8188

Broder, A. (2002). A taxonomy of web search. *SIGIR Forum, 36*(2), 3–10.

Fox, S., Karnawat, K., Mydland, M., Dumais, S., & White, T. (2005). Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems, 23*(2), 147–168.

Lee, U., Liu, Z., & Cho, J. (2005). Automatic identification of user goals in web search. In *WWW '05*: In *Proceedings of the 14th international conference on World Wide Web* (pp. 391–400). New York: ACM Press.

Markev, K. (2007a). Twenty-five years of end-user searching, part 1: Research findings. *Journal of the American Society for Information Science and Technology, 58*(8), 1071–1081.

Markev, K. (2007b). Twenty-five years of end-user searching, part 2: Future research directions. *Journal of the American Society for Information Science and Technology, 58*(8), 1123–1130.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). *The pagerank citation ranking: Bringing order to the web*. Technical report. Stanford, CA: Stanford University Press.

Robertson, S. E., Walker, S., & Beaulieu, M. (1999a). Okapi at trec-7: Automatic ad hoc, filtering, VLC and filtering tracks. In E. Voorhees & D. Harman (Eds.), In *Proceedings of the seventh text retrieval conference* (pp. 253–264). Gaithersburg, MD.

Robertson, S. E., Walker, S., & Beaulieu, M. (1999b). On modeling of information retrieval concepts in vector spaces. *ACM Transactions on Database Systems, 12*(2), 299–321.

Self-Organizing Feature Maps

►Self-Organizing Maps

Self-Organizing Maps

Samuel Kaski
Helsinki University of Technology, Finland

Synonyms

Kohonen maps; Self-organizing feature maps; SOM

Definition

Self-organizing map (SOM), or Kohonen Map, is a computational data analysis method which produces nonlinear mappings of data to lower dimensions. Alternatively, the SOM can be viewed as a ►clustering algorithm which produces a set of clusters organized on a regular grid. The roots of SOM are in neural computation (see ►neural networks); it has been used as an abstract model for the formation of ordered maps of brain functions, such as sensory feature maps. Several variants have been proposed, ranging from dynamic models to Bayesian variants. The SOM has been used widely as an engineering tool for data analysis, process monitoring, and information visualization, in numerous application areas.

Motivation and Background

The SOM (Kohonen, 1982, 2001) was originally introduced in the context of modeling of how the spatial organization of brain functions forms. Formation of feature detectors selective to certain sensory inputs, such as orientation-selective visual neurons, had earlier been modeled by ►competitive learning in neural networks, and some models of how the feature detectors become spatially ordered had been published (von der Malsburg, 1973). The SOM introduced an *adaptation kernel* or *neighborhood function* that governs the adaptation in such networks; while in plain competitive learning only the winning neuron that best matches the inputs adapts, in SOM all neurons within a local neighborhood of the winner learn. The neighborhood is determined by the neighborhood function. The SOM is an algorithm for computing such ordered mappings.

While some of the motivation of the SOM comes from neural computation, its main uses have been as a practical data analysis method. The SOM can be viewed as a topographic vector quantizer, a nonlinear projection method, or a clustering method. In particular, it is a clustering-type algorithm that orders the clusters. Alternatively, it is a nonlinear projection-type algorithm that clusters, or more specifically quantizes, the data.

The SOM was very popular in the 1990s and still is; it is intuitively relatively easily understandable, yet hard to analyze thoroughly. It connects many research traditions and works well in practice. An impressive set of variants have been published over the years, of which

probabilistic variants (e.g., Bishop, Svensén, & Williams (1998) and Heskes (2001)) are perhaps closest to the current mainstream machine learning. While there currently are excellent alternative choices for many of the specific tasks SOMs have been applied for over the years, even the basic SOM algorithm is still viable as a versatile engineering tool in data-analysis tasks.

Structure of Learning System

The SOM consists of a regular grid of nodes (Fig. 1). A *model* of data has been attached to each node. For vector-valued data $\mathbf{x} = [x_1, \ldots, x_d]^T$, the models are vectors in the same space; the model at the ith node is $\mathbf{m}_i = [m_{i1}, \ldots, m_{id}]$. The models define a mapping from the grid to the data space. The coordinates on the grid are uniquely determined by the index i of a node, and the model $\mathbf{m}_i$ gives the location in the data space. The whole grid becomes mapped into an "elastic net" in the data space. While being a mapping from the grid to the input space, the SOM defines a projection from the input space to the discrete grid locations as well; each data point is projected to the node having the closest model.

The original online SOM algorithm updates the model vectors toward the current input vector at time t,

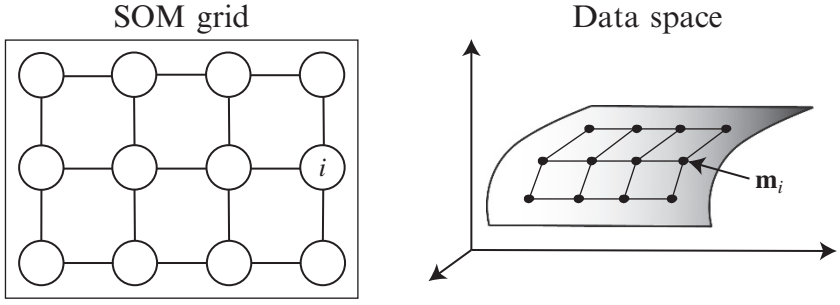
$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}(t)(\mathbf{x}(t) - \mathbf{m}_i(t)) \; .$$

Here c is the index of the unit having the closest model vector to $\mathbf{x}(t)$, and $h_{ci}(t)$ is the neighborhood function or adaptation kernel. The kernel is a decreasing function of the distance between the units i and c on the grid; it forces neighboring units to adapt toward similar input samples. The height and width of h are decreasing functions of time t. In an iteration over time and over the different inputs, the model vectors become ordered and specialize to represent different regions of the input space.

The online version of ▶K-means clustering is a special case of the SOM learning rule, where only the closest model vector is adapted. That is, the neighborhood function is $h_{ci}(t) = \alpha(t)$ for $i = c$ and $h_{ci} = 0$ otherwise. Here $\alpha(t)$ is the adaptation coefficient, a decreasing scalar. In short, K-means and SOM use the prototypes in the same way, but in SOM the prototypes have an inherent order that stems from fixing them onto a grid and updating the prototypes to represent both the data mapped to themselves and to their neighbors.

A neural interpretation of the SOM adaptation process is that the nodes are feature detector neurons or processing modules that in a ▶competitive learning process become specialized to represent different kinds of inputs. The neighborhood function is a plasticity kernel that forces neighboring neurons to adapt at the same time. The kernel transforms the discrete set of feature detectors into feature maps analogous to ordered brain maps of sensor inputs, and more generally to maps of more abstract properties of the input data.

A third interpretation of the SOM is as a vector quantizer. The task of a vector quantizer is to encode inputs with indexes of prototypes, often called codebook vectors, such that a distortion measure is minimized. If there is noise that may change the indexes, the



Self-Organizing Maps. Figure 1. A schematic diagram showing how the SOM grid of units (circles on the left, neighbors connected with lines) corresponds to an "elastic net" in the data space. The mapping from the grid locations, determined by the indices *i*, to the data space is given by the model vectors $\mathbf{m}_i$ attached to the units *i*

distribution of the noise should be used as the neighborhood function, and then the distortion becomes minimized by a variant of SOM (Luttrell, 1994). In summary, the SOM can be viewed as an algorithm for producing codebooks ordered on a grid.

While it has turned out to be hard to rigorously analyze the properties of the SOM algorithm (Fort, 2006), its fixed points may be informative. In a fixed point the models must fulfill

$$m_i = \frac{\sum_{\mathbf{x}} h_{c(\mathbf{x}),i}\mathbf{x}}{\sum_{\mathbf{x}} h_{c(\mathbf{x}),i}},$$

that is, each model vector is in the centroid of data projected to it and its neighbors. The definition of a *principal curve* (Hastie, Tibshirani, & Friedman, 2001), a nonlinear generalization of principal components (see ►principle components analysis), essentially is that the curve goes through the centroid of data projected to it. Hence, one interpretation of the SOM is a discretized, smoothed, nonlinear generalization of principal components. In short, SOMs aim to describe the variation in the data nonlinearly with their discrete grids.

Finally, a popular prototype-based classifier, ►learning vector quantization (LVQ) (Kohonen, 2001), can be loosely interpreted as a variant of SOMs, although it does not have the neighborhood function and hence, the prototypes do not have an order.

Programs and Data

The SOM has been implemented in several commercial packages and as freeware. Two examples, SOM_PAK written in C and Matlab SOM Toolbox (http://www.cis.hut.fi/research/software) came from Kohonen's group.

Applications

The SOM can be used as a nonlinear dimensionality reduction method, by projecting each data vector into the grid location having the closest model vector. An image of the grid can be used for *information visualization*. Since all grid locations are clusters, the SOM display actually visualizes an ordered set of clusters, or a quantized image of the principal manifold in data. More specifically, the SOM units can be thought of as subclusters, and data clusters may form larger areas on the SOM grid.

SOM-based visualizations can be used for illustrating the proximity relationships of data vectors, such as documents in the WEBSOM document maps (Kohonen et al., 2000), or monitoring the change of a system such as an industrial process or the utterances of a speaker, as a trajectory on the SOM display. More applications can be found in a collected bibliography (the latest one is Pöllä, Honkela, & Kohonen (in press)).

Cross References

►ART
►Competitive Learning
►Dimensionality Reduction
►Hebbian Learning
►K-means Clustering
►Learning Vector Quantization

Recommended Reading

Bishop, C. M., Svensén, M., & Williams, C. K. I. (1998). GTM: The generative topographic mapping. *Neural Computation, 10*, 215–234.

Fort, J. C. (2006). SOM's mathematics. *Neural Networks, 19*, 812–816.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. New York: Springer.

Heskes, T. (2001). Self-organizing maps, vector quantization, and mixture modeling. *IEEE Transactions on Neural Networks, 12*, 1299–1305.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics, 43*, 59–69.

Kohonen, T. (2001). *Self-organizing maps* (3rd ed.). Berlin: Springer.

Kohonen, T., Kaski, S., Lagus, K., Salojärvi, J., Honkela, J., Paatero, V., et al. (2000). Self organization of a massive document collection. *IEEE Transactions on Neural Networks, 11*, 574–585.

Luttrell, S. P. (1994). A Bayesian analysis of self-organizing maps. *Neural Computation, 6*, 767–794.

Pöllä, M., Honkela, T., & Kohonen, T. (2009). Bibliography of self-organizing map (SOM) papers: 2002-2005 addendum. Report TKK-ICS-R23, Helsinki University of Technology, Department of Information and Computer Science, Espoo, Finland.

von der Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik, 14*, 85–100.

Semantic Mapping

►Text Visualization

Semi-Naive Bayesian Learning

Fei Zheng, Geoffrey I. Webb
Monash University,
Clayton, Melbourne,
Victoria, Australia

Definition

Semi-naive Bayesian learning refers to a field of ►Supervised Classification that seeks to enhance the classification and conditional probability estimation accuracy of ►naive Bayes by relaxing its attribute independence assumption.

Motivation and Background

The assumption underlying ►naive Bayes is that attributes are independent of each other, given the class. This is an unrealistic assumption for many applications. Violations of this assumption can render naive Bayes' classification suboptimal. There have been many attempts to improve the classification accuracy and probability estimation of naive Bayes by relaxing the attribute independence assumption while at the same time retaining much of its simplicity and efficiency.

Taxonomy of Semi-Naive Bayesian Techniques

Semi-naive Bayesian methods can be roughly subdivided into five high-level strategies for relaxing the independence assumption.

- The first strategy forms an attribute subset by deleting attributes to remove harmful interdependencies and applies conventional naive Bayes to this attribute subset.
- The second strategy modifies naive Bayes by adding explicit interdependencies between attributes.
- The third strategy accommodates violations of the attribute independence assumption by applying naive Bayes to a subset of training set. Note that the second and third strategies are not mutually exclusive.
- The fourth strategy performs adjustments to the output of naive Bayes without altering its direct operation.
- The fifth strategy introduces hidden variables to naive Bayes.

Methods That Apply Naive Bayes to a Subset of Attributes

Due to the attribute independence assumption, the accuracy of naive Bayes is often degraded by the presence of strongly correlated attributes. Irrelevant attributes may also degrade the accuracy of naive Bayes, in effect increasing variance without decreasing bias. Hence, it is useful to remove both strongly correlated and irrelevant attributes.

Backward sequential elimination (Kittler, 1986) is an effective wrapper technique to select an attribute subset and has been profitably applied to naive Bayes. It begins with the complete attribute set and iteratively removes successive attributes. On each iteration, naive Bayes is applied to every subset of attributes that can be formed by removing one further attribute. The attribute whose deletion most improves training set accuracy is then removed, and the process repeated. It terminates the process when subsequent attribute deletion does not improve training set accuracy. Conventional naive Bayes is then applied to the resulting attribute subset.

One extreme type of interdependencies between attributes results in a value of one being a generalization of a value of the other. For example, *Gender=female* is a generalization of *Pregnant=yes*. Subsumption resolution (SR) (Zheng & Webb, 2006) identifies at classification time pairs of attribute values such that one appears to subsume (be a generalization of) the other and delete the generalization. It uses the criterion $|T_{x_i}| = |T_{x_i,x_j}| \geq u$ to infer that attribute value x_j is a generalization of attribute value x_i, where $|T_{x_i}|$ is the number of training cases with value x_i, $|T_{x_i,x_j}|$ is the number of training cases with both values, and u is a user-specified minimum frequency. When SR is applied to naive Bayes, the resulting classifier acts as naive Bayes except that it deletes generalization attribute-values at classification time if a specialization is detected.

Methods That Alter Naive Bayes by Allowing Interdependencies between Attributes

Interdependencies between attributes can be addressed directly by allowing an attribute to depend on other non-class attributes. Sahami (1996) introduces the terminology of the z-dependence Bayesian classifier, in which each attribute depends upon the class and at most z other attributes. Figure 1 depicts methods in this group from the ►Bayesian Network perspective.

In Fig. 1a, each attribute depends on the class and at most one another attribute. ►Tree Augmented Naive Bayes (TAN) (Friedman, Geiger, & Goldszmidt, 1997) is a representative one-dependence classifier. It efficiently finds a directed spanning tree by maximizing the log-likelihood and employs this tree to perform classification. SuperParent TAN (Keogh & Pazzani, 1999) is an effective variant of TAN.

A SuperParent one-dependence classifier (Fig. 1b) is a special case of one-dependence classifiers, in which an attribute called the SuperParent (X_1 in this graph), is selected as the parent of all the other attributes. ►Averaged One-Dependence Estimators (AODE) (Webb, Boughton, & Wang, 2005) selects a restricted class of one-dependence classifiers and aggregates the predictions of all qualified classifiers within this class. Maximum a posteriori linear mixture of generative distributions (MAPLMG) (Cerquides & Mántaras, 2005) extends AODE by assigning a weight to each one-dependence classifier.

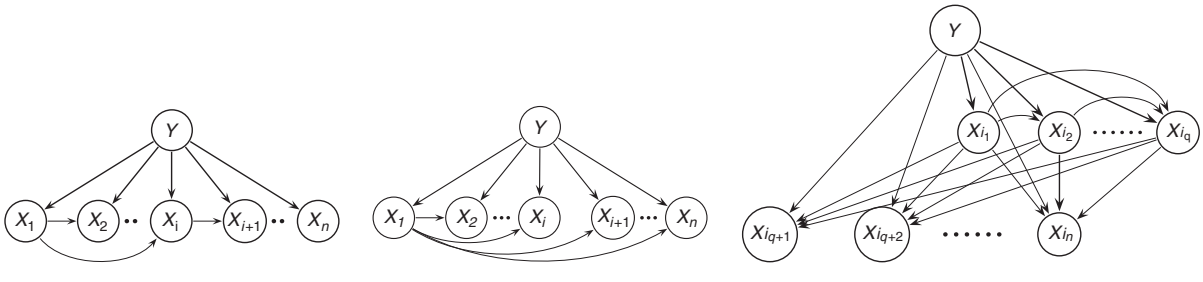
Two z-dependence classifiers ($z \geq 0$) are NBTree (Kohavi, 1996) and lazy Bayesian rules (LBR) (Zheng & Webb, 2000), both of which may add any number of non-class-parents for an attribute. In Fig. 1c, attributes in $\{X_{i_{q+1}}, \ldots, X_{i_n}\}$ depend on all the attributes in $\{X_{i_1}, \ldots, X_{i_q}\}$. The main difference between these two methods is that NBTree builds a single tree for all training instances while LBR generates a Bayesian rule for each test instance.

Methods That Apply Naive Bayes to a Subset of the Training Set

Another effective approach to accommodating violations of the conditional independence assumption is to apply naive Bayes to a subset of the training set, as it is possible that the assumption, although violated in the whole training set, may hold or approximately hold in a subset of the training set. NBTree and LBR use a local naive Bayes to classify an instance and can also be classified into this group. Locally weighted naive Bayes (LWNB) (Frank, Hall, & Pfahringer, 2003) applies naive Bayes to a neighborhood of the test instance, in which each instance is assigned a weight decreasing linearly with the Euclidean distance to the test instance. The number of instances in the subset is determined by a user-specified parameter. Only those instances whose weights are greater than zero are used for classification.

Methods That Calibrate Naive Bayes' Probability Estimates

Methods in this group make adjustments to the distortion in estimated posterior probabilities resulting from violations of independence assumption. Isotonic regression (IR) (Zadrozny & Elkan, 2002) is a nonparametric calibration method which produces a monotonically increasing transformation of the probability outcome



Semi-Naive Bayesian Learning. Figure 1. Bayesian Network. (a) one-dependence classifier, (b) SuperParent one-dependence classifier and (c) z-dependence classifier ($z \geq 0$)

of naive Bayes. It uses a pair-adjacent violators algorithm (Ayer, Brunk, Ewing, Reid, & Silverman, 1955) to perform calibration. To classify a test instance, IR first finds the interval in which the estimated posterior probability fits and predicts the isotonic regression estimate of this interval as the calibrated posterior probability. Adjusted probability naive Bayesian classification (Webb & Pazzani, 1998) makes adjustments to class probabilities, using a simple hill-climbing search to find adjustments that maximize the ►leave-one-out cross validation accuracy estimate. Starting with the conditional attribute-value frequency table generated by naive Bayes, iterative Bayes (Gama, 2003) iteratively updates the frequency table by cycling through all training instances.

Methods That Introduce Hidden Variables to Naive Bayes

Creating hidden variables or joining attributes is another effective approach to relaxing the attribute independence assumption. Backward sequential elimination and joining (BSEJ) (Pazzani, 1996) extends BSE by creating new Cartesian product attributes. It considers joining each pair of attributes and creates new Cartesian product attributes if the action improves leave-one-out cross validation accuracy. It deletes original attributes and also new Cartesian product attributes during a hill-climbing search. This process of joining or deleting is repeated until there is no further accuracy improvement. Hierarchical naive Bayes (Zhang, Nielsen, & Jensen, 2004) uses conditional mutual information as a criterion to create a hidden variable whose value set is initialized to the Cartesian product over all the value sets of its children. Values of a hidden variable are then collapsed by maximizing conditional log-likelihood via the ►minimum description length principle (Rissanen, 1978).

Selection Between Semi-Naive Bayesian Methods

No algorithm is universally optimal in terms of generalization accuracy. General recommendations for selection between semi-naive Bayesian methods is provided based on ►bias-variance tradeoff together with characteristics of the application to which they are applied.

Error can be decomposed into bias and variance (see ►bias variance decomposition). Bias measures how closely a learner is able to approximate the decision surfaces for a domain and variance measures the sensitivity of a learner to random variations in the training data. Unfortunately, we cannot, in general, minimize bias and variance simultaneously. There is a bias-variance tradeoff such that bias typically decreases when variance increases and vice versa. Data set size usually interacts with bias and variance and in turn affects error. Since differences between samples are expected to decrease with increasing sample size, differences between models formed from those samples are expected to decrease and hence variance is expected to decrease. Therefore, the bias proportion of error may be higher on large data sets than on small data sets and the variance proportion of error may be higher on small data sets than on large data sets. Consequently, low bias algorithms may have advantage in error on large data sets and low variance algorithms may have advantage in error on small data sets (Brain & Webb, 2002).

Zheng & Webb (2005) compare eight semi-naive Bayesian methods with naive Bayes. These methods are BSE, FSS, TAN, SP-TAN, AODE, NBTree, LBR, and BSEJ. NBTree, SP-TAN, and BSEJ have relatively high training time complexity, while LBR has high classification time complexity. BSEJ has very high space complexity. NBTree and BSEJ have very low bias and high variance. Naive Bayes and AODE have very low variance. AODE has a significant advantage in error over other semi-naive Bayesian algorithms tested, with the exceptions of LBR and SP-TAN. It achieves a lower error for more data sets than LBR and SP-TAN without SP-TAN's high training time complexity and LBR's high test time complexity. Subsequent researches (Cerquides & Mántaras, 2005; Zheng & Webb, 2006) show that MAPLMG and SR can in practice significantly improve both classification accuracy and the precision of conditional probability estimates of AODE. However, MAPLMG imposes very high training time overheads on AODE, while SR imposes no extra training time overheads and only modest test time overheads on AODE.

Within the prevailing computational complexity constraints, we suggest using the lowest bias semi-naive Bayesian method for large training data and lowest variance semi-naive Bayesian method for small training

data. An appropriate tradeoff between bias and variance should be sought for intermediate size training data. For extremely small data, naive Bayes may be superior and for large data NBTree and BSEJ may be more appealing options if their computational complexity satisfies the computational constraints of the application context. AODE achieves very low variance, relatively low bias and low training time and space complexity. MAPLMG and SR further enhance AODE by substantially reducing bias and error and improving probability prediction with modest time complexity. Consequently, they may prove competitive over a considerable range of classification tasks. Furthermore, MAPLMG may excel if the primary consideration is attaining the highest possible classification accuracy and SR may have an advantage if one wishes efficient classification.

Cross References

►Bayesian Network
►Naive Bayes

Recommended Reading

Ayer, M., Brunk, H. D., Ewing, G. M., Reid, W. T., & Silverman, E. (1955). An empirical distribution function for sampling with incomplete information. *The Annals of Mathematical Statistics, 26*(4), 641–647.

Brain, D., & Webb, G. I. (2002). The need for low bias algorithms in classification learning from large data sets. In *Proceedings of the Sixteenth European Conference on Principles of Data Mining and Knowledge Discovery* (pp. 62–73). Berlin: Springer-Verlag.

Cerquides, J., & Mántaras, R. L. D. (2005). Robust Bayesian linear classifier ensembles. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pp. 70–81.

Frank, E., Hall, M., & Pfahringer, B. (2003). Locally weighted naive Bayes. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, Acapulco, Mexico (pp. 249–256). San Francisco, CA: Morgan Kaufmann.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning, 29*(2), 131–163.

Gama, J. (2003). Iterative Bayes. *Theoretical Computer Science, 292*(2), 417–430.

Keogh, E. J., & Pazzani, M. J. (1999). Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, pp. 225–230.

Kittler, J., (1986). Feature selection and extraction. In T. Y. Young & K. S. Fu (Eds.), *Handbook of Pattern Recognition and Image Processing*. New York: Academic Press.

Kohavi, R. (1996). Scaling up the accuracy of naive-Bayes classifiers: A decisiontree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 202–207.

Pazzani, M. J. (1996). Constructive induction of Cartesian product attributes. In *ISIS: Information. Statistics and Induction in Science*, Melbourne, Australia, (pp. 66–77). Singapore: World Scientific.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica, 14*, 465–471.

Sahami, M. (1996). Learning limited dependence Bayesian classifiers. In *Proceedings of the Second International Conference on Knowledge Discovery in Databases* (pp. 334–338) Menlo Park: AAAI Press.

Webb, G. I., & Pazzani, M. J. (1998). Adjusted probability naive Bayesian induction. In *Proceedings of the Eleventh Australian Joint Conference on Artificial Intelligence*, Sydney, Australia (pp. 285–295). Berlin: Springer.

Webb, G. I., Boughton, J., & Wang, Z. (2005). Not so naive Bayes: Aggregating onedependence estimators. *Machine Learning, 58*(1), 5–24.

Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada (pp. 694–699). New York: ACM Press.

Zhang, N. L., Nielsen, T. D., & Jensen, F. V. (2004). Latent variable discovery in classification models. *Artificial Intelligence in Medicine, 30*(3), 283–299.

Zheng, Z., & Webb, G. I. (2000). Lazy learning of Bayesian rules. *Machine Learning, 41*(1), 53–84.

Zheng, F., & Webb, G. I. (2005). A comparative study of semi-naive Bayes methods in classification learning. In *Proceedings of the Fourth Australasian Data Mining Conference*, Sydney, pp. 141–156.

Zheng, F., & Webb, G. I. (2006). Efficient lazy elimination for averaged-one dependence estimators. In *Proceedings of the Twenty-third International Conference on Machine Learning* (pp. 1113–1120). New York: ACM Press.

Semi-Supervised Learning

Xiaojin Zhu
University of Wisconsin-Madison,
Madison, WI, USA

Synonyms

Co-training; Learning from labeled and unlabeled data; Transductive learning

Definition

Semi-supervised learning uses both labeled and unlabeled data to perform an otherwise ►supervised learning or ►unsupervised learning task.

In the former case, there is a distinction between inductive semi-supervised learning and transductive

learning. In inductive semi-supervised learning, the learner has both labeled training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{l} \overset{iid}{\sim} p(\mathbf{x}, y)$ and unlabeled training data $\{\mathbf{x}_i\}_{i=l+1}^{l+u} \overset{iid}{\sim} p(\mathbf{x})$, and learns a predictor $f : \mathcal{X} \mapsto \mathcal{Y}, f \in \mathcal{F}$ where $\mathcal{F}$ is the hypothesis space. Here $\mathbf{x} \in \mathcal{X}$ is an input instance, $y \in \mathcal{Y}$ its target label (discrete for ►classification or continuous for ►regression), $p(\mathbf{x}, y)$ the unknown joint distribution and $p(\mathbf{x})$ its marginal, and typically $l \ll u$. The goal is to learn a predictor that predicts future test data better than the predictor learned from the labeled training data alone. In transductive learning, the setting is the same except that one is solely interested in the predictions on the unlabeled training data $\{\mathbf{x}_i\}_{i=l+1}^{l+u}$, without any intention to generalize to future test data.

In the latter case, an unsupervised learning task is enhanced by labeled data. For example, in semi-supervised clustering (a.k.a. ►constrained clustering) one may have a few must-links (two instances must be in the same cluster) and cannot-links (two instances cannot be in the same cluster) in addition to the unlabeled instances to be clustered; in semi-supervised ►dimensionality reduction one might have the target low-dimensional coordinates on a few instances.

This entry will focus on the former case of learning a predictor.

Motivation and Background

Semi-supervised learning is initially motivated by its practical value in learning faster, better, and cheaper. In many real world applications, it is relatively easy to acquire a large amount of unlabeled data $\{\mathbf{x}\}$. For example, documents can be crawled from the Web, images can be obtained from surveillance cameras, and speech can be collected from broadcast. However, their corresponding labels $\{y\}$ for the prediction task, such as sentiment orientation, intrusion detection, and phonetic transcript, often requires slow human annotation and expensive laboratory experiments. This labeling bottleneck results in a scarce of labeled data and a surplus of unlabeled data. Therefore, being able to utilize the surplus unlabeled data is desirable.

Recently, semi-supervised learning also finds applications in cognitive psychology as a computational model for human learning. In human categorization and concept forming, the environment provides unsupervised data (e.g., a child watching surrounding objects by herself) in addition to labeled data from a teacher (e.g., Dad points to an object and says "bird!"). There is evidence that human beings can combine labeled and unlabeled data to facilitate learning.

The history of semi-supervised learning goes back to at least the 1970s, when self-training, transduction, and Gaussian mixtures with the expectation-maximization (EM) algorithm first emerged. It enjoyed an explosion of interest since the 1990s, with the development of new algorithms like co-training and transductive support vector machines, new applications in natural language processing and computer vision, and new theoretical analyses. More discussions can be found in section 1.1.3 in Chapelle, Zien, and Schölkopf (2006).

Theory

Unlabeled data $\{\mathbf{x}_i\}_{i=l+1}^{l+u}$ by itself does not carry any information on the mapping $\mathcal{X} \mapsto \mathcal{Y}$. How can it help us learn a better predictor $f : \mathcal{X} \mapsto \mathcal{Y}$? Balcan and Blum pointed out in 2009 that the key lies in an implicit ordering of $f \in \mathcal{F}$ induced by the unlabeled data. Informally, if the implicit ordering happens to rank the target predictor f^* near the top, then one needs less labeled data to learn f^*. This idea will be formalized later on using PAC learning bounds. In other contexts, the implicit ordering is interpreted as a prior over $\mathcal{F}$ or as a regularizer.

A semi-supervised learning method must address two questions: what implicit ordering is induced by the unlabeled data, and how to algorithmically find a predictor near the top of this implicit ordering and fits the labeled data well. Many semi-supervised learning methods have been proposed, with different answers to these two questions (Abney, 2007; Chapelle et al., 2006; Seeger, 2001; Zhu & Goldberg, 2009). It is impossible to enumerate all methods in this entry. Instead, we present a few representative methods.

Generative Models

This semi-supervised learning method assumes the form of joint probability $p(\mathbf{x}, y \mid \theta) = p(y \mid \theta)p(\mathbf{x} \mid y, \theta)$. For example, the class prior distribution $p(y \mid \theta)$ can be a multinomial over $\mathcal{Y}$, while the class conditional distribution $p(\mathbf{x} \mid y, \theta)$ can be a multivariate Gaussian in $\mathcal{X}$ (Castelli & Cover, 1995; Nigam, McCallum, Thrun, & Mitchell, 2000). We use $\theta \in \Theta$ to denote the

parameters of the joint probability. Each θ corresponds to a predictor f_θ via Bayes rule:

$$f_\theta(\mathbf{x}) \equiv \text{argmax}_y p(y \mid \mathbf{x}, \theta) = \text{argmax}_y \frac{p(\mathbf{x}, y \mid \theta)}{\sum_{y'} p(\mathbf{x}, y' \mid \theta)}.$$

Therefore, $\mathcal{F} = \{f_\theta : \theta \in \Theta\}$.

What is the implicit ordering of f_θ induced by unlabeled training data $\{\mathbf{x}_i\}_{i=l+1}^{l+u}$? It is the large to small ordering of log likelihood of θ on unlabeled data:

$$\log p(\{\mathbf{x}_i\}_{i=l+1}^{l+u} \mid \theta) = \sum_{i=l+1}^{l+u} \log \left(\sum_{y \in \mathcal{Y}} p(\mathbf{x}_i, y \mid \theta) \right).$$

The top ranked f_θ is the one whose θ (or rather the generative model with parameters θ) best fits the unlabeled data. Therefore, this method assumes that the form of the joint probability is correct for the task.

To identify the f_θ that both fits the labeled data well and ranks high, one maximizes the log likelihood of θ on both labeled and unlabeled data:

$$\text{argmax}_\theta \log p(\{\mathbf{x}_i, y_i\}_{i=1}^{l} \mid \theta) + \lambda \log p(\{\mathbf{x}_i\}_{i=l+1}^{l+u} \mid \theta),$$

where λ is a balancing weight. This is a non-concave problem. A local maximum can be found with the EM algorithm, or other numerical optimization methods. (See also, ▶generative learning.)

Semi-Supervised Support Vector Machines

This semi-supervised learning method assumes that the decision boundary $f(\mathbf{x}) = 0$ is situated in a low-density region (in terms of unlabeled data) between the two classes $y \in \{-1, 1\}$ (Joachims, 1999; Vapnik, 1998). Consider the following hat loss function on an unlabeled instance $\mathbf{x}$:

$$\max(1 - |f(\mathbf{x})|, 0),$$

which is positive when $-1 < f(\mathbf{x}) < 1$, and zero outside. The hat loss thus measures the violation in (unlabeled) large margin separation between f and $\mathbf{x}$. Averaging over all unlabeled training instances, it induces an implicit ordering from small to large over $f \in \mathcal{F}$:

$$\frac{1}{u} \sum_{i=l+1}^{l+u} \max(1 - |f(\mathbf{x})|, 0).$$

The top ranked f is one whose decision boundary avoids most unlabeled instances by a large margin.

To find the f that both fits the labeled data well and ranks high, one typically minimizes the following objective:

$$\text{argmin}_f \frac{1}{l} \sum_{i=1}^{l} \max(1 - y_i f(\mathbf{x}_i), 0) + \lambda_1 \|f\|^2 + \lambda_2 \frac{1}{u} \sum_{i=l+1}^{l+u} \max(1 - |f(\mathbf{x})|, 0),$$

which is a combination of the objective for supervised support vector machines, and the average hat loss. Algorithmically, the optimization problem is difficult because the hat loss is non-convex. Existing solutions include semi-definite programming relaxation, deterministic annealing, continuation method, concave-convex procedure (CCCP), stochastic gradient descent, and Branch and Bound. (See also ▶support vector machines.)

Graph-Based Models

This semi-supervised learning method assumes that there is a graph $G = \{V, E\}$ such that the vertices V are the labeled and unlabeled training instances, and the undirected edges E connect instances i, j with weight w_{ij} (Belkin, Niyogi, & Sindhwani, 2006; Blum & Chawla, 2001; Zhu, Ghahramani, & Lafferty, 2003). The graph is sometimes assumed to be a random instantiation of an underlying manifold structure that supports $p(\mathbf{x})$. Typically, w_{ij} reflects the proximity of $\mathbf{x}_i, \mathbf{x}_j$. For example, the Gaussian edge weight function defines $w_{ij} = \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\sigma^2\right)$. As another example, the kNN edge weight function defines $w_{ij} = 1$ if $\mathbf{x}_i$ is within the k nearest neighbors of $\mathbf{x}_j$ or vice versa, and $w_{ij} = 0$ otherwise. Other commonly used edge weight functions include ϵ-radius neighbors, b-matching, and combinations of the above.

Large w_{ij} implies a preference for the predictions $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$ to be the same. This can be formalized by the graph energy of a function f:

$$\sum_{i,j=1}^{l+u} w_{ij} (f(\mathbf{x}_i) - f(\mathbf{x}_j))^2.$$

The graph energy induces an implicit ordering of $f \in \mathcal{F}$ from small to large. The top ranked function is the smoothest with respect to the graph (in fact, it is any constant function). The graph energy can be

equivalently expressed using the so-called unnormalized graph Laplacian matrix. Variants including the normalized Laplacian and the powers of these matrices.

To find the f that both fits the labeled data well and ranks high (i.e., being smooth on the graph or manifold), one typically minimizes the following objective:

$$\text{argmin}_f \frac{1}{l}\sum_{i=1}^{l} c(f(\mathbf{x}_i), y_i) + \lambda_1 \|f\|^2 + \lambda_2 \sum_{i,j=1}^{l+u} w_{ij}(f(\mathbf{x}_i) - f(\mathbf{x}_j))^2,$$

where $c(f(\mathbf{x}), y)$ is a convex loss function such as the hinge loss or the squared loss. This is a convex optimization problem with efficient solvers.

Co-training and Multiview Models

This semi-supervised learning method assumes that there are multiple, different learners trained on the same labeled data, and these learners agree on the unlabeled data. A classic algorithm is co-training (Blum & Mitchell, 1998). Take the example of web page classification, where each web page $\mathbf{x}$ is represented by two subsets of features, or "views" $\mathbf{x} = \langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \rangle$. For instance, $\mathbf{x}^{(1)}$ can represent the words on the page itself, and $\mathbf{x}^{(2)}$ the words on the hyperlinks (on other web pages) pointing to this page. The co-training algorithm trains two predictors: $f^{(1)}$ on $\mathbf{x}^{(1)}$ (ignoring the $\mathbf{x}^{(2)}$ portion of the feature) and $f^{(2)}$ on $\mathbf{x}^{(2)}$, both initially from the labeled data. If $f^{(1)}$ confidently predicts the label of an unlabeled instance $\mathbf{x}$, then the instance-label pair $(\mathbf{x}, f^{(1)}(\mathbf{x}))$ is added to $f^{(2)}$'s labeled training data, and vice versa. Note this promotes $f^{(1)}$ and $f^{(2)}$ to predict the same on $\mathbf{x}$. This repeats so that each view teaches the other. Multiview models generalize co-training by utilizing more than two predictors, and relaxing the requirement of having separate views (Sindhwani, Niyogi, & Belkin, 2005). In either case, the final prediction is obtained from a (confidence weighted) average or vote among the predictors.

To define the implicit ordering on the hypothesis space, we need a slight extension. In general, let there be m predictors $f^{(1)}, \ldots, f^{(m)}$. Now let a hypothesis be an m-tuple of predictors $\langle f^{(1)}, \ldots, f^{(m)} \rangle$. The disagreement of a tuple on the unlabeled data can be defined as

$$\sum_{i=l+1}^{l+u} \sum_{u,v=1}^{m} c(f^{(u)}(\mathbf{x}_i), f^{(v)}(\mathbf{x}_i)),$$

where $c()$ is a loss function. Typical choices of $c()$ are the 0–1 loss for classification, and the squared loss for regression. Then the disagreement induces an implicit ordering on tuples from small to large.

It is important for these m predictors to be of diverse types, and have different ▶inductive biases. In general, each predictor $f^{(u)}, u = 1 \ldots m$ may be evaluated by its individual loss function $c^{(u)}$ and regularizer $\Omega^{(u)}$. To find a hypothesis (i.e., m predictors) that fits the labeled data well and ranks high, one can minimize the following objective:

$$\text{argmin}_{\langle f^{(1)}, \ldots, f^{(m)} \rangle} \sum_{u=1}^{m} \left(\frac{1}{l} \sum_{i=1}^{l} c^{(u)}(f^{(u)}(\mathbf{x}_i), y_i) + \lambda_1 \Omega^{(u)}(f^{(u)}) \right) + \lambda_2 \sum_{i=l+1}^{l+u} \sum_{u,v=1}^{m} c(f^{(u)}(\mathbf{x}_i), f^{(v)}(\mathbf{x}_i)).$$

Multiview learning typically optimizes this objective directly. When the loss functions and regularizers are convex, numerical solution is relatively easy to obtain. In the special cases when the loss functions are the squared loss, and the regularizers are squared ℓ_2 norms, there is a closed form solution. On the other hand, the co-training algorithm, as presented earlier, optimizes the objective indirectly with the iterative procedure. One advantage of co-training is that the algorithm is a wrapper method, in that it can use any "blackbox" learners $f^{(1)}$ and $f^{(2)}$ without the need to modify the learners.

A PAC Bound for Semi-Supervised Learning

Previously, we presented several semi-supervised learning methods, each induces an implicit ordering on the hypothesis space using the unlabeled training data, and each attempts to find a hypothesis that fit the labeled training data well as well as rank high in that implicit ordering. We now present a theoretical justification on why this is a good idea. In particular, we present a uniform convergence bound by Balcan and Blum

(Theorem 11 in Balcan and Blum (2009)). Alternative theoretical analyses on semi-supervised learning can be found by following the recommended reading.

First, we introduce some notations. Consider the 0–1 loss for classification. Let $c^*: \mathcal{X} \mapsto \{0,1\}$ be the unknown target function, which may not be in $\mathcal{F}$. Let $\text{err}(f) = E_{\mathbf{x}\sim p}[f(\mathbf{x}) \neq c^*(\mathbf{x})]$ be the true error rate of a hypothesis f, and $\widehat{\text{err}}(f) = \frac{1}{l}\sum_{i=1}^{l} f(\mathbf{x}_i) \neq c^*(\mathbf{x}_i)$ be the empirical error rate of f on the labeled training sample. To characterize the implicit ordering, we defined an "unlabeled error rate" $\text{err}_{\text{unl}}(f) = 1 - E_{\mathbf{x}\sim p}[\chi(f,\mathbf{x})]$, where the *compatibility function* $\chi: \mathcal{F} \times \mathcal{X} \mapsto [0,1]$ measures how "compatible" f is to an unlabeled instance $\mathbf{x}$. As an example, in semi-supervised support vector machines, if $\mathbf{x}$ is far away from the decision boundary produced by f, then $\chi(f,\mathbf{x})$ is large; but if $\mathbf{x}$ is close to the decision boundary, $\chi(f,\mathbf{x})$ is small. In this example, a large $\text{err}_{\text{unl}}(f)$ then means that the decision boundary of f cuts through dense unlabeled data regions, and thus f is undesirable for semi-supervised learning. In contrast, a small $\text{err}_{\text{unl}}(f)$ means that the decision boundary of f lies in a low density gap, which is more desirable. In theory, the implicit ordering on $f \in \mathcal{F}$ is to sort $\text{err}_{\text{unl}}(f)$ from small to large. In practice, we use the empirical unlabeled error rate $\widehat{\text{err}}_{\text{unl}}(f) = 1 - \frac{1}{u}\sum_{i=l+1}^{l+u} \chi(f,\mathbf{x}_i)$.

Our goal is to show that if an $f \in \mathcal{F}$ "fits the labeled data well and ranks high," then f is almost as good as the best hypothesis in $\mathcal{F}$. Let $t \in [0,1]$. We first consider the best hypothesis f_t^* in the subset of $\mathcal{F}$ that consists of hypotheses whose unlabeled error rate is no worse than t: $f_t^* = \text{argmin}_{f'\in\mathcal{F}, \text{err}_{\text{unl}}(f')\leq t}\text{err}(f')$. Obviously, $t=1$ gives the best hypothesis in the whole $\mathcal{F}$. However, the nature of the guarantee has the form $\text{err}(f) \leq \text{err}(f_t^*) + \text{EstimationError}(t) + c$, where the EstimationError term increases with t. Thus, with $t = 1$ the bound can be loose. On the other hand, if t is close to 0, $\text{EstimationError}(t)$ is small, but $\text{err}(f_t^*)$ can be much worse than $\text{err}(f_{t=1}^*)$. The bound will account for the optimal t.

We introduce a few more definitions. Let $\mathcal{F}(f) = \{f' \in \mathcal{F} : \widehat{err}_{unl}(f') \leq \widehat{err}_{unl}(f)\}$ be the subset of $\mathcal{F}$ with empirical error no worse than that of f. As a complexity measure, let $[\mathcal{F}(f)]$ be the number of different partitions of the first l unlabeled instances $\mathbf{x}_{l+1} \ldots \mathbf{x}_{2l}$, using $f \in \mathcal{F}(f)$. Finally, let $\hat{\epsilon}(f) = \sqrt{\frac{24}{l}\log(8[\mathcal{F}(f)])}$. Then we have the following agnostic bound (meaning that c^* may not be in $\mathcal{F}$, and $\widehat{\text{err}}_{\text{unl}}(f)$ may not be zero for any $f \in \mathcal{F}$):

Theorem 1 *Given l labeled instances and sufficient unlabeled instances, with probability at least $1-\delta$, the function*

$$f = argmin_{f'\in\mathcal{F}}\widehat{\text{err}}(f') + \hat{\epsilon}(f')$$

satisfies the guarantee that

$$\text{err}(f) \leq \min_t(\text{err}(f_t^*) + \hat{\epsilon}(f_t^*)) + 5\sqrt{\frac{log(8/\delta)}{l}}.$$

If a function f fits the labeled data well, it has a small $\widehat{\text{err}}(f)$. If it ranks high, then $\mathcal{F}(f)$ will be a small set, consequently $\hat{\epsilon}(f)$ is small. The argmin operator identifies the best such function during training. The bound account for the minimum of all possible t tradeoffs. Therefore, we see that the "lucky" case is when the implicit ordering is good such that $f_{t=1}^*$, the best hypothesis in $\mathcal{F}$, is near the top of the ranking. This is when semi-supervised learning is expected to perform well. Balcan and Blum also give results addressing the key issue of how much *unlabeled* data is needed for $\widehat{\text{err}}_{\text{unl}}(f)$ and $\text{err}_{\text{unl}}(f)$ to be close for all $f \in \mathcal{F}$.

Applications

Because the type of semi-supervised learning discussed in this entry has the same goal of creating a predictor as supervised learning, it is applicable to essentially any problems where supervised learning can be applied. For example, semi-supervised learning has been applied to natural language processing (word sense disambiguation (Yarowsky, 1995), document categorization, named entity classification, sentiment analysis, machine translation), computer vision (object recognition, image segmentation), bioinformatics (protein function prediction), and cognitive psychology. Follow the recommended reading for individual papers.

Future Directions

There are several directions to further enhance the value semi-supervised learning. First, we need guarantees that it will outperform supervised learning. Currently, the practitioner has to manually choose a particular

semi-supervised learning method, and often manually set learning parameters. Sometimes, a bad choice that does not match the task (e.g., modeling each class with a Gaussian when the data does not have this distribution) can make semi-supervised learning worse than supervised learning. Second, we need methods that benefit from unlabeled when l, the size of labeled data, is large. It has been widely observed that the gain over supervised learning is the largest when l is small, but diminishes as l increases. Third, we need good ways to combine semi-supervised learning and ►active learning. In natural learning systems such as humans, we routinely observe unlabeled input, which often naturally leads to questions. And finally, we need methods that can efficiently process massive unlabeled data, especially in an ►online learning setting.

Cross References

►Active Learning
►Classification
►Constrained Clustering
►Dimensionality Reduction
►Online Learning
►Regression
►Supervised Learning
►Unsupervised Learning

Recommended Reading

Abney, S. (2007). *Semisupervised learning for computational linguistics*. Florida: Chapman & Hall/CRC.

Balcan, M.-F., & Blum, A. (2009). A discriminative model for semi-supervised learning. *Journal of the ACM*.

Belkin, M., Niyogi, P., & Sindhwani, V. (2006). Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research, 7*, 2399–2434.

Blum, A., & Chawla, S. (2001). Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the 18th international conference on machine learning* (pp. 19–26). San Francisco: Morgan Kaufmann.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the workshop on computational learning theory* (pp. 92–100). New York: ACM.

Castelli, V., & Cover, T. (1995). The exponential value of labeled samples. *Pattern Recognition Letters, 16*(1), 105–111.

Chapelle, O., Zien, A., & Schölkopf, B., (Eds.) (2006). *Semi-supervised learning*. Cambridge, MA MIT Press.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *Proceedings of the 16th international conference on machine learning* (pp. 200–209). San Francisco: Morgan Kaufmann.

Nigam, K., McCallum, A. K., Thrun, S., & Mitchell, T. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning, 39*(2/3), 103–134.

Seeger, M. (2001). *Learning with labeled and unlabeled data*. Technical report. University of Edinburgh, Edinburgh.

Sindhwani, V., Niyogi, P., & Belkin, M. (2005). A co-regularized approach to semi-supervised learning with multiple views. In *Proceedings of the 22nd ICML workshop on learning with multiple views.*

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting of the association for computational linguistics* (pp. 189–196).

Zhu, X., Ghahramani, Z., & Lafferty, J. (2003). Semi-supervised learning using Gaussian fields and harmonic functions. In *The 20th international conference on machine learning (ICML)*.

Zhu, X., & Goldberg, A. B. (2009). Synthesis lectures on artificial intelligence and machine learning. In *Introduction to semi-supervised learning*. Morgan & Claypool.

Semi-Supervised Text Processing

Ion Muslea
Language Weaver, Inc.,
Marina del Rey, CA, USA

Synonyms

Learning from labeled and unlabeled data; Transductive learning

Definition

In contrast to supervised and unsupervised learners, which use solely labeled or unlabeled examples, respectively, semi-supervised learning systems exploit both labeled and unlabeled examples. In a typical semi-supervised framework, the system takes as input a (small) training set of labeled examples and a (larger) working set of unlabeled examples; the learner's performance is evaluated on a test set that consists of unlabeled examples. Transductive learning is a particular case of semi-supervised learning in which the working set and the test set are identical.

Semi-supervised learners use the unlabeled examples to improve the performance of the system that could be learned solely from labeled data. Such learners typically exploit – directly or indirectly – the distribution of the available unlabeled examples. Text

processing is an ideal application domain for semi-supervised learning because the abundance of text documents available on the Web makes it impossible for humans to label them all. We focus here on two related types of text processing tasks that were heavily studied in the semi-supervised framework: text classification and text ▶Clustering.

Motivation and Background

In most applications of machine learning, collecting large amounts of labeled examples is an expensive, tedious, and error-prone process. In contrast, one may often have cheap or even free access to large amounts of unlabeled examples. For example, for text classification, which is the task of classifying text documents into categories such as politics, sports, entertainment, etc., one can easily crawl the Web and download billions of Web pages; however, manually labeling all these documents according to the taxonomy of interest is an extremely expensive task.

The key idea in semi-supervised learning is to complement a small amount of labeled data by a large number of unlabeled examples. Under certain conditions, the unlabeled examples can be mined for knowledge that will allow the semi-supervised learner to build a system that performs better than one learned solely from the labeled data. More precisely, semi-supervised learners assume that the learning model matches the structure of the application domain. If this is the case, the information extracted from the unlabeled data can be used to *guide* the search towards the optimal solution (e.g., by modifying or re-ranking the learned hypotheses); otherwise, the unlabeled examples may *hurt* rather than help the learning process (Cozman, Cohen, & Cirelo, 2003).

For the sake of concision and clarity, we have had to make several compromises in terms of the algorithms and the applications presented here. Given the vastness of the field of text processing, we have decided to focus only on the two related tasks of text classification and text clustering. They are the most studied text processing applications within the field of machine learning; furthermore, virtually all the main types of semi-supervised algorithms were applied to these two tasks. This decision has two main consequences. First, we do not consider many other text processing tasks, such as information extraction, natural language parsing, or base noun–phrase identification; for these we refer the interested reader to Muslea, Minton, and Knoblock (2006). Second, we discuss and cite approaches that were applied to text classification or clustering there is however, alone an excellent survey by Zhu (2005) covering seminal work on semi-supervised learning that was not applied to text processing.

Structure of the Learning System

Generative Models

The early work on semi-supervised text categorization (Nigam, McCallum, Thrun, & Mitchell, 2000) was based primarily on generative models (see ▶generative learning). Such approaches make two major assumptions: (1) the data is generated by a mixture model, and (2) there is a correspondence between the components of the mixture and the classes of the application domain. Intuitively, if these assumptions hold, the unlabeled examples become instrumental in identifying the mixture's components, while the labeled examples can be used to label each individual component.

The iterative approach proposed by Nigam et al. (2000) is based on ▶The EM Algorithm and works as follows. First, the labeled examples are used to learn an initial classifier, which is used to probabilistically label all unlabeled data; then the newly labeled examples are added to the training set. Finally, a new classifier is learned from all the data, and the entire process is repeated till convergence is reached (or, alternatively, till the number of iterations is fixed).

Nigam et al. (2000) noticed that, in practice, the two above-mentioned assumptions about the generative model may not hold; in order to deal with this problem, the authors propose two extensions of their basic approach. First, they allow each class to be generated by multiple mixture components. Second, they introduce a weighting factor that adjusts the contribution of the unlabeled examples; this factor is tuned during the learning process so that the influence of the unlabeled examples correlates with the degree in which the data distribution is consistent with the mixture model.

The same general framework can also be applied to the related task of text clustering. In the clustering framework, the learner is not concerned with the

actual label of an example; instead, it tries to find a partitioning of the examples in clusters that are similar *respect to* a predefined objective function. For example, Seeded-KMeans (Basu, Banerjee, & Mooney, 2002) is a semi-supervised text clustering algorithm that uses the few available labeled examples to seed the search for the data clusters. In order to optimize the target objective function, Seeded-KMeans uses an EM algorithm on a mixture of Gaussians.

Discriminative Approaches

►Support vector machines (SVMs) (Joachims, 1999) are particularly well suited for text classification because of their ability to deal with high-dimensional input spaces (each word in the corpus is a feature) and sparse feature-value vectors (any given document contains only a small fraction of the corpus vocabulary). SVMs are called maximum margin classifiers because they minimize the empirical classification error by maximizing the geometric margin between the domain's positive and negative examples. Intuitively, this is equivalent to finding a discriminative decision boundary that avoids the high-density regions in the instance space.

Transductive SVMs (Joachims, 1999) are designed to find an optimal decision boundary for a particular test set. More precisely, they have access to both the (labeled) training set and the unlabeled test set. Transductive SVMs work by finding a labeling of the test examples that maximizes the margin over all the examples in the training and the test set. This transductive approach has shown significant improvements over the traditional inductive SVMs, especially if the size of the training set is small.

In contrast to transductive SVMs, semi-supervised SVMs (S3VM) work in a true semi-supervised setting in which the test set is not available to the learner. A major difficulty in the S3VM framework is the fact that the resulting optimization problem is not convex, thus being sensitive to the issue of (non-optimal) local minima. CS3VMs (Chapelle, Chi, & Zien, 2006) alleviate this problem by using a global optimization technique called continuation. On binary classification tasks CS3VMs compare favorably against other S3VM approaches, but applying it on multiclass domains is still an open problem.

Multiview Approaches

Multiview learners are a class of algorithms for domains in which the features can be partitioned in disjoint subsets (views), each of which is sufficient to learn the target concept. For example, when classifying Web pages, one can use either the words that appear in the documents or those that appear in the hyper-links pointing to them. Co-training (Blum & Mitchell, 1998) is a semi-supervised, multiview learner that, intuitively, works by bootstrapping the views from each other. First, it uses the labeled examples to learn a classifier in each view. Then it applies the learned classifiers to the unlabeled data and detects the examples on which each view makes the most confident prediction; these examples are labeled by the respective classifiers and added to the (labeled) training set of the other view. The entire process is repeated for a number of iterations.

Multiview learners rely on two main assumptions, namely that the views are compatible and uncorrelated. The former requires that each example is identically labeled by the target concept in each view; the latter means that given an example's label, its description in each view are independent. In practice, both these assumptions are likely to be violated; in order to deal with the first issue, one can use the adaptive view validation algorithm (Muslea, Minton, & Knoblock, 2002b), which predicts whether the views are sufficiently compatible for multiview learning.

With respect to view correlation Muslea, Minton, and Knoblock (2002a) have shown that by interleaving active and semi-supervised learning, multiview approaches become robust the view correlation. A similar idea was previously used in the generative, single-view framework: McCallum and Nigam (1998) have shown that by allowing the algorithm to (smartly) choose which examples to include in the training set, one can significantly improve over the performance of both supervised and semi-supervised learners that used randomly chosen training sets.

The main limitation of multiview learning is the requirement that the user identifies at least two suitable views. In order to cope with this problem, researchers have proposed algorithms that work in a way similar to co-training, but exploit multiple ►inductive biases instead of multiple views. For example, tri-training (Zhou & Li, 2005) uses all domain features to train three supervised classifiers (e.g., a decision tree, a neural

network, and a Naive Bayes classifier). These classifiers are then applied to each unlabeled example; if two of them agree on the example's label, they label it accordingly and add it to the third classifier's training set. A degenerate case is represented by *self-training*, which uses a single classifier that repeatedly goes through the unlabeled data and adds to its own training set, the examples on which its predictions are the most confident.

Graph-Based Approaches

The work on graph-based, semi-supervised text learning is based on the idea of representing the labeled and unlabeled examples as vertices in a graph. The edges of this graph are weighted by the pair-wise similarity between the corresponding examples, thus offering a flexible way to incorporate prior domain knowledge. With the learning task encoded in this manner, the problem to be solved becomes one of graph theory, namely finding a partitioning of the graph that agrees with the labeled examples. A major challenge for the graph-based approaches is to find a balanced partitioning of the graph (e.g., in a degenerate scenario, one can propose an unbalanced, undesirable partition in which, except for the negative examples in the training set, all other examples are labeled as positive).

One possible approach to cope with the issue on unbalanced partitions is to use randomized min-cuts (Blum, Lafferty, Rwebangira, & Reddy, 2004). The algorithm starts with the original graph and repeatedly adds random noise to the weights of the edges. Then, for each modified graph, it finds a partitioning by using minimum cuts. Finally, the results from the various runs aggregated in order to create probabilistic labels for the unlabeled examples. This approach has the additional benefit of offering a measure of the confidence in each particular prediction.

The SGT algorithm (Joachims, 2003) uses spectral methods to perform the graph partitioning. SGT can be seen as a transductive version of the k nearest-neighbor classifier; furthermore Joachims (2003) also show that co-training emerges as a special case of SGT. In contrast to transductive SVMs and co-training, SGT does not require additional heuristics for avoiding unbalanced graph partitionings (e.g., in the original co-training algorithm, the examples that are added to the training set after each iteration must respect the domain-dependent ratio of negative-to-positive examples).

LapSVM (Sindhwani, Niyogi, & Belkin, 2005) is a graph-based kernel method that uses a weighted combination a regularizer learned solely from labeled data and a graph Laplacian obtained from both the labeled and unlabeled examples. This approach allows LapSVM to perform a principled search for a decision boundary that is both consistent with the labeled examples and reflects the underlying geometry of all available data points.

Approaches that Exploit Background Knowledge

WHIRL-BG (Zelikovitz & Hirsh, 2000) is an algorithm for classifying short text fragments. It uses an information integration approach that combines three different information sources: the training set, which consists of the labeled examples; the test set that WHIRL-BG must label; and a secondary corpus that consists longer, related documents that are not labeled. Intuitively, WHIRL-BG exploits the secondary corpus as background knowledge that allows the system to link a test example to the most similar labeled training example. In other words, instead of trying to measure directly a (unreliable) similarity between two short strings (i.e., a test and a training example), the system searches for a background document that may include (a large fraction of) both strings.

HMRF-KMEANS (Basu, Bilenko, & Mooney, 2004) unifies the two main approaches to semi-supervised text clustering: the constraint-based one and the adaptive distance one. The former exploits user-provided background knowledge to find an appropriate partitioning of the data; for HMRF-KMEANS, the domain knowledge consists of must-link or cannot-link constraints, which specify whether two examples should or should not have the same label, respectively. The later uses a small number of labeled examples to learn a domain-specific distance measure that is appropriate for the clustering task at hand. HMRF-KMEANS can use any Bregman divergence to measure the clustering distortion, thus supporting a wide variety of learnable distances.

HMRF-KMEANS exploits the labeled examples in three main ways. First, it uses the neighborhoods induced from the constraints to initialize the cluster centroids. Second, when assigning examples to clusters, the algorithm tries to simultaneously minimize both the similarity to the cluster's centroid and the number of violated constraints. Last but not least, during the clustering process, HMRF-KMEANS iteratively re-estimates the distance measure so that it takes into account both the background knowledge and the data variance.

Recommended Reading

Basu, S., Banerjee, A., & Mooney, R. (2002). Semi-supervised clustering by seeding. In *Proceedings of the international conference on machine learning* (pp. 19–26). Sydney, Australia.

Basu, S., Bilenko, M., & Mooney, R. (2004). A probabilistic framework for semi-supervised clustering. In *Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 59–68). Seattle, WA.

Blum, A., Lafferty, J., Rwebangira, M. R., & Reddy, R. (2004). Semi-supervised learning using randomized mincuts. In *Proceedings of the twenty-first international conference on machine learning* (p. 13).

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the 1988 conference on computational learning theory* (pp. 92–100).

Chapelle, O., Chi, M., & Zien, A. (2006). A continuation method for semi-supervised SVMs. In *Proceedings of the 23rd international conference on machine learning* (pp. 185–192). New York: ACM Press.

Cozman, F., Cohen, I., & Cirelo, M. (2003). Semi-supervised learning of mixture models. In *Proceedings of the international conference on machine learning* (pp. 99–106). Washington, DC.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *Proceedings of the 16th international conference on machine learning (ICML-99)* (pp. 200–209). San Francisco: Morgan Kaufmann.

Joachims, T. (2003). Transductive learning via spectral graph partitioning. In *Proceedings of the international conference on machine learning*.

McCallum, A., & Nigam, K. (1998). Employing EM in pool-based active learning for text classification. In *Proceedings of the 15th international conference on machine learning* (pp. 359–367).

Muslea, I., Minton, S., & Knoblock, C. (2002a). Active + semi-supervised learning = robust multi-view learning. In *The 19th international conference on machine learning (ICML-2002)* (pp. 435–442). Sydney, Australia.

Muslea, I., Minton, S., & Knoblock, C. (2002b). Adaptive view validation: A first step towards automatic view detection. In *The 19th international conference on machine learning (ICML-2002)* (pp. 443–450). Sydney, Australia.

Muslea, I., Minton, S., & Knoblock, C. (2006). Active learning with multiple views. *Journal of Artificial Intelligence Research, 27*, 203–233.

Nigam, K., McCallum, A. K., Thrun, S., & Mitchell, T. M. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning, 39*(2/3), 103–134.

Sindhwani, V., Niyogi, P., & Belkin, M. (2005). Beyond the point cloud: From transductive to semi-supervised learning. In *Proceedings of the 22nd international conference on machine learning* (pp. 824–831). Bonn, Germany.

Zelikovitz, S., & Hirsh, H. (2000). Improving short text classification using unlabeled background knowledge. In *Proceedings of the 17th international conference on machine learning* (pp. 1183–1190).

Zhou, Z.-H., & Li, M. (2005). Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering, 17*(11), 1529–1541.

Zhu, X. (2005). *Semi-supervised learning literature survey*. Technical report 1530, Department of Computer Sciences, University of Wisconsin, Madison.

Sensitivity

Synonyms

Recall; True positive rate

Sensitivity is the fraction of positive examples predicted correctly by a model. See ►Sensitivity and Specificity, ►Recall and Precision.

Sensitivity and Specificity

Kai Ming Ting
Monash University, Gippsland Campus Churchill, VIC, Australia

Definition

Sensitivity and specificity are two measures used together in some domains to measure the predictive performance of a classification model or a diagnostic test. For example, to measure the effectiveness of a diagnostic test in the medical domain, sensitivity measures the fraction of people with disease (i.e., positive examples) who have a positive test result; and specificity measures the fraction of people without disease (i.e., negative examples) who have a negative test result. They are defined with reference to a special case of the ►confusion matrix, with two classes, one designated

Sensitivity and Specificity. Table 1 The outcomes of classification into positive and negative classes

		Assigned Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

the *positive* class, and the other the *negative* class, as indicated in Table 1.

Sensitivity is sometimes also called *true positive rate*. Specificity is sometimes also called *true negative rate*. They are defined as follows:

Sensitivity = TP/(TP + FN)

Specificity = TP/(TN + FP)

Instead of two measures, they are sometimes combined to provide a single measure of predictive performance as follows:

Sensitivity × Specificity

= TP * TN/[(TP + FN) * (TN + FP)]

Note that sensitivity is equivalent to ►recall.

Cross References

►Confusion Matrix

Sequence Data

►Sequential Data

Sequential Data

Synonyms

Sequence data

Sequential Data refers to any *data* that contain elements that are ordered into sequences. Examples include ►time series, DNA sequences (see ►biomedical informatics) and sequences of user actions. Techniques for learning from sequential data include ►Markov models, ►Conditional Random Fields and ►time series techniques.

Sequential Inductive Transfer

►Cumulative Learning

Sequential Prediction

►Online Learning

Set

►Class

Shannon's Information

If a message announces an event E_1 of probability $P(E_1)$ its information content is $-\log_2 P(E_1)$. This is also its length in bits.

Shattering Coefficient

Synonyms

Growth function

Definition

The shattering coefficient $S_{\mathcal{F}}(n)$ is a function that measures the size of a function class $\mathcal{F}$ when its functions $f : \mathcal{X} \to \mathbb{R}$ are restricted to sets of points $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ of size n. Specifically, for each $n \in \mathbb{N}$ the shattering coefficient is the maximum size of the set of vectors $\mathcal{F}_{\mathbf{x}} = \{(f(x_1), \ldots, f(x_n)) : f \in \mathcal{F}\} \subset \mathbb{R}^n$ that can be realized for some choice of $\mathbf{x} \in \mathcal{X}^n$. That is,

$$S_{\mathcal{F}}(n) = \sup_{\mathbf{x} \in \mathcal{X}^n} |\mathcal{F}_{\mathbf{x}}|.$$

The shattering coefficient of a hypothesis class $\mathcal{H}$ is used in ►generalization bounds as an analogue to the class's size in the finite case.

Similarity Measures

Michail Vlachos
IBM Zürich Research Laboratory, Rüschlikon, Switzerland

Synonyms

Distance; Distance metrics; Distance functions; Distance measures

Definition

The term similarity measure refers to a function that is used for comparing objects of any type. The objects can be data structures, database records, or even multimedia objects (audio, video, etc.). Therefore, the input of a similarity measure is two objects and the output is, in general, a number between 0 and 1; "zero" meaning that the objects are completely dissimilar and "one" signifying that the two objects are identical. Similarity is related to distance, which is the inverse of similarity. That is, a similarity of 1 implies a distance of 0 between two objects.

Motivation and Background

Similarity measures are typically used for quantifying the affinity between objects in search operations, where the user presents an object (query) and requests other objects "similar" to the given query. Therefore, a similarity measure is a mathematical abstraction for comparing objects, assigning a single number that indicates the affinity between the said pair of objects. The results of the search are typically presented to the user in the order suggested by the returned similarity value. Objects with higher similarity value are presented first to the user because they are deemed to be more relevant to the query posed by the user. For example, when searching for specific keywords on an Internet search engine, Internet pages that are more relevant/similar to the posed query are presented first. The selection of the proper similarity function is a important parameter in many applications, including ►instance-based learning, ►clustering, and ►anomaly detection.

Most similarity measures attempt to model (imitate) the human notion of similarity between objects. If a similarity function resembles very closely the similarity ranking between objects as returned by a human, then it is considered successful. This is where the difficulty also lies, because in general similarity is something that is very subjective.

Consider the case where a user poses a keyword query 'crane' at a search engine, while searching for images. The returned results would contain images with machineries, birds or even origami creations. This is because when the similarity measure used is solely based on textual information then, then all such images are indeed proper answers to the query. If one was interested also in the semantics of an image, then perhaps additional features such as texture, color or shape could have been utilized. Therefore, for defining an effective similarity measure, one has to first extract the proper object features and then evaluate the similarity using an appropriate distance function.

Classes of Similarity Functions

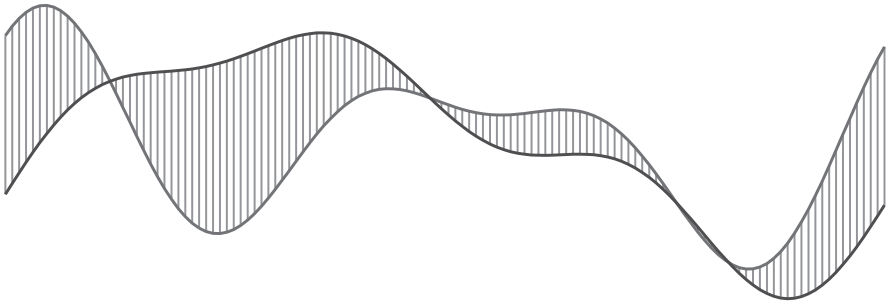
There are two major classes of similarity functions: metric functions and non-metric functions. In order for a function d to be a metric it has to satisfy all the following three properties for any objects X, Y, Z:

1. $d(X,Y) = 0$ iff $X = Y$ (identity axiom)
2. $d(X,Y) = d(Y,X)$ (symmetry axiom)
3. $d(X,Y) + d(Y,Z) \geq d(X,Z)$ (triangle inequality)

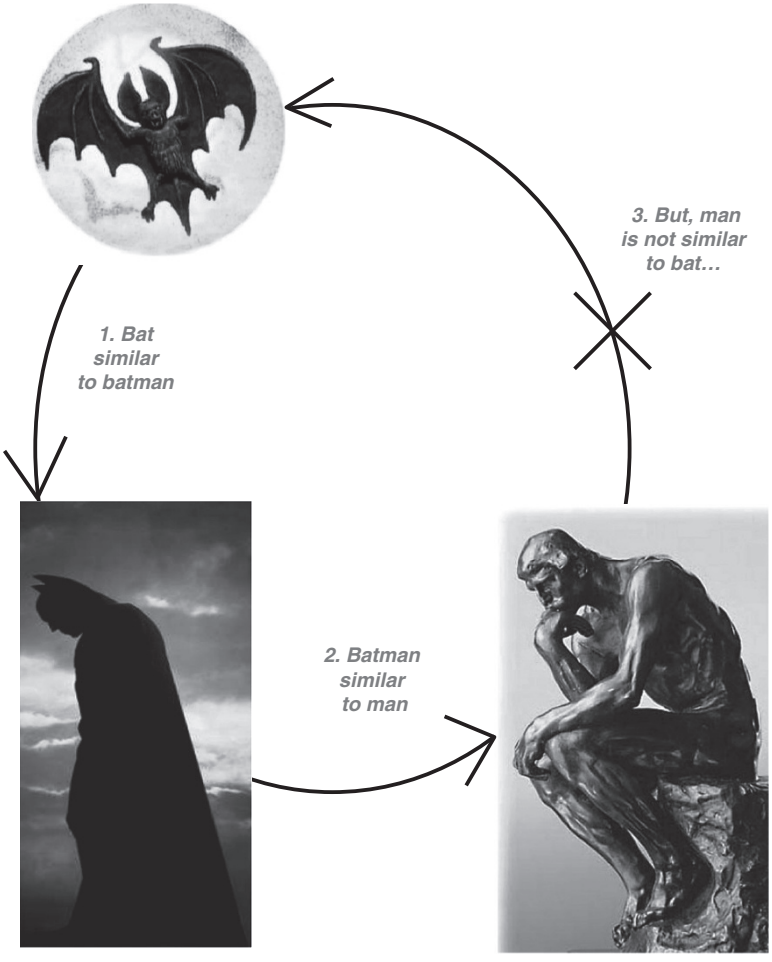
Metric similarity functions are very widely used in search operations because of their support of the triangle inequality. The triangle inequality can help prune a lot of the search space, by eliminating objects from examination that are guaranteed to be distant to the given query (Agrawal et al., 1993; Zezula et al., 2005). The most frequently used metric similarity function is the Euclidean distance. For two objects X and Y that are characterized by set of n features $X = (x_1, x_2, \ldots, x_n)$ and similarly $Y = (y_1, y_2, \ldots, y_n)$ the Euclidean distance is defined as

$$D = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

If we represent the objects X and Y as an ordered sequence of their features, we can visualize the



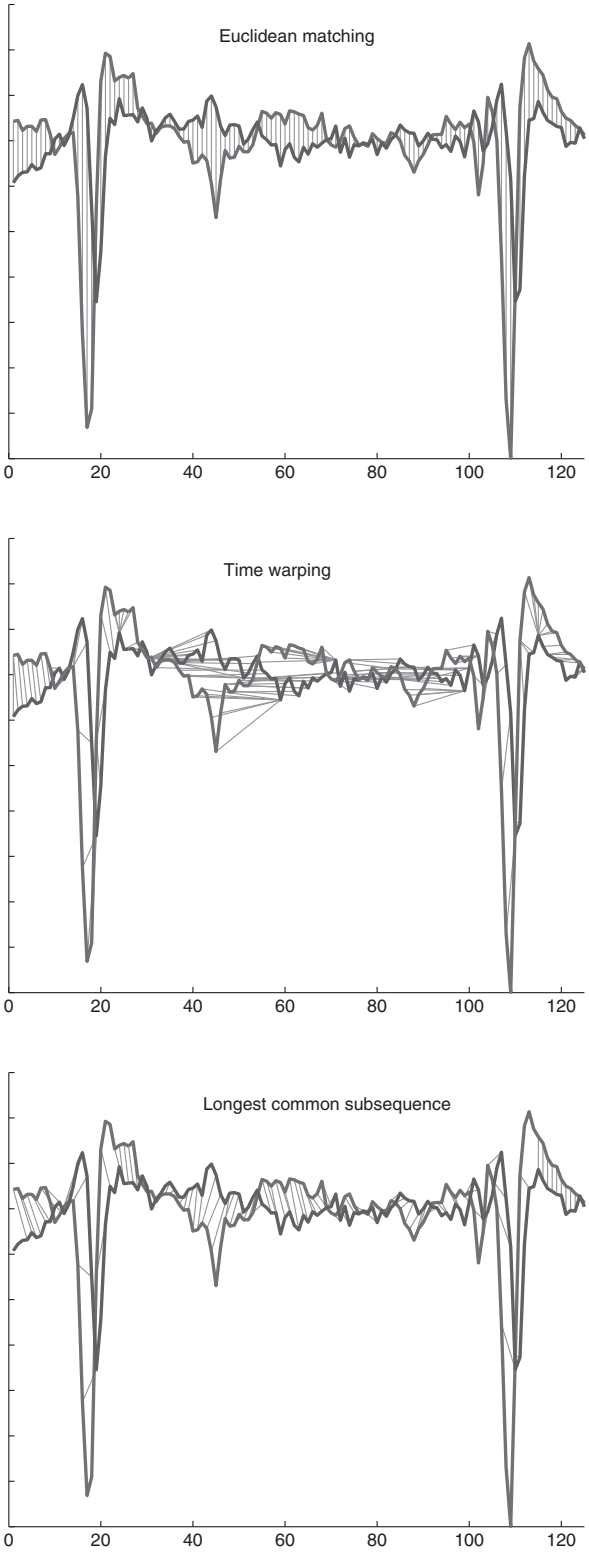
Similarity Measures. Figure 1. Mapping achieved by the Euclidean distance between time-series data



Similarity Measures. Figure 2. Nonmetric similarity that disobeys the triangle inequality

linear mapping achieved by the Euclidean distance in Fig. 1.

Non-metric similarity measures resemble more closely the human notion of similarity by allowing more flexible matching between the examined objects, for example, by allowing non-linear mappings or even by accommodating occlusion of points or features. The human visual system is in general considered nonmetric. Non-metric functions typically disobey the triangle inequality. We can see an example of this below in Fig. 2.



Similarity Measures. Figure 3. Comparison of Euclidean, warping, and longest common subsequence measures

Widely used non-metric similarity functions are the Warping distance and the Longest Common Subsequence (LCSS). The Warping distance (also known as dynamic time warping – DTW) has been very extensively used in the past in voice recognition tasks, due to its ability to perform compression or decompression of the features, allowing flexible non-linear mappings. In Fig. 3 we visually depict the outcome of the previously mentioned measures for ►time-series data. The Euclidean distance performs a rigid linear mapping of points, the DTW can perform nonlinear one-to-many mappings, and the LCSS constructs a one-to-one non-linear mapping.

Recently, similarity metrics based on information theory, and in specific, on Kolmogorov complexity have been presented (Keogh et al., 2004; Li et al., 2004) and can also be considered as *compression-based* measures. A very simple and easily implementable version of a compression based distance is

$$d_c(X, Y) = \frac{C(XY)}{C(X) + C(Y)}$$

where $C(X)$ is the compressed size (bytes) of X given a certain compression algorithm. The distance will be close to 1, if X and Y are dissimilar and less than 1 when X and Y are related. Therefore, we exploit the fact that if X and Y are "similar" they should compress equally well (approximately same amount of bytes) either when considered separately or together, because the compression dictionaries will be similar when the two objects are related.

In summary, the choice of a similarity metric is highly dependent on the application at hand. The practitioner should also closely consider on which object features the similarity measure will be applied. Ultimately, the combination of both feature selection and similarity metric will define the quality of a search process.

Cross References

►Dimensionality Reduction
►Feature Selection

Recommended Readings

Agrawal, R., Faloutsos, C., & Swami, A. (1993). Efficient similarity search in sequence databases. In *Proceedings of foundations of data organization and algorithms (FODO)*, (pp. 69–84). Chicago, Illinois, USA.

Keogh, E., Lonardi, S., & Ratanamahatana, A. (2004). Towards parameter-free data mining. *Proceedings of International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (pp. 206–215). Seattle, Washington, USA.

Li, M., Chen, X., Li, X., Ma, B., & Vitanyi, P. M. B. (2004). The similarity metric. *IEEE Transactions on Information Theory, 50*(12), 3250–3264.

Zezula, P., Amato, G., Dohnal, V., & Batko, M. (2005). *Similarity search: the metric approach*. Advances in Database Systems, Springer.

Simple Bayes

►Naïve Bayes

Simple Recurrent Network

Risto Miikkulainen
The University of Texas at Austin, Austin, TX, USA

Synonyms

Elman network; Feedforward recurrent network

Definition

The simple recurrent network is a specific version of the ►Backpropagation neural network that makes it possible to process of sequential input and output (Elman, 1990). It is typically a three-layer network where a copy of the hidden layer activations is saved and used (in addition to the actual input) as input to the hidden layer in the next time step. The previous hidden layer is fully connected to the hidden layer. Because the network has no recurrent connections per se (only a copy of the activation values), the entire network (including the weights from the previous hidden layer to the hidden layer) can be trained with the backpropagation algorithm as usual. It can be trained to read a sequence of inputs into a target output pattern, to generate a sequence of outputs from a given input pattern, or to map an input sequence to an output sequence (as in predicting the next input). Simple recurrent networks have been particularly useful in ►time series prediction, as well as in modeling cognitive processes, such as language understanding and production.

Recommended Reading

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14*, 179–211.

SMT

►Statistical Machine Translation

Solution Concept

A criterion specifying which locations in the search space are solutions and which are not. In designing a coevolutionary algorithm, it is important to consider whether the solution concept implemented by the algorithm (i.e., the set of individuals to which it can converge) corresponds with the intended solution concept.

Solving Semantic Ambiguity

►Word Sense Disambiguation

SOM

►Self-Organizing Maps

SORT

►Class

Spam Detection

►Text Mining for Spam Filtering

Specialization

Specialization is the converse of ►generalization. Thus, if h_1 is a generalization of h_2 then h_2 is a specialization of h_1.

Cross References

►Generalization
►Induction
►Learning as Search
►Subsumption
►Logic of Generality

Specificity

Synonyms

True negative rate

Specificity is the fraction of negative examples predicted correctly by a model. See ►Sensitivity and Specificity.

Spectral Clustering

►Graph Clustering

Speedup Learning

Alan Fern
Science, Oregon State University,
Corvallis, OR, USA

Definition

Speedup learning is a branch of machine learning that studies learning mechanisms for speeding up problem solvers based on problem-solving experience. The input to a speedup learner typically consists of observations of prior problem-solving experience, which may include traces of the problem solver's operations and/or solutions to solve the problems. The output is knowledge that the problem solver can exploit to find solutions more quickly than before learning without seriously effecting the solution quality. The most distinctive feature of speedup learning, compared with most branches of machine learning, is that the learned knowledge does not provide the problem solver with the ability to solve new problem instances. Rather, the learned knowledge is intended solely to facilitate faster solution times compared to the solver without the knowledge.

Motivation and Background

Much of the work in computer science and especially artificial intelligence aims at developing practically-efficient problem solvers for combinatorially hard problem classes such as automated planning, logical and probabilistic reasoning, game playing, constraint satisfaction, and combinatorial optimization. While it is often straightforward to develop optimal problem solvers for these problems using brute-force, exponential-time search procedures, it is generally much more difficult to develop solvers that are efficient across a wide range of problem instances. The main motivation behind speedup learning is to create adaptive problem solvers that can learn patterns from problem solving experience that can be exploited for efficiency gains. Such adaptive solvers have the potential to significantly outperform traditional static solvers by specializing their behavior to the characteristics of a single problem instance or to an entire class of related problem instances. The exact form of knowledge and learning mechanism is tightly tied to the problem class and the problem-solver architecture.

Most branches of machine learning, such as ►supervised classification, aim to learn fundamentally new problem solving capabilities that are not easily programmed by hand even when ignoring efficiency issues – for example, learning to recognize hand-written digits. Speedup learning is distinct in that it is typically applied in situations where hand-coding an optimal, but inefficient, problem solver is straightforward – for example, solving satisfiability problems. Rather, learning is aimed exclusively at finding solutions in a more practical time frame.

Work in speedup learning grew out of various subfields of artificial intelligence, and more generally computer science. An early example, from automated planning involved learning knowledge for speeding up the original STRIPS planner Fikes, Hart, and Nilsson (1972) via the learning of triangle tables or macros that could later be exploited by the problem solver. Throughout the 1980s and early 1990s, there was a great deal of

additional work on speedup learning in the area of automated planning as overviewed in Minton (1993) and Zimmerman and Kambhampati (2003).

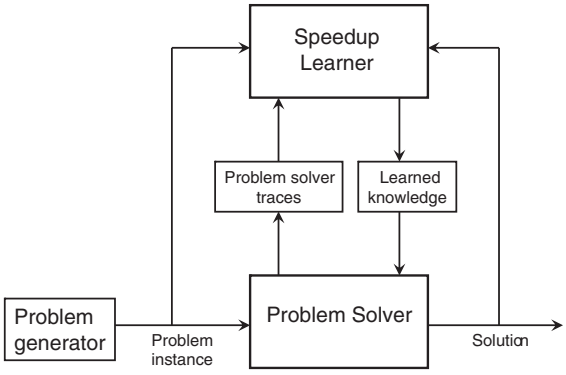
Another major source of speedup learning research has originated from the areas of AI search and constraint satisfaction. Many of the ▶intelligent backtracking mechanisms from these areas, which are critical to perform, can be viewed as speedup learning techniques Kambhampati (1998) where knowledge is learned, while solving a problem instance that better informs later search decisions. Such methods have also come out of the area of logic programming Kumar and Lin (1988), where search efficiency plays a central role.

In addition, various branches of AI have developed speedup-learning approaches based on learning improved heuristic evaluation functions. Samuel's checker player Samuel (1959) was one such early examples, where learned evaluation functions allowed for the performance of deep game tree search to be approximated by shallower, a less expensive search.

Structure of Learning System

Figure 1 shows a generic diagram of a speedup learning system. The main components are the problem solver and the speedup learner. The role of the problem solver is to receive problem instances from a problem generator and to produce solutions for those instances. For example, problem solvers might include constraint-satisfaction engines, automated planners, or A^* search. The role of the speedup learner is to produce knowledge that the problem solver can use to improve its solution time. The input to the speedup learner, which is analyzed in order to produce the knowledge, can include one or more of the following data sources: (1) the input problem instances, (2) traces of the problem solver's decisions while solving the input problems, and (3) solutions to solved problems.

Clearly there is a large space of possible speedup learning systems that result from different problem solvers, forms of learned knowledge, learning methods, and intended mode of applicability. Some of the main dimensions are described in the following section along which speedup learning approaches can be characterized. Examples of typical learners that span this space are provided, noting that the examples are far from an exhaustive list.



Speedup Learning. Figure 1. Schematic diagram of a speedup learning system. The problem solver receives problem instances from a problem generator and produces solutions. The speedup learner can observe the input problem instances, traces of the problem solver while solving the problem instances, and sometimes also the solutions to previously solved problem instances. The speedup learner outputs knowledge that can be used by the problem solver to speedup its solution time either on the current problem instance (intra-problem speedup) and/or future related instances (inter-problem speedup)

Dimensions of Speedup Learning

Intra-Problem versus Inter-Problem Speedup. Intra-problem speedup learning is when knowledge is learned during the solution of the current problem instance and is only applicable to speeding up the solution of the current instance. After a solution is found, the knowledge is discarded as it is not applicable for the future instances. Inter-problem speedup learning is when the learned knowledge is applicable not only to the problem(s) it was learned on but also to new problems to be encountered in the future. In this sense, the learned knowledge can be viewed as a generalized knowledge about how to find solutions more quickly for an entire class of problems.

Typically in the inter-problem learning, the problem generator produces instances that are related in some way, and, thus, share common structure that can be learned from the earlier instances and exploited when solving the later instances. Rather intra-problem speedup learners treat each problem instance as completely distinct from the rest. Also note that inter-problem learners have the potential to benefit from

the analysis of solutions to previous problem instances. Rather, intra-problem learners are unable to use this source of information, since, once the current problem is solved, no further learning is warranted.

Types of Learned Knowledge. Most problem solvers can be viewed as search procedures, which is the view that will be taken when characterizing the various forms of learned knowledge in speedup learning. Four types of commonly used knowledge are listed below, noting that this is far from an exhaustive list. First, *pruning constraints* are the sets of constraints on search nodes that signal when certain branch of the search space can be safely pruned. Second, *macro operators* (macros) are sequences of search operators that are typically useful when executed in order. Problem solvers can often utilize macros in order to decrease the effective solution depth of the search space by treating macros as additional search operators. It is important that the decrease in effective depth is enough to compensate for the increase in number of operators, which increases the search complexity. Third, *search-control rules* are the sets of rules that typically test the current problem solving state and suggest problem-solving actions such as rejecting, selecting, or preferring a particular search operator. In the extreme case, learned search control rules can completely remove the need for search. Fourth, *heuristic evaluation functions* are used to measure the quality of a particular search node. Learning-improved heuristics can result in better directed search behavior.

Deductive versus Inductive Learning. ►Deductive learning refers to a learning process for which the learned knowledge can be deductively proven to be correct. For example, in the case of learned pruning constraints, a deductive learning mechanism would provide a guarantee that the pruning was sound in the sense that the optimality of the problem solver would be unaffected. ►Inductive learning mechanisms rather are statistical in nature and typically do not produce knowledge with associated deductive guarantees. Rather, inductive methods focus on finding statistical regularities that are typically useful, though perhaps not correct in all cases. For example, an inductive learner may discover patterns that are strongly correlated to pruning opportunities, though these patterns may have a small probability of leading to unsound pruning.

In cases where one must guarantee a sound and complete problem solver, deductive learning approaches are always applicable, though their utility depends on the particular application. In certain cases, inductively-learned knowledge can also be utilized in a way that does not effect the correctness of the problem solver. For example, inductively learned search-control rules that assert preferences, rather than prune nodes from the search, do not lead to incompleteness. Traditionally, the primary disadvantage of deductive learning, compared with inductive learning, is that the inductive methods typically produce knowledge that generalizes to a wider range of situations than deductive methods. In addition, deductive learning methods are often more costly in terms of learning time as they rely on expensive deductive reasoning mechanisms. Naturally, a number of speedup learning systems exist that utilize a combination of inductive and deductive learning techniques.

Examples of Intra-Problem Speedup Learning

Much of the speedup learning work arising from research in AI search and constraint satisfaction falls into the intra-problem paradigm. The most common forms of learning are deductive and are based on computing explanations of "search failures" that occur during the solution of a particular problem. Here a search failure typically corresponds to a point where the problem solver must backtrack. By computing and forming such failure explanations the problem solver is typically able to avoid similar types of failures in the future by detecting that a search path will lead to failure without fully exploring that path. ►Nogood learning is a very successful, and commonly used, example of the general failure-explanation approach Schiex and Verfaillie (1994). Nogoods are combinations of variable values that lead to search failures. By computing and recording nogoods, it is possible to immediately prune search states that consider those value combinations. There are many variations of nogood learning, with different techniques utilizing different approaches to analyzing search failures to extract general nogoods.

Another example of the failure-explanation approach, which is commonly utilized in satisfiability solvers, is ►clause learning. The idea is similar to

nogood learning. When a failure occurs during the systematic search, a proof of the failure is constructed and analyzed to extract implied constraints, or clauses, that the solution must satisfy. These learned clauses are then added to the set of clauses of the original satisfiability problem and in later search trigger early pruning when they, or their consequences, are violated. Efficient implementations of this idea have lead to huge gains in satisfiability solvers. In addition, it has been shown theoretically that clause learning can improve solution times by an exponential factor Beame and Sabharwal (2004).

Inductive techniques for learning heuristic evaluation functions have also been investigated in the intra-problem speedup paradigm. Here we discuss just two such approaches, where in both cases the key idea is to observe the problem solver and extract training examples that can be used to learn an accurate evaluation function. A particularly successful example of this approach is the STAGE system Boyan and Moore (1998) for solving combinatorial optimization problems such as traveling salesman and circuit layout. The problem solving architecture used by STAGE is based on repeated random restarts of a fast hill-climbing local optimizer, which when given an initial configuration of the combinatorial object, performs a greedy search to a local minimum configuration. The speedup learning mechanism for STAGE is to learn an approximate function that maps initial configurations to the performance of the local optimizer when started at that configuration. Note that on each restart of the problem solver the learning component gets a training example that can be used to improve the function. The problem solver uses the learned function in order to select promising configurations from which to restart, rather than choosing randomly. In particular, STAGE attempts to restart from a configuration that optimizes the learned function, which is the predicted best starting point for the hill-climber. This overall approach has shown impressive performance gains in a number of combinatorial optimization domains.

As a second example of inductive learning of heuristics in the intra-problem paradigm, there has been work within the more traditional problem solving paradigm of best-first search Sarkar, Chakrabarti, and Ghose (1998). Here the speedup learner observes the sequence of search nodes traversed by the problem solver. For any pair of nodes observed to be on the same search path, the learner creates a training example in an attempt to train a heuristic to better predict the distance between those two nodes. Ideally, this updated heuristic function better reflects the distance from nodes in the search queue to the goal node of the current problem instance, and, hence, result in improved search performance.

Examples of Inter-Problem Speedup Learning

Much of the work on inter-problem speedup learning came out of AI planning research, where researchers have long studied learning approaches for speeding up planners. speedup in planning is focused in this chapter, noting that similar ideas have also been pursued in other research areas such as constraint satisfaction. For a collection and survey of work on speedup in planning see Minton (1993) and Zimmerman and Kambhampati (2003). Typically in this work, one is interested in learning knowledge for an entire planning domain, which is a collection of problems that share the same set of actions. The Blocksworld is a classic example of such a planning domain. After experiencing and solving a number of problems from a target domain, such as the Blocksworld, the learned knowledge is then used to speed up performance on new problems from the same domain.

There have been a number of deductive learning approaches to speedup learning in planning, which are traditionally cited as ▶explanation-based learning (EBL) approaches Minton et al. (1989). EBL for AI planning is strongly related to the failure-explanation approaches developed for CSPs as characterized nicely by Kambhampati (1998). There are two main differences between the inter-problem EBL work in planning and the intra-problem EBL approaches for CSPs. First, EBL approaches in planning produce more general explanations that are applicable not only in the problem in which they were learned, but also new problems. This is often made possible by introducing variables in the place of specific objects into the explanations derived from a particular problem. This allows the explanations to apply to contexts in new problems that share similar structure but involve different objects. The second difference is that inter-problem EBL approaches

in planning often produce explanations of successes and not just of failures. These positive explanations are not possible in the context of intra-problem speedup since the intra-problem learner is only interested in solving a single problem.

Despite the relatively large effort invested in inter-problem EBL research, the best approaches typically did not consistently lead to significant gains, and even hurt performance in many cases. A primary way that EBL can hurt performance is by learning too many explanations, which results in the problem solver spending too much time simply evaluating the explanations at the cost of reducing the number of search nodes considered. This problem is commonly referred to as the EBL utility problem Minton (1988) as it is difficult to determine which explanations have high enough utility to be worth keeping.

In addition to EBL, there has also been work on inductive mechanisms for acquiring search-control rules to speedup AI planners. Typically, statistical learning mechanisms are used to find common patterns that can distinguish between good and bad search decisions. As one example, Huang et al. learn action-rejection and selection rules based on the solutions to planning problems from a common domain Huang, Selman, and Kautz (2000). The learned rules were then added as constraints to the constraint satisfaction engine, which served to guide the solver to solution plans more quickly. Another approach, which has been studied at a theoretical and empirical level, is to learn heuristic functions to guide a bounded search process Xu, Fern (2009), in particular, bread-first beam search. Results in a number of planning domains demonstrate significant improvements over planners that do not incorporate a learning component. One other class of approach is based on attempting to learn knowledge that removes the need for a problem solver altogether. In particular, to learn a reactive policy for quickly selecting actions in any given state of the environment. Such policies can be learned via statistical techniques by simply trying to learn an efficient function that maps planning states to the actions selected by the planner. Despite its simplicity, this approach has demonstrated considerable success Khardon (1999) and has also been characterized at a theoretical level Tadepalli and Natarajan (1996).

Cross References

►Explanation-Based Learning

Recommended Reading

Beame, P., Kautz, H., & Sabharwal, A. (2004). Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, *22*, 319–351.

Boyan, J. A., & Moore, A. W. (1998). Learning evaluation functions for global optimization and boolean satisfiability. In *National conference on artificial intelligence* (pp. 3–10). Mlenio Park, CA: AAAI Press.

Fikes, R., Hart, P., & Nilsson, N. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*(1–3), 251–288.

Huang, Y.-C., Selman, B., & Kautz, H. (2000). Learning declarative control rules for constraint-based planning. In *International conference on machine learning* (pp. 415–422). San Francisco: Morgan Kaufmann.

Kambhampati, S. (1998). On the relations between intelligent backtracking and failure-driven explanation-based learning in constraint satisfaction and planning. *Artificial Intelligence*, *105*(1-2), 161–208.

Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*, *113*(1-2), 125–148.

Kumar, V., & Lin, Y. (1988). A data-dependency based intelligent backtracking scheme for prolog. *The Journal of Logic Programming*, *5*(2), 165–181.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *National conference on artificial intelligence* (pp. 564–569). St. Paul, MN: Morgan Kaufmann.

Minton, S. (Ed.) (1993). *Machine learning methods for planning*. San Francisco: Morgan Kaufmann.

Minton, S., Carbonell, J., Knoblock, C. A., Kuokka, D. R., Etzioni, O., & Gil, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, *40*, 63–118.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 211–229.

Sarkar, S., Chakrabarti, P., & Ghose, S. (1998). Learning whiles solving problems in best first search. *IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans*, *28*(4), 553–541.

Schiex, T., & Verfaillie, G. (1994). Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools*, *3*(2), 187–207.

Tadepalli, P., & Natarajan, B. (1996). A formal framework for speedup learning from problems and solutions. *Journal of Artificial Intelligence Research*, *4*, 445–475.

Zimmerman, T., & Kambhampati, S. (2003). Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, *24*(2), 73–96.

Speedup Learning For Planning

►Explanation-Based Learning for Planning

Spike-Timing-Dependent Plasticity

A biological form of Hebbian learning where the change of synaptic weights depends on the exact timing of presynaptic and postsynaptic action potentials.

Cross References

►Biological Learning: Synaptic Plasticity
►Hebb Rule
►Spike Timing Dependent Plasticity

Sponsored Search

►Text Mining for Advertising

Squared Error

►Error Squared

Squared Error Loss

►Mean Squared Error

Stacked Generalization

Synonyms

Stacking

Definition

Stacking is an ►ensemble learning technique. A set of models are constructed from bootstrap samples of a dataset, then their outputs on a hold-out dataset are used as *input* to a "meta"-model. The set of base models are called *level*-0, and the meta-model *level*-1. The task of the level-1 model is to combine the set of outputs so as to correctly classify the target, thereby correcting any mistakes made by the level-0 models.

Recommended Reading

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks* 5(2), 241–259.

Stacking

►Stacked Generalization

Starting Clause

►Bottom Clause

State

In a ►*Markov decision process*, *states* represent the possible system configurations facing the decision-maker at each *decision epoch*. They must contain all variable information relevant to the decision-making process.

Statistical Learning

►Inductive Learning

Statistical Machine Translation

Miles Osborne
University of Edinburgh, Edinburgh, UK

Synonyms

SMT

Definition

Statistical machine translation (SMT) deals with automatically mapping sentences in one human language (for example, French) into another human language (such as English). The first language is called the *source* and the second language is called the *target*. This process can be thought of as a stochastic process. There are many SMT variants, depending upon how translation is modeled. Some approaches are in terms of

a string-to-string mapping, some use trees-to-strings, and some use tree-to-tree models. All share in common the central idea that translation is automatic, with models estimated from parallel corpora (source-target pairs) and also from monolingual corpora (examples of target sentences).

Motivation and Background

Machine Translation has widespread commercial, military, and political applications. For example, increasingly, the Web is accessed by non-English speakers reading non-English pages. The ability to find relevant information clearly should not be bounded by our language-speaking capabilities. Furthermore, we may not have sufficient linguists in some language of interest to cope with the sheer volume of documents that we would like translated. Enter automatic translation. Machine translation poses a number of interesting machine learning challenges: data sets are typically very large, as are the associated models; the training material used is often noisy and plagued with sparse statistics; the search space of possible translations is sufficiently large that exhaustive search is not possible. Advances in machine learning, such as maximum-margin methods, frequently appear in translation research. SMT systems are now sufficiently mature that they can be deployed in production systems. A good example of this is Google's online Arabic-English translation, which is based upon SMT techniques.

Structure of the Learning System

Modeling

Formally, translation can be described as finding the most likely target sentence e^* for some source sentence f:

$$e^* = \text{argmax}_e P(f \mid e)P(e)$$

(e conventionally stands for English and f for French, but any language pairs can be substituted.)

This approach has three major aspects:

- A translation model ($P(f \mid e)$), which specifies the set of possible translations for some target sentence. The translation model also assigns probabilities to these translations, representing their relative correctness.
- A ▶language model ($P(e)$), which models the fluency of the proposed target sentence. This assigns a distribution over strings, with higher probabilities being assigned to sentences which are more representative of natural language. Language models are usually smoothed n-gram models, typically conditioning on two (or more) previous words when predicting the probability of the current word.
- A search process (the argmax operation), which is concerned with navigating through the space of possible target translations. This is called *decoding*. Decoding for SMT is NP-hard, so most approaches use a beam search.

This is called the *Source-Channel* approach to translation (Brown, Pietra, Pietra, & Mercer, 1994). Most modern SMT systems instead use a ▶log-linear model, as it is more flexible and allows for various aspects of translation to be balanced together (Och & Ney, 2001):

$$e^* = \text{argmax}_e \left(\sum_i f_i(e,f)\lambda_i \right)$$

Here, feature functions $f_i(e,f)$ capture some aspect of translation and each feature function has an associated weight λ_i. When we have the two feature functions $P(f \mid e)$ and $P(e)$, we have the Source-Channel model. The weights are scaling factors (balancing the contributions that each feature function makes) and are optimized with respect to some ▶loss function which evaluates translation quality. Frequently, this is in terms of the *BLEU* evaluation metric Papineni, Roukos, Ward, & Zhu (2001). Typically, the error surface is nonconvex and the loss function is nondifferentiable, so search techniques which do not use first-order derivatives must be employed. It is worth noting that machine translation evaluation is a complex problem and that methods such as BLEU are not without criticism.

SMT systems usually decompose entire sentences into a sequence of strings called *phrases* (Koehn, Och, & Marcu, 2003). The modeling task then becomes one of determining how to break a source sentence into a sequence of contiguous phrases and how to specify which source phrase should be associated with each target phrase. Figure 1 shows an example English-French sentence pair. Figure 2 shows that sentence pair decomposed into phrase-pairs. Phrase-based systems

Those people have grown up, lived and worked for many years in a farming district.
Ces gens ont grandi, vécu et oeuvré des dizaines d'années dans le domain agricole.

Statistical Machine Translation. Figure 1. A sentence pair

Ces gens ont	Those people have
gens ont grandi	people have grown up
ont grandi ,	have grown up ,
grandi , vécu	grown up , lived
, vécu et	, lived and
vécu et oeuvré	lived and worked
et oeuvré des dizaines d' oeuvré	and worked many
oeuvré des dizaines d' années dizaines	worked many years
des dizaines d' années dans	many years in
années dans le	years in a
le domaine agricole	a farming districtle
domaine agricole .	farming district .

Statistical Machine Translation. Figure 2. Example phrase pairs

represented an advance over previous word-based models, since phrase-based translation can capture local (within a phrase) word order. Furthermore, phrase-based translation approaches need to make fewer decisions than word-based models. This means there are fewer errors to make.

A major aspect of any SMT approach is dealing with phrasal *reordering*. Typically, the translation of each source phrase need not follow the same temporal order in the target sentence. Simple approaches model the absolute distance a target phrase can "move" from the originating target phrase. More sophisticated reordering models condition this movement upon the aspects of the phrase pair.

Our description of SMT is in terms of a string-to-string model. There are numerous other SMT approaches, for example those which use notions of syntax (Chiang, 2005). These models are now showing promising results, but are significantly more complex to describe.

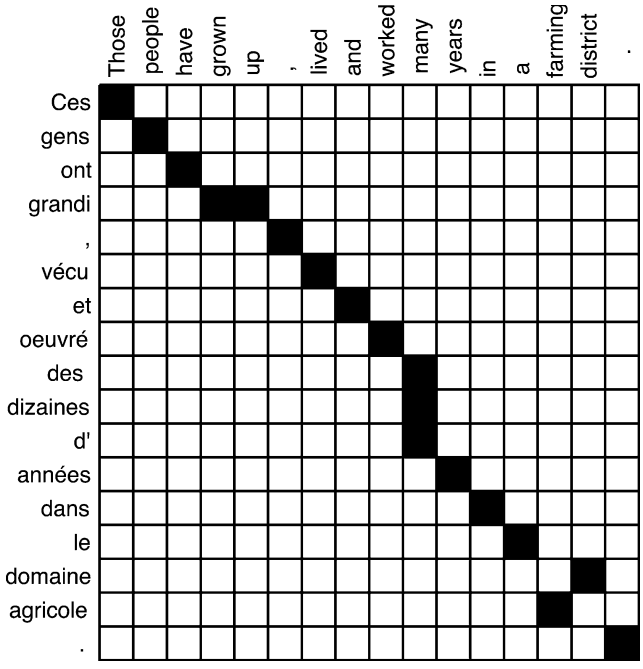
Estimation

The translation model of a SMT system is estimated using *parallel corpora*. Because the search space is so large and that parallel corpora is not aligned at the word level, the estimation process is based upon a large-scale application of Expectation-Maximization, along with heuristics. This consists of the following steps:

- Determine how each source word translates to zero or more target words. The IBM models are used for this task, which are based upon the Expectation-Maximization algorithm for parameter estimation (Brown et al., 1994).
- Repeat this process, but instead determine how each target word translates to zero or more source words.
- Harmonize the previous two steps, creating a set of *word alignments* for each sentence pair. This process is designed to use the two directions as alternative views on how words should be translated. Figure 3 shows the sentence pair aligned at the word level.
- Heuristically, determine which sequence of source words translates to a sequence of target words. This produces a set of *phrase-pairs*: a snippet of text in the source sentence and the associated snippet of text in the target sentence.
- Relative frequency estimators can then be used to characterize how each source phrase translates to a given target phrase.

Parallel corpora varies in size tremendously; for language pairs such as Arabic to English, we have on the order of ten million sentence pairs. Most other language pairs (for example, Finnish to Irish) will have far smaller parallel corpora available. Parallel corpora exists for all European languages and for many other pairs, such as Mandarin to English.

The language model is instead estimated from monolingual corpora, typically using relative frequency estimates, which are then smoothed. For languages such as English, typically billion (and more) words are used. Deploying such large models can pose significant engineering challenges. This is because the language model can easily be so large that it will not fit into the memory



Statistical Machine Translation. Figure 3. The sentence pair in Fig. 1 aligned at the word-level

of conventional machines. Also, the language model can be queried millions of times when translating sentences, which precludes storing it on disk.

Programs and Data

All of the code and data necessary to begin work on SMT is available either as public source, or for a small payment (in the case of corpora from the LDC):

- The standard software to estimate word-based translation models is Giza++: http://www.fjoch.com/GIZA++.html
- Converting word-based to phrase-based models and decoding can be achieved using the Moses decoder and associated sets of scripts: http://www.statmt.org/jhuws/?n=Moses.HomePage
- Translation performance can be evaluated using BLEU: http://www.nist.gov/speech/tests/mt/resources/scoring.htm
- The SRILM is the standard toolkit for building and using language models: http://www.speech.sri.com/projects/srilm/
- Europarl is a set of parallel corpora, dealing with European languages: http://www.statmt.org/europarl/
- The Linguistics Data Consortium (LDC) maintains corpora of various kinds, including large volumes of monolingual data which can be used to train language models: http://www.ldc.upenn.edu/

Recommended Reading

Brown, P. F., Pietra, S. D., Pietra, V. J. D., & Mercer, R. L. (1994). The mathematic of statistical machine translation: Parameter estimation. *Computational Linguistics*, *19*(2), 263–311.

Chiang, D. (2005, June). A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL'05)* (pp. 263–270). Ann Arbor, MI: Association for Computational Linguistics.

Koehn, P., Och, F. J., & Marcu, D. (2003). Statistical phrase-based translation. In *NAACL '03: Proceedings of the 2003 conference of the north american chapter of the association for computational linguistics on human language technology* (pp. 48–54). Morristown, NJ: Association for Computational Linguistics.

Och, F. J., & Ney, H. (2001). Discriminative training and maximum entropy models for statistical machine translation. In *ACL '02: Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 295–302). Morristown, NJ: Association for Computational Linguistics.

Papineni, K., Roukos, S., Ward, T., & Zhu, W. -J. (2001). Bleu: A method for automatic evaluation of machine translation. In *ACL '02: Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 311–318). Morristown, NJ: Association for Computational Linguistics.

Statistical Natural Language Processing

►Maximum Entropy Models for Natural Language Processing

Statistical Physics Of Learning

►Phase Transitions in Machine Learning

Statistical Relational Learning

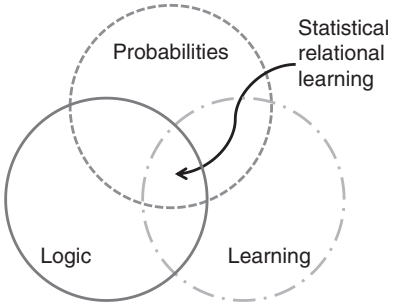
Luc De Raedt[1], Kristian Kersting[2]
[1]Katholieke Universiteit Leuven, Heverlee, Belgium
[2]Knowledge Discovery, Fraunhofer IAIS, Sankt Augustin, Germany

Definition

Statistical relational learning a.k.a. probabilistic inductive logic programming deals with machine learning and data mining in relational domains where observations may be missing, partially observed, or noisy. In doing so, it addresses one of the central questions of artificial intelligence – the integration of probabilistic reasoning with machine learning and first-order and relational representations – and deals with all related aspects such as reasoning, parameter estimation, and structure learning.

Motivation and Background

One of the central questions of artificial intelligence is concerned with combining expressive knowledge representation formalisms such as relational and first-order logic with principled probabilistic and statistical approaches to inference and learning. While traditionally relational and logical representations, probabilistic and statistical reasoning, and machine learning have been studied independently of one another, statistical relational learning investigates them jointly, cf. Fig. 1. A major driving force is the explosive growth in the amount of heterogeneous data that is being collected in the business and scientific world in domains such as bioinformatics, transportation systems, communication networks, social network analysis, citation analysis, and robotics. Characteristic for these domains is that they provide *uncertain* information about varying numbers of entities and relationships among the entities, that is, about *relational* domains. Traditional machine learning approaches are able to cope either with uncertainty or with relational representations but typically not with both.



Statistical Relational Learning. Figure 1. Statistical relational learning a.k.a. probabilistic inductive logic programming combines probability, logic, and learning

Many formalisms and representations have been developed in statistical relational learning. For instance, Eisele (1994) has introduced a probabilistic variant of comprehensive unification formalism (CUF). In a similar manner, Muggleton (1996) and Cussens (1999) have upgraded stochastic grammars toward *stochastic logic programs*. Sato (1995) has introduced *probabilistic distributional semantics* for logic programs. Taskar, Abbeel, and Koller (2002) have upgraded Markov networks toward *relational Markov networks*, and Richardson and Domingos (2006) toward *Markov logic networks*. Neville and Jensen (2004) have extended dependency networks toward *relational dependency networks*. Another research stream has investigated logical and relational extensions of Bayesian networks. It includes Poole's *independent choice logic* (Poole, 1993), Ngo and Haddawy's *probabilistic logic programs* (Ngo & Haddawy, 1997), Jäger's *relational Bayesian networks* (Jager, 1997), Koller, Getoor, and Pfeffer's *probabilistic relational models* (Getoor, 2001; Pfeffer 2000), and Kersting and De Raedt's *Bayesian logic programs* (Kersting & De Raedt, 2007).

The benefits of employing logical abstraction and relations within statistical learning are manyfold:

1. Relations among entities allow one to use information about one entity to help reach conclusions about other, related entities.
2. Variables, that is, placeholders for entities allow one to make abstraction of specific entities.
3. Unification allows one to share information among entities. Thus, instead of learning regularities for each single entity independently, statistical relational learning aims at finding general regularities among groups of entities.
4. The learned knowledge is often declarative and compact, which makes it easier for people to understand and to validate.
5. In many applications, there is a rich background theory available, which can efficiently and elegantly be represented as a set of general regularities. This is important because background knowledge may improve the quality of learning as it focuses the learning on the relevant patterns, that is, it restricts the search space.
6. When learning a model from data, relational and logical abstraction allow one to reuse experience in that *learning about one entity improves the prediction for other entities*; and this may even generalize to objects that have never been observed before.

Thus, relational and logical abstraction make statistical learning more robust and efficient. This has proven to be beneficial in many fascinating real-world applications in citation analysis, web mining, natural language processing, robotics, bio- and chemo-informatics, electronic games, and activity recognition.

Theory

Whereas most of the existing works on statistical relational learning have started from a statistical and probabilistic learning perspective and extended probabilistic formalisms with relational aspects, statistical relational learning can elegantly be introduced by starting from ►inductive logic programming (De Raedt, 2008; Muggleton & De Raedt, 1994), which is often also called *multi-relational data mining* (MRDM) (Džeroski & Lavrač, 2001). Inductive logic programming is a research field at the intersection of machine learning and logic programming. It forms a formal framework and has introduced practical algorithms for inductively learning relational descriptions (in the form of logic programs) from examples and background knowledge. So, the only difference to statistical relational learning is that it does not explicitly deal with uncertainty.

Essentially, there are only two changes to apply to inductive logic programming approaches in order to arrive at statistical relational learning:

1. ►clauses (i.e., logical formulae that can be interpreted as rules; cf. below) are annotated with probabilistic information such as conditional probabilities; and
2. the ►covers relation (which states the conditions under which a hypothesis considers an example as positive) becomes probabilistic.

A probabilistic covers relation softens the hard covers relation employed in traditional inductive logic programming and is defined as the probability of an example given the hypothesis and the background theory.

Definition 1 (**Probabilistic Covers Relation**). *A probabilistic covers relation takes as arguments an example e, a hypothesis H and possibly the background theory B, and returns the probability value* $\mathbf{P}(e \mid H, B)$ *between* 0 *and* 1 *of the example e given H and B, that is,* $covers(e, H, B) = \mathbf{P}(e \mid H, B)$.

It specifies the likelihood of the example given the hypothesis and the background theory. Different choices of the probabilistic covers relation lead to different statistical relational learning approaches; this is akin to the learning settings in inductive logic programming.

Statistical Relational Languages

There is a multitude of different languages and formalisms for statistical relational learning. For an overview of these languages we refer to (Getoor & Taskar, 2007) and (De Raedt, Frasconi, Kersting, & Muggleton, 2008). Here, we choose two formalisms that are representatives of the two main streams in statistical relational learning. First, we discuss Markov logic (Richardson & Domingos, 2006), which upgrades Markov network toward first-order logic, and second,

we discuss ProbLog (De Raedt, Kimmig, & Toivonen, 2007), which is a probabilistic Prolog based on Sato's distribution semantics (Sato, 1995). While Markov logic is a typical example of knowledge-based model construction, ProbLog is a probabilistic programming language.

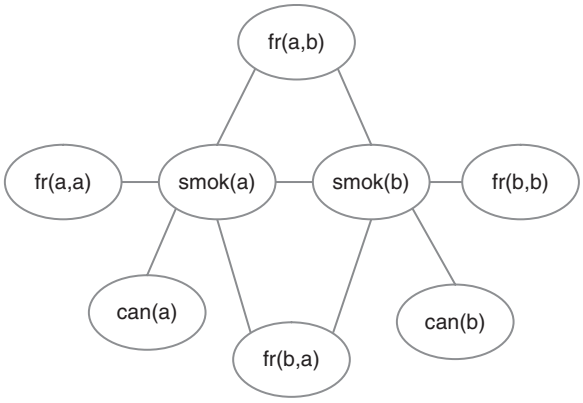
Case Study: Markov Logic Networks Markov logic combines first-order logic with ▶Markov networks. The idea is to view logical formulae as soft constraints on the set of possible worlds, that is, on the ▶interpretations (an interpretation is a set of facts). If an interpretation does not satisfy a logical formula, it becomes less probable, but not necessarily impossible as in traditional logic. Hence, the more formulae an interpretation satisfies, the more likely it becomes. In a Markov logic network, this is realized by associating a weight to each formula that reflects how strong the constraint is. More precisely, a Markov logic network consists of a set of weighted clauses $H = \{c_1, \ldots, c_m\}$. (Markov logic networks, in principle, also allow one to use arbitrary logical formulae, not just clauses. However, for reasons of simplicity, we only employ clauses and make some further simplifications.) The weights w_i of the clauses then specify the strength of the clausal constraint.

Example 1 *Consider the following example (adapted from Richardson and Domingos (2006)). Friends & Smokers is a small Markov logic network that computes the probability of a person having lung cancer on the basis of her friends smoking. This can be encoded using the following weighted clauses:*

1.5: cancer(P) ← smoking(P)
1.1: smoking(X) ← friends(X,Y), smoking(Y)
1.1: smoking(Y) ← friends(X,Y), smoking(X)

The first clause states the soft constraint that smoking causes cancer. So, interpretations in which persons that smoke have cancer are more likely than those where they do not (under the assumptions that other properties remain constant). The second and third clauses state that friends of smokers are typically also smokers.

A Markov logic network together with a Herbrand domain (in the form of a set of constants $\{d_1, \ldots, d_k\}$) then induces a grounded Markov network, which defines a probability distribution over the possible Herbrand interpretations.



Statistical Relational Learning. Figure 2. The Markov network for the constants ann and bob. Adapted from Richardson and Domingos (2006)

The nodes, that is, the random variables in the grounded network, are the atoms in the Herbrand base, that is, the facts of the form $p(d'_1, \ldots, d'_n)$ where p is a predicate or relation and the d'_i are constants. Furthermore, for every ground instance $c_i\theta$ of a clause c_i in H, there will be an edge between any pair of atoms $a\theta, b\theta$ that occurs in $c_i\theta$. The Markov network obtained for the constants anna and bob is shown in Fig. 2. To obtain a probability distribution over the Herbrand interpretations, we still need to define the potentials. The probability distribution over interpretations I is

$$\mathbf{P}(I) = \frac{1}{Z} \prod_{c:clause} \mathbf{f}_c(I) \tag{1}$$

where the f_c are defined as

$$f_c(I) = e^{n_c(I)w_c} \tag{2}$$

and $n_c(I)$ denotes the number of substitutions θ for which $c\theta$ is satisfied by I, and Z is a normalization constant. The definition of a potential as an exponential function of a weighted feature of a clique is common in Markov networks; cf. ▶graphical models. The reason is that the resulting probability distribution is easier to manipulate.

Note that for different (Herbrand) domains, different Markov networks will be produced. Therefore, one can view Markov logic networks as a kind of

template for generating Markov networks, and, hence, Markov logic is based on knowledge-based model construction. Notice also that Markov logic networks define a probability distribution over interpretations, and nicely separate the qualitative from the quantitative component.

Case Study: ProbLog Many formalisms do not explicitly encode a set of conditional independency assumptions, as in Bayesian or Markov networks, but rather extend a (logic) programming language with probabilistic choices. Stochastic logic programs (Cussens, 2001; Muggleton, 1996) directly upgrade stochastic context-free grammars toward definite clause logic, whereas PRISM (Sato, 1995), probabilistic Horn abduction (PHA) (Poole, 1993), and the more recent independent choice logic (ICL) (Poole, 1997) specify probabilities on facts from which further knowledge can be deduced. As a simple representative of this stream of work, we introduce the probabilistic Prolog called ProbLog (De Raedt et al., 2007).

The key idea underlying Problog is that some facts f for *probabilistic* predicates are annotated with a probability value. This value indicates the degree of belief, that is the probability, that any ground instance $f\theta$ of f is true. It is also assumed that the $f\theta$ are marginally independent. The probabilistic facts are then augmented with a set of definite clauses defining further predicates (which should be disjoint from the probabilistic ones). An example adapted from De Raedt et al. (2007) is given below.

Example 2 *Consider the facts*

0.9: edge(a,c) ←
0.7: edge(c,b) ←
0.6: edge(d,c) ←
0.9: edge(d,b) ←

which specify that with probability 0.9 *there is an edge from* a *to* c. *Consider also the following (simplified) definition of* path/2.

path(X,Y)edge(X,Y) ←
path(X,Y)edge(X,Z), path(Z,Y) ←

One can now define a probability distribution on (ground) proofs as follows. The probability of a ground proof is the product of the probabilities of the (ground) clauses (here, facts) used in the proof. For instance, the only proof for the goal ← path(a,b) employs the facts edge(a,c) and edge(c,b); these facts are marginally independent, and hence the probability of the proof is 0.9×0.7. The probabilistic facts used in a single proof are sometimes called an *explanation*.

It is now tempting to define the probability of a ground atom as the sum of the probabilities of the proofs for that atom. However, this does not work without additional restrictions, as shown in the following example.

Example 3 *The fact* path(d,b) *has two explanations:*

1. *{*edge(d,c), edge(c,b)*} with probability* $0.6 \times 0.7 = 0.42$, *and*
2. *{*edge(d,b)*} with probability* 0.9.

Summing the probabilities of these explanations gives a value of 1.32, which is clearly impossible.

The reason for this problem is that the different explanations are not mutually exclusive, and therefore their probabilities may not be summed. The probability $P(\text{path(d,b)} = \textit{true})$ is, however, equal to the probability that *a* proof succeeds, that is,

$$P(\text{path(d,b)} = \textit{true}) = P[(\text{e(d,c)} \wedge \text{e(c,b)}) \vee \text{e(d,b)}]$$

which shows that computing the probability of a derived ground fact reduces to computing the probability of a boolean formula in disjunctive normal form (DNF), where all random variables are marginally independent of one another. Computing the probability of such formulae is an NP-hard problem, the *disjoint-sum* problem. Using the *inclusion-exclusion* principle from set theory, one can compute the probability as

$$\begin{aligned} P(\text{path(d,b)} = \textit{true}) &= P[(\text{e(d,c)} \wedge \text{e(c,b)}) \vee \text{e(d,b)}] \\ &= P(\text{e(d,c)} \wedge \text{e(c,b)}) \\ &\quad + P(\text{e(d,b)}) \\ &\quad - P((\text{e(d,c)} \wedge \text{e(c,b)}) \\ &\quad \wedge \text{e(d,b)}) \\ &= 0.6 \times 0.7 + 0.9 - 0.6 \times 0.7 \\ &\quad \times 0.9 = 0.942 \end{aligned}$$

There exist more effective ways to compute the probability of such DNF formulae (De Raedt et al., 2007), where binary decision diagrams are employed to represent the DNF formulae.

The above example shows how the probability of a specific fact is defined and can be computed. The distribution at the level of individual facts (or goals) can easily be generalized to a possible world semantics, specifying a probability distribution on interpretations. It is formalized in the *distribution semantics* of Sato (1995), which is defined by starting from the set of all probabilistic ground facts F for the given program. For simplicity, we shall assume that this set is finite, though Sato's results also hold for the infinite case. The distribution semantics then starts from a probability distribution $P_F(S)$ defined on subsets $S \subseteq F$:

$$P_F(S) = \prod_{f \in S} P(f) \prod_{f \notin S} (1 - P(f)). \tag{3}$$

Each subset S is now interpreted as a set of logical facts and combined with the definite clause program R that specifies the logical part of the probabilistic logic program. Any such combination $S \cup R$ possesses a unique least Herbrand model $M(S \cup R)$, which corresponds to a possible world. The probability of such a possible world is then the sum of the probabilities of the subsets S yielding that possible world, that is,

$$P_W(M) = \sum_{S \subseteq F : M(S \cup R) = M} P_F(S) \tag{4}$$

For instance, in the path example, there are 16 possible worlds, which can be obtained from the 16 different truth assignments to the facts, and whose probabilities can be computed using Eq. (4). As for graphical models, the probability of any logical formula can be computed from a possible world semantics (specified here by P_W).

Because computing the probability of a fact or goal under the distribution semantics is hard, systems such as PRISM (Sato, 1995) and PHA (Poole, 1993) impose additional restrictions that can be used to improve the efficiency of the inference procedure. The key assumption is that the explanations for a goal are *mutually exclusive*, which overcomes the disjoint-sum problem. If the different explanations of a goal do not overlap, then its probability is simply the sum of the probabilities of its explanations. This directly follows from the inclusion-exclusion formulae as under the exclusive-explanation assumption the conjunctions (or intersections) are empty.

Learning

Essentially, any statistical relational approach can be viewed as lifting a traditional inductive logic programming setting by associating probabilistic information to clauses and by replacing the deterministic coverage relation by a probabilistic one. In contrast to traditional graphical models such as Bayesian networks or Markov networks, however, we can also employ "counterexamples" for learning. Consider a simple kinship domain. Assume `rex` is a male person. Consequently, he cannot be the `daughter` of any other person, say `ann`. Thus, `daughter(rex,ann)` can be listed as a negative example although we will never observe it. "Counterexamples" conflict with the usual view on learning examples in statistical learning.

In statistical learning, we seek to find that hypothesis H^*, which is most likely given the learning examples:

$$H^* = \arg\max_H P(H|E) = \arg\max_H \frac{P(E|H) \cdot P(F)}{P(E)}$$
$$\text{with} \quad P(E) > 0 \,.$$

Thus, examples E in traditional statistical learning are always observable, that is, $P(E) > 0$. However, in statistical relational learning, as in inductive logic programming, we may also employ "counterexamples" such as `daughter(rex,ann)`, which have probability "0," and that actually never can be observed.

Definition 2 (**SRL Problem**). **Given** *a set* $E = E_p \cup E_i$ *of positive and negative examples* E_p *and* E_i *(with* $E_p \cap E_i = \varnothing$*) over some example language* $\mathcal{L}_E$*, a probabilistic covers relation* $covers(e, H, B) = P(e \mid H, B)$*, a probabilistic logical language* $\mathcal{L}_H$ *for hypotheses, and a background theory* B*,* ***find*** *a hypothesis* H^* *in* $\mathcal{L}_H$ *such that* $H^* = \arg\max_H score(E, H, B)$ *and the following constraints hold:* $\forall\ e_p \in E_p : covers(e_p, H^*, B) > 0$ *and* $\forall\ e_i \in E_i : covers(e_i, H^*, B) = 0$*. The score is some objective function, usually involving the probabilistic covers relation of the observed examples such as the observed likelihood* $\prod_{e_p \in E_p} covers(e_p, H^*, B)$ *or some penalized variant thereof.*

This learning setting unifies inductive logic programming and statistical learning in the following sense: using a deterministic covers relation (either 1 or 0), it yields the classical inductive logic programming learning problem; sticking to propositional logic and learning from *positive* examples, that is, $P(E) > 0$, only yields traditional statistical learning.

To come up with algorithms solving the SRL problem, say for density estimation, one typically distinguishes two subtasks because $H = (L, \lambda)$ is essentially a logical theory L annotated with probabilistic parameters λ:

1. *Parameter estimation* where it is assumed that the underlying logic program L is fixed, and the learning task consists of estimating the parameters λ that maximize the likelihood.
2. *Structure learning* where both L and λ have to be learned from the data.

In the following paragraphs, we will sketch the basic parameter estimation and structure learning techniques, and illustrate them for each setting.

Parameter Estimation The problem of parameter estimation is concerned with estimating the values of the parameters λ of a fixed probabilistic program $H = (L, \lambda)$ that best explains the examples E. So, λ is a set of parameters and can be represented as a vector. As already indicated above, to measure the extent to which a model fits the data, one usually employs the likelihood of the data, that is, $P(E \mid L, \lambda)$, though other scores or variants could be used as well.

When all examples are fully observable, maximum likelihood reduces to frequency counting. In the presence of missing data, however, the maximum likelihood estimate typically cannot be written in closed form. It is a numerical optimization problem, and all known algorithms involve nonlinear optimization. The most commonly adopted technique for probabilistic logic learning is the expectation-maximization (EM) algorithm (Dempster, Laird, Rubin, 1977; McLachlan & Krishnan, 1997). EM is based on the observation that learning would be easy (i.e., correspond to frequency counting), if the values of all the random variables would be known. Therefore, it estimates these values, maximizes the likelihood based on the estimates, and then iterates. More specifically, EM assumes that the parameters have been initialized (e.g., at random) and then iteratively performs the following two steps until convergence:

(E-Step) On the basis of the observed data and the present parameters of the model, it computes a distribution over all possible completions of each partially observed data case.

(M-Step) Treating each completion as a fully observed data case weighted by its probability, it computes the improved parameter values using (weighted) frequency counting.

The frequencies over the completions are called the *expected counts*. Examples for parameter estimation of probabilistic relational models can be found in (Getoor & Taskar, 2007) and (De Raedt, Frasconi, Kersting, & Muggleton, 2008).

Structure Learning The problem is now to learn both the structure L and the parameters λ of the probabilistic program $H = (L, \lambda)$ from data. Often, further information is given as well. As in inductive logic programming, the additional knowledge can take various different forms, including a ▶*language bias* that imposes restrictions on the syntax of L, and an *initial hypothesis* (L, λ) from which the learning process can start.

Nearly all (score-based) approaches to structure learning perform a heuristic search through the space of possible hypotheses. Typically, hill-climbing or beam-search is applied until the hypothesis satisfies the logical constraints and the $score(H, E)$ is no longer improving. The steps in the search-space are typically made using refinement operators, which make small, syntactic modifications to the (underlying) logic program.

At this point, it is interesting to observe that the logical constraints often require that the positive examples are covered in the logical sense. For instance, when learning ProbLog programs from entailment, the observed example clauses must be entailed by the logic program. Thus, for a probabilistic program $H = (L_H, \lambda_H)$ and a background theory $B = (L_B, \lambda_B)$ it holds that $\forall e_p \in E_p : P(e|H, B) > 0$ if and only if $covers(e, L_H, L_B) = 1$, where L_H (respectively L_B) is the underlying logic program (logical background theory)

and $covers(e, L_H, L_B)$ is the purely logical *covers* relation, which is either 0 or 1.

Applications

Applications of statistical relational learning can be found in many areas such as web search and mining, text mining, bioinformatics, natural language processing, robotics, and social network analysis, among others. Due to space restrictions, we will only name a few of these exciting applications.

For instance, Getoor, Taskar, & Koller (2001) have used statistical relational models to estimate the result size of complex database queries. Segal et al. have employed probabilistic relational models to cluster gene expression data (Segal, Taskar, Gasch, Friedman, & Koller, 2001) and to discover cellular processes from gene expression data (Segal, Battle, & Koller, 2003). Getoor et al. have used probabilistic relational models to understand tuberculosis epidemiology (Getoor, Rhee, Koller, & Small, 2004). McGovern et al. (2003) have estimated probabilistic relational trees to discover publication patterns in high-energy physics. Probabilistic relational trees have also been used to learn to rank brokers with respect to the probability that they would commit a serious violation of securities regulations in the near future (Neville et al., 2005). Anguelov et al. (2005) have used relational Markov networks for segmentation of 3D scan data. Markov networks have also been used to compactly represent object maps and to estimate trajectories of people (Limketkai, Liao, & Fox, 2005). Kersting et al. have employed relational hidden Markov models for protein fold recognition (Kersting, De Raedt, & Raiko, 2006). Poon and Domingos (2008) have shown how to use Markov logic to perform joint unsupervised coreference resolution. Xu et al. have used nonparametric relational models for analyzing social networks (Xu, Tresp, Rettinger, & Kersting, 2010). Kersting and Xu (2009) have used relational Gaussian processes for learning to rank search results. Recently, Poon and Domingos (2009) have shown how to perform unsupervised semantic parsing using Markov logic networks.

Future Directions

We have provided an overview of the new and exciting area of statistical relational learning. It combines principles of probabilistic reasoning, logical representation, and statistical learning into a coherent whole. The techniques of probabilistic logic learning were analyzed starting from an inductive logic programming perspective by lifting the coverage relation to a probabilistic one and annotating the logical formulae. Different choices of the probabilistic coverage relation lead to different representational formalisms, two of which were introduced.

Statistical relational learning is an active area of research within the machine learning and the artificial intelligence community. First, there is the issue of *efficient inference* and learning. Most current inference algorithms for statistical relational models require explicit state enumeration, which is often impractical: the number of states grows very quickly with the number of domain objects and relations. *Lifted* inference algorithms seek to avoid explicit state enumeration and directly work at the level of groups of atoms, eliminating all the instantiations of a set of atoms in a single step, in some cases independently of the number of these instantiations. Despite various approaches to lifted inference (de Salvo Braz, Amir, & Roth, 2005; Jaimovich, Meshi, & Friedman, 2007; Kersting, Ahmadi, & Natarajan, 2009; Kisynski & Poole, 2009; Milch, Zettlemoyer, Kersting, Haimes, & Kaelbling , 2008; Poole, 2003; Sen, Deshpande, & Getoor, 2008; Singla & Domingos, 2008), it largely remains a challenging problem. For what concerns learning, advanced principles of both statistical learning and logical and relational learning can be employed for learning the parameters and the structure of probabilistic logics such as statistical *predicate invention* (Kok & Domingos, 2007) and *boosting* (Gutmann & Kersting, 2006). Recently, people started to investigate *learning from weighted examples* (see e.g., Chen, Muggleton, & Santos, 2008) and to link statistical relational learning to support vector machines (see e.g., Passerini, Frasconi, & De Raedt, 2006). Second, there is the issue of *closed-world versus open-world* assumption that is, do we know how many objects there are (see e.g., Milch et al., 2005). Third, there is interest in dealing with *continuous values* within statistical relational learning (see e.g., Chu, Sindhwani, Ghahramani, & Keerthi, 2006; Silva, Chu, & Ghahramani, 2007; Wang & Domingos, 2008; Xu, Kersting, & Tresp, 2009). This is mainly motivated by the fact that most real-world applications actually

contain continuous values. *Nonparametric Bayesian* approaches to statistical relational learning have also been developed (see e.g., Kemp, Tenenbaum, Griffiths, Yamada, & Ueda, 2006; Xu, Tresp, Yu, & Kriegel, 2006; Yu & Chu, 2007; Yu, Chu, Yu, Tresp, & Xu, 2006), to overcome the typically strong parametric assumptions underlying current statistical relational learning. People have also started to investigate *relational variants of classical statistical learning tasks* such as matrix factorizations (see e.g., Singh & Gordon, 2008). Finally, while statistical relational learning approaches have been used successfully in a number of applications, they do not yet cope with the *dynamic environments* in an effective way.

Cross References

►Multi-Relational Data Mining
►Relational Learning

Recommended Reading

In addition to the references embedded in the text above, we also recommend De Raedt et al. (2008), Getoor & Taskar (2007), De Raedt (2008) and the SRL tutorials at major artificial intelligence and machine learning conferences.

Anguelov, D., Taskar, B., Chatalbashev, V., Koller, D., Gupta, D., Heitz, G., et al. (2005). Discriminative learning of Markov random fields for segmentation of 3D scan data. In C. Schmid, S. Soatto, & C. Tomasi (Eds.), *IEEE Computer Society international conference on computer vision and pattern recognition (CVPR-05)*, San Diego, CA, USA (Vol. 2, pp. 169–176).

Chen, J., Muggleton, S., & Santos, J. (2008). Learning probabilistic logic models from probabilistic examples. *Machine Learning, 73*(1), 55–85.

Chu, W., Sindhwani, V., Ghahramani, Z., & Keerthi, S. (2006). Relational learning with Gaussian processes. In *Advances in Neural information processing systems 19* (NIPS-2006). Cambridge, MA: MIT Press.

Cussens, J. (1999). Loglinear models for first-order probabilistic reasoning. In K. Blackmond Laskey & H. Prade (Eds.), *Proceedings of the fifteenth annual conference on uncertainty in artificial intelligence (UAI-99)* (pp. 126–133), Stockholm, Sweden. San Francisco: Morgan Kaufmann.

Cussens, J. (2001). Parameter estimation in stochastic logic programs. *Machine Learning Journal, 44*(3), 245–271.

De Raedt, L. (2008). *Logical and relational learning*. Springer.

De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (Eds.). (2008). *Probabilistic inductive logic programming. Lecture notes in computer science* (Vol. 4911). Berlin/Heidelberg: Springer.

De Raedt, L., Kimmig, A., & Toivonen, H. (2007). Problog: A probabilistic Prolog and its application in link discovery. In M. Veloso (Ed.), *Proceedings of the 20th international joint conference on artificial intelligence* (pp. 2462–2467). Hyderabad, India.

de Salvo Braz, R., Amir, E., & Roth, D. (2005). Lifted first order probabilistic inference. In *Proceedings of the 19th international joint conference on artificial intelligence (IJCAI-05)* (pp. 1319–1325). Edinburgh, Scotland.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B, 39*, 1–39.

Džeroski, S., & Lavrač, N. (Eds.). (2001). *Relational data mining*. Berlin: Springer.

Eisele, A. (1994). Towards probabilistic extensions of contraint-based grammars. In J. Dörne (Ed.), *Computational aspects of constraint-based linguistics description-II*. Stuttgart: Institute for Computational Linguistics (IMS-CL). DYNA-2 deliverable R1.2.B.

Getoor, L. (2001). *Learning statistical models from relational data*. PhD thesis, Stanford University, USA.

Getoor, L., Rhee, J., Koller, D., & Small, P. (2004). Understanding tuberculosis epidemiology using probabilistic relational models. *Journal of Artificial Intelligence in Medicine, 30*, 233–256.

Getoor, L., & Taskar, B. (Eds.). (2007). *Introduction to statistical relational learning*. Cambridge, MA, USA: The MIT Press.

Getoor, L., Taskar, B., & Koller, D. (2001). Using probabilistic models for selectivity estimation. In *Proceedings of ACM SIGMOD international conference on management of data* (pp. 461–472). Santa Barbara, CA, USA: ACM Press.

Gutmann, B., & Kersting, K. (2006). TildeCRF: Conditional random fields for logical sequences. In J. Fuernkranz, T. Scheffer, & M. Spiliopoulou (Eds.), *Proceedings of the 17th European conference on machine learning (ECML-2006)*, Berlin, Germany (pp. 174–185).

Jäger, M. (1997). Relational Bayesian networks. In K. Laskey & H. Prade (Eds.), *Proceedings of the thirteenth conference on uncertainty in artificial intelligence (UAI-97)*, Stockholm, Sweden (pp. 266–273). San Franciso, CA, USA: Morgan Kaufmann.

Jaimovich, A., Meshi, O., & Friedman, N. (2007). Template-based inference in symmetric relational Markov random fields. In *Proceedings of the conference on uncertainty in artificial intelligence (UAI-07)* (pp. 191–199).

Kemp, C., Tenenbaum, J., Griffiths, T., Yamada, T., & Ueda, N. (2006). Learning systems of concepts with an infinite relational model. In *Proceedings of 21st AAAI*.

Kersting, K., Ahmadi, B., & Natarajan, S. (2009). Counting belief propagation. In *Proceedings of the 25th conference on uncertainty in artificial intelligence (UAI-09)*. Montreal, Canada.

Kersting, K., & De Raedt, L. (2007). Bayesian logic programming: Theory and tool. In L. Getoor & B. Taskar (Eds.), *An introduction to statistical relational learning* (pp. 291–321). Cambridge, MA, USA: MIT Press.

Kersting, K., De Raedt, L., & Raiko, T. (2006). Logial Hidden Markov Models. *Journal of Artificial Intelligence Research (JAIR), 25*, 425–456.

Kersting, K., & Xu, Z. (2009). Learning preferences with hidden common cause relations. In *Proceedings of the European conference on machine learning and principles and practice of knowledge discovery in databases (ECML PKDD 09). LNAI*, Bled, Slovenia, Springer.

Kisynski, J., & Poole, D. (2009). Lifted aggregation in directed first-order probabilistic models. In C. Boutilier (Ed.), *Proceedings of the international joint conference on artificial intelligence (IJCAI-09)*. Pasadena, CA, USA.

Kok, S., & Domingos, P. (2007). Statistical predicate invention. In *Proceedings of the twenty-fourth international conference on machine learning (ICML-07)*, Corvallis, OR, USA (pp. 433–440). ACM Press.

Limketkai, B., Liao, L., & Fox, D. (2005). Relational object maps for mobile robots. In F. Giunchiglia & L. P. Kaelbling (Eds.), *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI-05)*, Edinburgh, Scotland (pp. 1471–1476). AAAI Press.

McGovern, A., Friedland, L., Hay, M., Gallagher, B., Fast, A., Neville, J., et al. (2003). Exploiting relational structure to understand publication patterns in high-energy physics. *SIGKDD Explorations, 5*(2), 165–173.

McLachlan, G., & Krishnan, T. (1997). *The EM algorithm and extensions*. New York: Wiley.

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D., & Kolobov, A. (2005). BLOG: Probabilistic models with unknown objects. In F. Giunchiglia, L. P. Kaelbling (Eds.), *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI-05)*, Edinburgh, Scotland (pp. 1352–1359). Edinburgh, Scotland: AAAI Press.

Milch, B., Zettlemoyer, L., Kersting, K., Haimes, M., & Pack Kaelbling, L. (2008). Lifted probabilistic inference with counting formulas. In *Proceedings of the 23rd AAAI conference on artificial intelligence (AAAI-08)*.

Muggleton, S. (1996). Stochastic logic programs. In L. De Raedt (Ed.), *Advances in inductive logic programming* (pp. 254–264). Amsterdam: IOS Press.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19*(20), 629–679.

Neville, J., & Jensen, D. (2004). Dependency networks for relational data. In R. Rastogi, K. Morik, M. Bramer, & X. Wu (Eds.), *Proceedings of the fourth IEEE international conference on data mining (ICDM-04)*, Brighton, UK (pp. 170–177). IEEE Computer Society Press.

Neville, J., Simsek, Ö., Jensen, D., Komoroske, J., Palmer, K., & Goldberg, H. (2005). Using relational knowledge discovery to prevent securities fraud. In *Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery and data mining*. Chicago, IL, USA: ACM Press.

Ngo, L., & Haddawy, P. (1997). Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science, 171*, 147–177.

Passerini, A., Frasconi, P., & De Raedt, L. (2006). Kernels on prolog proof trees: Statistical learning in the ILP setting. *Journal of Machine Learning Research, 7*, 307–342.

Pfeffer, A. (2000). *Probabilistic reasoning for complex systems*. PhD thesis, Computer Science Department, Stanford University.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence Journal, 64*, 81–129.

Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence, 94*(1–2), 7–56.

Poole, D. (2003). First-order probabilistic inference. In G. Gottlob & T. Walsh (Eds.), *Proceedings of the eighteenth international joint conference on artificial intelligence (IJCAI-03)*, Acapulco, Mexico (pp. 985–991). San Francisco: Morgan Kaufmann.

Poon, H., & Domingos, P. (2008). Joint unsupervised coreference resolution with markov logic. In: *Proceedings of the 2008 conference on empirical methods in natural language processing (EMNLP)*, Honolulu, HI, USA.

Poon, H., & Domingos, P. (2009). Unsupervised semantic parsing. In *Proceedings of the 2009 conference on empirical methods in natural language processing (EMNLP)*, Singapore.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning, 62*, 107–136.

Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In L. Sterling (Ed.), *Proceedings of the twelfth international conference on logic programming (ICLP-95)*, Tokyo, Japan (pp. 715–729). MIT Press.

Segal, E., Battle, A., & Koller, D. (2003). Decomposing gene expression into cellular processes. In *Proceedings of Pacific symposium on biocomputing (PSB)* (pp. 89–100). World Scientific.

Segal, E., Taskar, B., Gasch, A., Friedman, N., & Koller, D. (2001). Rich probabilistic models for gene expression. *Bioinformatics, 17*(Suppl. 1), S243–252 (Proceedings of ISMB 2001).

Sen, P., Deshpande, A., & Getoor, L. (2008). Exploiting shared correlations in probabilistic databases. In *Proceedings of the international conference on very large data bases (VLDB-08)*. Auckland, New Zealand.

Silva, R., Chu, W., & Ghahramani, Z. (2007). Hidden common cause relations in relational learning. In *Advances in Neural information processing systems 20* (NIPS-2007). Cambridge, MA: MIT Press.

Singh, A., & Gordon, G. (2008). Relational learning via collective matrix factorization. In *Proceedings of 14th international conference on knowledge discovery and data mining*. Las Vegas, US.

Singla, P., & Domingos, P. (2008). Lifted first-order belief propagation. In *Proceedings of the 23rd AAAI conference on artificial intelligence (AAAI-08)*, Chicago, IL, USA (pp. 1094–1099).

Taskar, B., Abbeel, P., & Koller, D. (2002). Discriminative probabilistic models for relational data. In A. Darwiche & N. Friedman (Eds.), *Proceedings of the eighteenth conference on uncertainty in artificial intelligence (UAI-02)*, Edmonton, Alberta, Canada (pp. 485–492).

Wang, J., & Domingos, P. (2008). Hybrid markov logic networks. In *Proceedings of the 23rd AAAI conference on artificial intelligence (AAAI-08)*, Chicago, IL, USA (pp. 1106–1111).

Xu, Z., Kersting, K., & Tresp, V. (2009). Multi-relational learning with Gaussian processes. In C. Boutilier (Ed.) *Proceedings of the international joint conference on artificial intelligence (IJCAI-09)*. Pasadena, CA.

Xu, Z., Tresp, V., Rettinger, A., & Kersting, K. (2010). Social network mining with nonparametric relational models. In *Advances in social network mining and analysis*. Lecture Notes in Computer Science (Vol. 5498), Berlin/Heidelberg: Springer.

Xu, Z., Tresp, V., Yu, K., & Kriegel, H. P. (2006). Infinite hidden relational models. In *Proceedings of 22nd UAI*.

Yu, K., & Chu, W. (2007). Gaussian process models for link analysis and transfer learning. In *Advances in Neural information processing systems 20* (NIPS-2007). Cambridge, MA: MIT Press.

Yu, K., Chu, W., Yu, S., Tresp, V., & Xu, Z. (2006). Stochastic relational models for discriminative link prediction. In *Advances in Neural information processing systems 19* (NIPS-2006). Cambridge, MA: MIT Press.

Stochastic Finite Learning

Thomas Zeugmann
Hokkaido University,
Sapporo, Japan

Motivation and Background

Assume that we are given a concept class $\mathcal{C}$ and should design a learner for it. Next, suppose we already know or could prove $\mathcal{C}$ not to be learnable in the model of ▶PAC Learning. But it can be shown that $\mathcal{C}$ is learnable within Gold's (1967) model of ▶Inductive Inference or learning in the limit. Thus, we can design a learner behaving as follows. When fed any of the data sequences allowed in this model, it converges in the limit to a hypothesis correctly describing the target concept. Nothing more is known. Let M be any fixed learner. If $(d_n)_{n \geq 0}$ is any data sequence, then the *stage of convergence* is the least integer m such that $M(d_m) = M(d_n)$ for all $n \geq m$, provided such an n exists (and infinite, otherwise). In general, it is undecidable whether or not the learner has already reached the stage of convergence, but even if it is decidable for a particular concept class, it may be practically infeasible to do so. This *uncertainty* may not be tolerable in many applications.

When we tried to overcome this uncertainty, the idea of stochastic finite learning emerged. Clearly, in general nothing can be done, since in Gold's (1967) model the learner has to learn from any data sequence. So for every concept that needs more than one datum to converge, one can easily construct a sequence, where the first datum is repeated very often and where therefore, the learner does not find the right hypothesis within the given bound. However, such data sequences seem unnatural. Therefore, we looked at data sequences that are generated with respect to some probability distribution taken from a prespecified class of probability distributions and computed the expected *total learning time*, i.e., the expected time until the learner reaches the stage of convergence (cf. Erlebach, Rossmanith, Stadtherr, Steger, & Zeugmann, 2001; Zeugmann, 2006). Clearly, one is then also interested in knowing how often the expected total learning time is exceeded. In general, Markov's inequality can be applied to obtain the relevant tail bounds. However, if the learner is known to be rearrangement independent and conservative, then we always get *exponentially* shrinking tail bounds (cf. Rossmanith & Zeugmann, 2001). A learner is said to be *rearrangement independent*, if its output depends exclusively on the range and length of its input (but not the order) (cf., e.g., Lange & Zeugmann, 1996 and the references therein). Furthermore, a learner is *conservative*, if it exclusively performs mind changes that can be justified by an inconsistency of the abandoned hypothesis with the data received so far (see Angluin, 1980b for a formal definition).

Combining these ideas results in stochastic finite learning. A stochastic finite learner is successively fed data about the target concept. Note that these data are generated randomly with respect to one of the probability distributions from the class of underlying probability distributions. Additionally, the learner takes a confidence parameter δ as input. But in contrast to learning in the limit, the learner itself decides how many examples it wants to read. Then it computes a hypothesis, outputs it and stops. The hypothesis output is correct for the target with a probability at least $1 - \delta$.

The description given above explains how it works, but not why it does. Intuitively, the stochastic finite learner simulates the limit learner until an upper bound for twice the expected total number of examples needed until convergence has been met. Assuming this to be true, by Markov's inequality, the limit learner has now converged with a probability $1/2$. All what is left is to decrease the probability of failure. This can be done by using again Markov's inequality, i.e., increasing the sample complexity by a factor of $1/\delta$ results in a confidence of $1 - \delta$ for having reached the stage of convergence.

Note that the stochastic finite learner has to calculate an upper bound for the stage of convergence. This is precisely the point where we need the parameterization of the class $\mathcal{D}$ of underlying probability distributions. Then, a bit of *prior knowledge* must be provided in the form of suitable upper and/or lower bounds for the parameters involved. A more serious difficulty is to incorporate the unknown target concept into this estimate. This step depends on the concrete learning problem on hand and requires some extra effort.

It should also be noted that our approach may be beneficial even in case that the considered concept class is PAC learnable.

Definition

Let $\mathcal{D}$ be a set of probability distributions on the learning domain, let $\mathcal{C}$ be a concept class, $\mathcal{H}$ a hypothesis space for $\mathcal{C}$, and $\delta \in (0,1)$. $(\mathcal{C},\mathcal{D})$ is said to be *stochastically finitely learnable with δ-confidence* with respect to $\mathcal{H}$ iff there is a learner M that for every $c \in \mathcal{C}$ and every $D \in \mathcal{D}$ performs as follows. Given any random data sequence θ for c generated according to D, M stops after having seen a finite number of examples and outputs a single hypothesis $h \in \mathcal{H}$. With a probability at least $1-\delta$ (with respect to distribution D), h has to be correct, i.e., $c = h$.

If stochastic finite learning can be achieved with δ-confidence for every $\delta > 0$ then we say that $(\mathcal{C},\mathcal{D})$ can be learned stochastically finite *with high confidence*.

Detail

Note that there are subtle differences between our model and PAC learning. By its definition, stochastic finite learning is not completely distribution independent. A bit of *additional knowledge* concerning the underlying probability distributions is required. Thus, from that perspective, stochastic finite learning is weaker than the PAC model. On the other hand, we do *not* measure the quality of the hypothesis with respect to the underlying probability distribution. Instead, we require the hypothesis computed to be exactly correct with high probability. Note that the exact identification with high confidence has been considered within the PAC paradigm, too (cf., e.g., Saly, Goldman, & Schapire, 1993). Conversely, we also can easily relax the requirement to learn *probably exactly correct*, but whenever possible we shall not do it.

Furthermore, in the uniform PAC model as introduced in Valiant (1984), the sample complexity depends exclusively on the VC dimension of the target concept class and the error and confidence parameters ε and δ, respectively. This model has been generalized by allowing the sample size to depend on the concept complexity, too (cf., e.g., Blumer, Ehrenfeucht, Haussler, & Warmuth, 1989; Haussler, Kearns, Littlestone, & Warmuth, 1991). Provided no upper bound for the concept complexity of the target concept is given, such PAC learners decide themselves how many examples they wish to read (cf. Haussler et al., 1991). This feature is also adopted to our setting of stochastic finite learning. However, all variants of efficient ▶PAC Learning, we are aware of, require that all hypotheses from the relevant hypothesis space are uniformly polynomially evaluable. Though this requirement may be necessary in some cases to achieve (efficient) stochastic finite learning, it is not necessary in general as we shall see below.

In the following, we provide two sample applications of Stochastic Finite Learning. We always choose the concept class $\mathcal{C}$ itself as hypothesis space.

Learning Monomials

Let $X_n = \{0,1\}^n$ be the learning domain, let $\mathcal{L}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2 \ldots, x_n, \bar{x}_n\}$ (set of literals), and consider the class $\mathcal{C}_n$ of all concepts describable by a conjunction of literals. As usual, we refer to any conjunction of literals as a *monomial*. A monomial m describes a concept $c \subseteq X_n$ in the obvious way: The concept contains exactly those binary vectors for which the monomial evaluates to 1.

The basic ingredient to the stochastic finite learner is Haussler's (1987) Wholist algorithm, and thus the main emphasis is on the resulting complexity. The Wholist algorithm can also be used to achieve ▶PAC Learning of the class $\mathcal{C}_n$ and the resulting sample complexity is $O(1/\varepsilon \cdot (n + \ln(1/\delta)))$ for all $\varepsilon, \delta \in (0,1]$. Since the Wholist algorithm learns from positive examples only, it is meaningful to study the learnability of $\mathcal{C}_n$ from positive examples only. So, the stage of convergence is *not* decidable.

Since the Wholist algorithm immediately converges for the empty concept, we exclude it from our considerations. That is, we consider concepts $c \in \mathcal{C}_n$ described by a monomial $m = \bigwedge_{j=1}^{\#(m)} \ell_{i_j}$ such that $k = k(m) = n - \#(m) > 0$. A literal not contained in m is said to be irrelevant. Bit i is said to be irrelevant for monomial m if neither x_i nor $\bar{x}_i$ appear in m. There are 2^k positive examples for c. For the sake of presentation, we assume these examples to be *binomially distributed* with parameter p. So, in a random positive example, all entries corresponding to irrelevant bits are selected independently of each other. With some probability p this will be a 1, and with probability $1-p$, this will be a 0. Only distributions where $0 < p < 1$ are considered, since otherwise exact identification is impossible. Now, one can show that the expected number of examples needed by the Wholist algorithm until convergence is bounded

by $\lceil \log_\psi k(m) \rceil + \tau + 2$, where $\psi := \min\{\frac{1}{1-p}, \frac{1}{p}\}$ and $\tau := \max\{\frac{p}{1-p}, \frac{1-p}{p}\}$.

Let *CON* denote a random variable for the stage of convergence. Since the Wholist algorithm is rearrangement independent and conservative, we can conclude (cf. Rossmanith & Zeugmann, 2001)

$$\Pr(CON > 2t \cdot \mathrm{E}[CON]) \leq 2^{-t} \text{for all natural numbers } t \geq 1.$$

Finally, in order to obtain a stochastic finite learner, we reasonably assume that the *prior knowledge* is provided by parameters p_{low} and p_{up} such that $p_{\text{low}} \leq p \leq p_{\text{up}}$ for the true parameter p. Binomial distributions fulfilling this requirement are called $(p_{\text{low}}, p_{\text{up}})$-*admissible distributions*. Let $\mathcal{D}_n[p_{\text{low}}, p_{\text{up}}]$ denote the set of such distributions on X_n. Then one can show

Let $0 < p_{\text{low}} \leq p_{\text{up}} < 1$ *and* $\psi := \min\left\{\frac{1}{1-p_{\text{low}}}, \frac{1}{p_{\text{up}}}\right\}$. *Then* $(\mathcal{C}_n, \mathcal{D}_n[p_{\text{low}}, p_{\text{up}}])$ *is stochastically finitely learnable with high confidence from positive examples. To achieve* δ*-confidence no more than* $O\left(\log_2 1/\delta \cdot \log_\psi n\right)$, *many examples are necessary.*

Therefore, we have achieved an exponential improvement on the number of examples needed for learning (compared to the PAC bound displayed above), and, in addition, our stochastic finite learner exactly identifies the target. Note that this result is due to Reischuk and Zeugmann, however, we refer the reader to Zeugmann (2006) for the relevant proofs.

The results obtained for learnability from positive examples only can be extended mutatis mutandis to the case when the learner is fed positive and negative examples (cf. Zeugmann, 2006 for details).

Learning Pattern Languages

The pattern languages have been introduced by Angluin (1980a) and can be informally defined as follows. Let $\Sigma = \{0, 1, \dots\}$ be any finite alphabet containing at least two elements. Let $X = \{x_0, x_1, \dots\}$ be a countably infinite set of variables such that $\Sigma \cap X = \varnothing$. *Patterns* are nonempty strings over $\Sigma \cup X$, e.g., 01, $0x_0111$, $1x_0x_00x_1x_2x_0$ are patterns. The length of a string $s \in \Sigma^*$ and of a pattern π is denoted by $|s|$ and $|\pi|$, respectively. A pattern π is in *canonical form* provided that if k is the number of different variables in π then the variables occurring in π are precisely $x_0, \dots, x_{k-1}$. Moreover, for every j with $0 \leq j < k-1$, the leftmost occurrence of x_j in π is left to the leftmost occurrence of x_{j+1}. The examples given above are patterns in canonical form.

If k is the number of different variables in π then we refer to π as to a *k-variable pattern*. For example, $x0xx$ is a one-variable pattern, and $x_010x_1x_0$ is a two-variable pattern. If π is a pattern, then the language generated by π is the set of all strings that can be obtained from π by substituting a nonnull element $s_i \in \Sigma^*$ for each occurrence of the variable symbol x_i in π, for all $i \geq 0$. We use $L(\pi)$ to denote the language generated by pattern π. So, 1011, 1001010 belong to $L(x0xx)$ (by substituting 1 and 10 for x, respectively) and 010110 is an element of $L(x_010x_1x_0)$ (by substituting 0 for x_0 and 11 for x_1). Note that even the class of all one-variable patterns has infinite ▶VC Dimension (cf. Mitchell, Scheffer, Sharma, & Stephan, 1999).

Reischuk and Zeugmann (2000) designed a stochastic finite learner for the class of all one-variable pattern languages that runs in time $O(|\pi| \log(1/\delta))$ for all meaningful distributions and learns from positive data only. That is, all data fed to the learner belong to the target pattern language. Furthermore, by meaningful distribution essentially the following is meant. The expected length of an example should be finite, and the distribution should allow to learn the target pattern. This is then expressed by fixing some suitable parameters. It should be noted that the algorithm is highly practical, and a modification of it also works for the case that *empty* substitutions are allowed.

For the class of all pattern languages, one can also provide a stochastic finite learner, identifying the whole class from positive data. In order to arrive at a suitable class of distributions, essentially three requirements are made. The first one is the same as in the one-variable case, i.e., the expected length $\mathrm{E}[\Lambda]$ of a generated string should be finite. Second, the class of distributions is restricted to regular product distributions, i.e., for all variables the substitutions are identically distributed. Third, two parameters α and β are introduced. The parameter α is the probability that a string of length 1 is substituted and β is the conditional probability that two random strings that get substituted into π are identical under the condition that both have length 1. These two parameters ensure that the target pattern language is learnable at all. The stochastic finite learner is then using as *a* priori

knowledge a lower bound α^* for α and an upper bound β^* for β. The basic ingredient to this stochastic finite learner is Lange and Wiehagen's (1991) pattern language learning algorithm. Rossmanith and Zeugmann's (2001) stochastic finite learner for the pattern languages runs in time $O\left((1/\alpha_*^k)\mathrm{E}[\Lambda]\log_{1/\beta_*}(k)\log_2(1/\delta)\right)$, where k is the number of different variables in the target pattern. So, with increasing k it becomes impractical.

Note that the two stochastic finite learners for the pattern languages can compute the expected stage of convergence, since the first string seen provides an upper bound for the length of the target pattern.

For further information, we refer the reader to Zeugmann (2006) and the references therein. More research is needed to explore the potential of stochastic finite learning. Such investigations should extend the learnable classes, study the incorporation of noise, and explore further possible classes of meaningful probability distributions.

Cross References

►Inductive Inference
►PAC Learning

Recommended Reading

Angluin, D. (1980a). Finding patterns common to a set of strings. *Journal of Computer and System Sciences, 21*(1), 46–62.

Angluin, D. (1980b). Inductive inference of formal languages from positive data. *Information Control, 45*(2), 117–135.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik–Chervonenkis dimension. *Journal of the ACM, 36*(4), 929–965.

Erlebach, T., Rossmanith, P., Stadtherr, H., Steger, A., & Zeugmann, T. (2001). Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science, 261*(1), 119–156.

Gold, E. M. (1967). Language identification in the limit. *Information and Control, 10*(5), 447–474.

Haussler, D. (1987). Bias, version spaces and Valiant's learning framework. In P. Langley (Ed.), *Proceedings of the fourth international workshop on machine learning* (pp. 324–336). San Mateo, CA: Morgan Kaufmann.

Haussler, D., Kearns, M., Littlestone, N., & Warmuth, M. K. (1991). Equivalence of models for polynomial learnability. *Information and Computation, 95*(2), 129–161.

Lange, S., & Wiehagen, R. (1991). Polynomial-time inference of arbitrary pattern languages. *New Generation Computing, 8*(4), 361–370.

Lange, S., & Zeugmann, T. (1996). Set-driven and rearrangement-independent learning of recursive languages. *Mathematical Systems Theory, 29*(6), 599–634.

Mitchell, A., Scheffer, T., Sharma, A., & Stephan, F. (1999). The VC-dimension of sub- classes of pattern languages. In O. Watanabe & T. Yokomori (Eds.), *Algorithmic learning theory, tenth international conference, ALT"99, Tokyo, Japan, December 1999, Proceedings, Lecture notes in artificial intelligence* (Vol. 1720, pp. 93–105). Springer.

Reischuk, R., & Zeugmann, T. (2000). An average-case optimal one-variable pattern language learner. *Journal of Computer and System Sciences, 60*(2), 302–335.

Rossmanith, P., & Zeugmann, T. (2001). Stochastic finite learning of the pattern languages. *Machine Learning, 44*(1/2), 67–91.

Saly, A., Goldman, M. J. K., & Schapire, R. E. (1993). Exact identification of circuits using fixed points of amplification functions. *SIAM Journal of Computing, 22*(4), 705–726.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM, 27*(11), 1134–1142.

Zeugmann, T. (1998). Lange and Wiehagen's pattern language learning algorithm: An average-case analysis with respect to its total learning time. *Annals of Mathematics and Artificial Intelligence, 23*, 117–145.

Zeugmann, T. (2006). From learning in the limit to stochastic finite learning. *Theoretical Computer Science, 364*(1), 77–97. Special issue for ALT 2003.

Stratified Cross Validation

Stratified Cross Validation is a form of ►cross validation in which the class distribution is kept as close as possible to being the same across all folds.

Stream Mining

A subfield of knowledge discovery called *stream mining* addresses the issue of rapidly changing data. The idea is to be able to deal with the stream of incoming data quickly enough to be able to simultaneously update the corresponding models (e.g., ontologies), as the amount of data is too large to be stored: new evidence from the incoming data is incorporated into the model without storing the data. For instance, modeling ontology changes and evolution over time using text mining methods (►Text Mining for Semantic Web). The underlying methods are based on the machine learning methods of ►Online Learning, where the model is built from the initially available data and updated regularly as more data become available.

Examples of data streams include computer network traffic, phone conversations, ATM transactions, web searches, and sensor data.

Cross References

►Clustering Data Streams
►Online Learning

String kernel

A *string kernel* is a function from any of various families of kernel functions (see ►kernel methods) that operate over strings and sequences. The most typical example is as follows. Suppose that we are dealing with strings over a finite alphabet Σ. Given a string $a = a_1 a_2 \ldots a_n \in \Sigma^\star$, we say that a substring $p = p_1 p_2 \ldots p_k$ *occurs* in a on positions $i_1\ i_2 \ldots i_k$ iff $1 \le i_1 < i_2 < \ldots < i_k \le n$ and $a_{ij} = p_j$ for all $j = 1, \ldots, k$. We define the *weight* of this occurrence as $\lambda^{i_k - i_1 - k + 1}$, where $\lambda \in [0, 1]$ is a constant chosen in advance; in other words, an occurrence weighs less if characters of p are separated by other characters. Let $\phi_p(a)$ be the sum of the weights of all occurrences of p in a, and let $\phi(a) = (\phi_p(a))_{p \in \Sigma^*}$ be an infinite-dimensional feature vector consisting of $\phi_p(a)$ for all possible substrings $p \in \Sigma^\star$. It turns out that the dot product of two such infinite-length vectors, $K(a, a') = \phi(a)^T \phi(a')$, can be computed in time polynomial in the length of a and a', e.g., using dynamic programming. The function K defined in this way can be used as a kernel with various kernel methods. See also ►feature construction in text mining.

String Matching Algorithm

A string matching algorithm returns parts of text matching a given pattern, such as a *regular expression*. Such algorithms have countless applications, from file editing to bioinformatics. Many algorithms compute deterministic finite automata, which can be expensive to build, but are usually efficient to use; they include the *Knuth–Morris–Pratt* algorithm and the *Boyer–Moore* algorithm, that build the automaton in time $O(m)$ and $O(m + s)$, respectively, where m is the length of the pattern and s the size of the alphabet, and match a text of length n in time $O(n)$ in the worst case.

Structural Credit Assignment

►Credit Assignment

Structural Risk Minimization

Xinhua Zhang
Australian National University, NICTA London Circuit,
Canberra, Australia

Definition

The goal of learning is usually to find a model which delivers good generalization performance over an underlying distribution of the data. Consider an input space $\mathcal{X}$ and output space $\mathcal{Y}$. Assume the pairs $(X \times Y) \in \mathcal{X} \times \mathcal{Y}$ are random variables whose (unknown) joint distribution is P_{XY}. It is our goal to find a predictor $f : \mathcal{X} \mapsto \mathcal{Y}$ which minimizes the expected risk:

$$P(f(X) \neq Y) = \mathbb{E}_{(X,Y) \sim P_{XY}} [\delta(f(X) \neq Y)],$$

where $\delta(z) = 1$ if z is true, and 0 otherwise.

In practice we only have n pairs of training examples (X_i, Y_i) drawn identically and independently from P_{XY}. Based on these samples, the ►empirical risk can be defined as

$$\frac{1}{n} \sum_{i=1}^{n} \delta(f(X_i) \neq Y_i).$$

Choosing a function f by minimizing the empirical risk often leads to ►overfitting. To alleviate this problem, the idea of structural risk minimization (SRM) is to employ an infinite sequence of models $\mathcal{F}_1, \mathcal{F}_2, \ldots$ with increasing capacity. Here each $\mathcal{F}_i$ is a set of functions, e.g., polynomials with degree 3. We minimize the empirical risk in each model with a penalty for the capacity of the model:

$$f_n := \operatorname*{arg\,min}_{f \in \mathcal{F}_i, i \in \mathbb{N}} \frac{1}{n} \sum_{j=1}^{n} \delta(f(X_j) \neq Y_j) + \text{capacity}(\mathcal{F}_i, n),$$

where $\text{capacity}(\mathcal{F}_i, n)$ quantifies the complexity of model $\mathcal{F}_i$ in the context of the given training set. For example, it equals 2 when $\mathcal{F}_i$ is the set of polynomials with degree 2. In other words, when trying to reduce

the risk on the training set, we prefer a predictor from a simple model.

Note the penalty is measured on the model $\mathcal{F}_i$, *not* the predictor f. This is different from the regularization framework, e.g., support vector machines, which penalizes the complexity of the *classifier*.

More details about SRM can be found in Vapnik (1998).

Recommended Reading

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.

Structure

►Topology of a Neural Network

Structured Data Clustering

►Graph Clustering

Structured Induction

Michael Bain
University of New South Wales,
Sydney, Australia

Definition

Structured induction is a method of applying machine learning in which a model for a task is learned using a representation where some of the components are themselves the outputs of learned models for specified sub-tasks. The idea was inspired by structured programming (Dahl, Dijkstra and Hoare, 1972), in which a complex task is solved by repeated decomposition into simpler sub-tasks that can be easily analyzed and implemented. The approach was first developed by Alen Shapiro (1987) in the context of constructing expert systems by ►decision tree learning, but in principle it could be applied using other learning methods.

Motivation and Background

Structured induction is designed to solve complex learning tasks for which it is difficult a priori to obtain a set of attributes or features in which it is possible to represent an accurate approximation of the target hypothesis reasonably concisely. In Shapiro's approach, a hierarchy of ►decision trees is learned, where in each tree of the hierarchy the attributes can have values that are outputs computed by a lower-level ►decision tree. Shapiro showed in computer chess applications that structured induction could learn accurate models, while significantly reducing their complexity. Structured induction was first commercialized in the 1980s by a number of companies providing expert systems solutions and has since seen many applications (Razzak, Hassan and Pettipher, 1984).

A key assumption is that human expertise is available to define the task structure. Several approaches have been proposed to address the problem of learning this structure (under headings such as ►constructive induction, ►representation change, ►feature construction, and ►predicate invention) although to date, none have received wide acceptance.

The identification of knowledge acquisition as the "bottleneck" in knowledge engineering by Feigenbaum (1977) sparked considerable interest in symbolic machine-learning methods as a potential solution. Early work on ►decision tree induction around this time was often driven by problems from computer chess, a challenging domain by the standards of the time due to relatively large data sets and the complexity of the target hypotheses. In a landmark paper on his ID3 ►decision tree learning algorithm, Quinlan (1983) reported experiments on learning to classify positions in a four-piece chess endgame as winnable (or not) within a certain number of moves ("lost N-ply"). A set of attributes was defined as *inadequate* for a classification task if two objects belonging to different classes had identical values for each attribute. He concluded that "almost all the effort (for a non chess-player, at least) must be devoted to finding attributes that are adequate for the classification problem being tackled".

The problem is that the effort of developing the set of attributes becomes disproportionate to the time taken to do the induction. Quinlan (1983) reported durations of three weeks and two man-months, respectively, to define an adequate set of attributes for the "lost

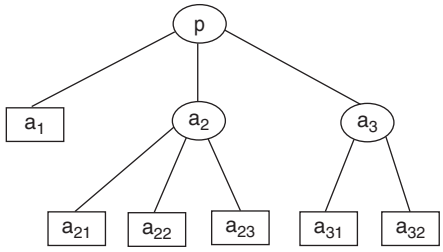
2-ply" and "lost 3-ply" experiments. In contrast, the implementation of ID3 used in that work induced the ►decision trees in 3 s and 34 s, respectively. It is worth noting that the more complex problem of "lost 4-ply" was abandoned due to the difficulty of developing an adequate set of attributes.

Although Quinlan's experiments with ID3 produced exact ►classifiers for his chess problems, the resulting ►decision trees were too large to be comprehensible to domain experts. This is a serious drawback when machine learning is used with the goal of installing learned ►rules in an expert system, since the system cannot provide understandable explanations. Shapiro and Niblett (1982) proposed structured induction as a solution to this problem, and the method was developed in Shapiro's PhD thesis (Shapiro, 1987) motivated by expert systems development.

Structure of Learning System

Structured induction is essentially a two-stage process, comprising a top-down decomposition of the problem into a solution structure, followed by a bottom-up series of ►classifier learning steps, one for each of the subproblems. A knowledge engineer and a domain expert are required to collaborate at each stage, with the latter acting as a source of examples. The use of machine learning to avoid the knowledge acquisition bottleneck is based on the finding that although domain experts find it difficult to express general and accurate ►rules for a problem, they are usually able to generate tutorial examples in an attribute-value formalism from which ►rules can be generalized automatically. The key insight of structured induction is that the task of specifying an attribute and its value set can be treated as a subproblem of the learning task, and solved in the same way.

The approach can be illustrated by a simple example using the structure shown in Fig. 1. Suppose the task is to learn a model for some concept **p**. Suppose further that the domain expert proposes three attributes $\mathbf{a_1}, \mathbf{a_2}$, and $\mathbf{a_3}$ as adequate for the classification of **p**. Now the domain expert consults with the knowledge engineer and it is decided that while attribute $\mathbf{a_1}$ is directly implementable, the other two are not. An attribute that is directly implementable by a knowledge engineer is referred to as *primitive* for the domain.



Structured Induction. Figure 1. A schematic diagram of a model learned by structured induction (after Shapiro, 1987). Concepts to be learned are shown in ovals, and primitive attributes in boxes. The top-level concept p is defined in terms of the primitive attribute $\mathbf{a_1}$ and two sub-concepts $\mathbf{a_2}$ and $\mathbf{a_3}$. Each of the two sub-concepts are further decomposed into sets of primitive attributes, $\mathbf{a_{21...3}}$ and $\mathbf{a_{31...2}}$

The other attributes become sub-concepts $\mathbf{a_2}$ and $\mathbf{a_3}$, and each in turn is addressed by the domain expert. In this case, three attributes are proposed as most relevant to the solution of $\mathbf{a_2}$, and two for $\mathbf{a_3}$. Since all of these attributes are found to be primitive, the top-down problem decomposition stage is therefore complete.

The domain expert, having proposed a set of primitive attributes for a sub-concept, say $\mathbf{a_3}$, is now required to provide a set of classified examples defined in terms of values for attributes $\mathbf{a_{31}}$ and $\mathbf{a_{32}}$. Given these examples, the knowledge engineer will run a learning algorithm to induce a ►classifier such as a ►decision tree. The domain expert will then inspect the ►classifier and can optionally refine it by supplying further examples, until they are satisfied that it completely and correctly defines the sub-concept $\mathbf{a_3}$. This process is repeated in a bottom-up fashion for all sub-concepts. At every level of the hierarchy, once all sub-concepts have been defined, they are now directly executable ►classifiers and can be treated in the same way as primitive attributes and used for learning. The structured induction solution is complete once an acceptable ►classifier has been learned for the top-level concept, **p** in this example.

Structured Versus Unstructured Induction

On two chess end-game domains, Shapiro (1987) showed that structured induction could generate more compact trees from fewer examples compared with an unstructured approach. To quantify this improvement,

Shapiro made an analysis based on Michie's finite message information theory (Michie, 1982). This showed that on one of the domains, the information gain contributed by the structured induction approach over learning unstructured trees from the same set of examples was 84%. Essentially, this is because the structure devised by the domain expert in collaboration with the knowledge engineer provides a context for each of the induction tasks required. Since within this context only a subset of the complete attribute set is used to specify a sub-concept, it suffices to obtain only sufficient examples to learn a model for that sub-concept. However, without the benefit of such contextual restrictions the task of learning a complete solution can require considerably more examples. Shapiro's analysis is an attempt to quantify the relative increase in information per example in structured versus unstructured induction.

Related Work

While induction can bypass the knowledge acquisition bottleneck, in structured induction the process of acquiring the structure in collaboration with a domain expert can become a new bottleneck. In an attempt to avoid this, a number of researchers have attempted to develop methods whereby the structure, as well as the sub-concept models can be learned automatically.

Muggleton (1987) introduced ►inverse resolution as an approach to learning structured ►rule sets in a system called Duce. As part of this process, a domain expert is required to provide names for new sub-concepts or *predicates* that are proposed by the learning algorithm. A domain expert is also required to confirm learned ►rules. Both these roles are similar to those required of the expert in constructive induction, but the key difference is that the learning algorithm is now the source of both the structure and the ►rules. Duce was applied to one of the chess end-game domains used in Shapiro's study (Shapiro, 1987) and found a solution that was less compact, but still accepted as comprehensible by a chess expert.

The Duce system searches for commonly occurring subsets of attribute-value pairs within ►rules, and uses these to construct new sub-concept definitions. Many approaches have been developed using related methods to learn new sub-concepts in the context of ►decision tree or ►rule learning; some examples include Pagallo and Haussler (1990), Zheng (1995), and Zhang and Honavar (2003). Gaines (1996) proposed EDAGs (exception directed acyclic graphs) as a generalization of both ►decision trees and ►rules with exceptions, and reported EDAG representations of chess end-game ►classifiers that were more comprehensible than either ►rules or ►decision trees. Zupan, Bohanec, Demsar, and Bratko (1999) developed a system named HINT designed to learn a model represented as a concept hierarchy based on methods of function decomposition. Inverse resolution as used in Duce has been generalized to first-order logic representations in the field of inductive logic programming. In this framework, the construction of new intermediate concepts is referred to as ►*predicate invention*, but to date this remains a largely open problem. More recently, much of the interest in ►representation change has focused on approaches like support vector machines, where the so-called kernel trick enables the use of *implicit* ►feature construction (Shawe-Taylor and Cristianini, 2004).

Cross References

►Classifier
►Constructive Induction
►Decision Tree
►Feature Construction
►Predicate Invention
►Rule Learning

Recommended Reading

Dahl, O. J., Dijkstra, E. W., & Hoare, C. A. R. (Eds.). (1972). *Structured programming*. London: Academic Press.

Feigenbaum, E. A. (1977). The art of artificial intelligence: Themes and case studies of knowledge engineering. In R. Reddy (Ed.), *Proceedings of the fifth international conference on artificial intelligence* (IJCAI77) (pp. 1014–1029). Los Altos, CA: William Kaufmann.

Gaines, B. (1996). Transforming rules and trees into comprehensible knowledge structures. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 205–226). Cambridge, MA: MIT Press.

Michie, D. (1982). Measuring the knowledge-content of expert programs. *Bulletin of the Institute of Mathematics and its Applications, 18*(11/12), 216–220.

Muggleton, S. (1987). Duce, an oracle-based approach to constructive induction. In *IJCAI 87* (pp. 287–292). Los Altos, CA: Kaufmann.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine learning*, 5, 71–99.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In R. Michalski, J. Carbonnel, & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*, (pp. 464–482). Palo Alto, CA: Tioga.

Razzak, M. A., Hassan, T., & Pettipher, R. (1984). Extran-7: A Fortran-based software package for building expert systems. In M. A. Bramer (Ed.), *Research and development in expert systems* (pp. 23–30). Cambridge: Cambridge University Press.

Shapiro, A., & Niblett, T. (1982). Automatic induction of classification rules for a chess endgame. In M. R. B. Clarke (Ed.), *Advances in computer chess* (Vol. 3, pp. 73–91). Pergamon: Oxford.

Shapiro, A. D. (1987). *Structured Induction in expert systems*. Wokingham: Turing Institute Press with Addison Wesley.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.

Zhang, J., & Honavar, V. (2003). Learning decision tree classifiers from attribute value taxonomies and partially specified data. In *ICML-2003: Proceedings of the twentieth international conference on machine learning*, Menlo Park, CA: AAAI Press.

Zheng, Z. (1995). Constructing nominal X-of-N attributes. In *Proceedings of the fourteenth International joint conference on artificial intelligence (IJCAI, 95)* (pp. 1064–1070). Los Altos, CA: Morgan Kaufmann.

Zupan, B., Bohanec, M., Demsar, J., & Bratko, I. (1999). Learning by discovering concept hierarchies. *Artificial Intelligence, 109*, 211–242.

Subgroup Discovery

Definition

Subgroup discovery (Klösgen, 1996; Lavrač, Kavšek, Flach, & Todorovski, 2004) is an area of ►supervised descriptive rule induction. The subgroup discovery task is defined as given a population of individuals and a property of those individuals that we are interested in, find population subgroups that are statistically "most interesting," for example, are as large as possible and have the most unusual statistical (distributional) characteristics with respect to the property of interest.

Recommended Reading

Klösgen, W., (1996). Explora: A multipattern and multistrategy discovery assistant. In *Advances in knowledge discovery and data mining* (pp. 249–271). Cambridge: MIT Press.

Lavrač, N., Kavšek, B., Flach, P. A., & Todorovski, L. (2004). Subgroup discovery with CN2-SD. *Journal of Machine Learning Research, 5*, 153–188.

Sublinear Clustering

Artur Czumaj[1], Christian Sohler[2]
[1]University of Warwick, Coventry, UK,
[2]University of Paderborn,
Paderborn, Germany

Definition

Sublinear clustering describes the process of clustering a given set of input objects using only a small subset of the input set, which is typically selected by a random process. A solution computed by a sublinear clustering algorithm is an implicit description of the clustering (rather than a partition of the input objects), for example in the form of cluster centers. Sublinear clustering is usually applied when the input set is too large to be processed with standard clustering algorithms.

Motivation and Background

►Clustering is the process of partitioning a set of objects into subsets of similar objects. In machine learning, it is, for example, used in unsupervised learning to fit input data to a density model. In many modern applications of clustering, the input sets consist of billions of objects to be clustered. Typical examples include web search, analysis of web traffic, and spam detection. Therefore, even though many relatively efficient clustering algorithms are known, they are often too slow to cluster such huge inputs.

Since in some applications it is even not possible to cluster the entire input set, a new approach is needed to cope with very large data sets. The approach used in many different areas of science and engineering in this context is to *sample* a subset of the data and to analyze this sample instead of the whole data set. This is also the underlying method used in sublinear clustering. The main challenge and innovation in this area lies in the qualitative analysis of random sampling (in the form of approximation guarantees) and the design of *non uniform sampling* strategies that approximate the input set provably better than *uniform random sampling*.

Structure of the Learning System

In sublinear clustering a large input set of objects is to be partitioned into subsets of similar objects. Usually, this

input is a point set P either in the Euclidean space or in the metric space. The clustering problem is specified by an objective function that determines the quality or cost of every possible clustering. The goal is to find a clustering of minimum cost/maximum quality. For example, given a set P of points in the Euclidean space the objective of ▶*k*-means clustering is to find a set C of k centers that minimizes $\sum_{p\in P}(d(p,C))^2$, where $d(p,C)$ denotes the distance of p to the nearest center from C. Since usually the clustering problems are computationally hard ($\mathcal{NP}$-hard), one typically considers *approximate* solutions: instead of finding a clustering that optimizes the cost of the solution, one aims at a solution whose cost is close to the optimal one.

In sublinear algorithms a solution is computed for a small representative subset of objects, for example a random sample. The solution is represented implicitly, for example, in the form of cluster centers and it can be easily extended to the whole input point set. The quality of the output is analyzed *with respect to the original point set*. The challenge is to *design and analyze fast (sublinear-time) algorithms* that select a subset of objects that very well represent the entire input, such that the solution computed for this subset will also be a good solution for the original point set. This can be achieved by uniform and non uniform *random sampling* and the computation of *core-sets*, i.e., small weighted subsets of the input that approximate the input with respect to a clustering objective function.

Theory/Solution

Clustering Problems

For any point p and any set Q in a metric space (X,d), let $d(p,Q) = \min_{q\in Q} d(p,q)$. A point set P is *weighted* if there is a function w assigning a positive weight to each point in P.

Radius *k*-Clustering: Given a weighted set P of points in a metric space (X,d), find a set $C \subseteq P$ of k centers minimizing $\max_{p\in P} d(p,C)$.

Diameter *k*-Clustering: Given a weighted set P of points in a metric space (X,d), find a partition of P into k subsets $P_1,\ldots,P_k$, such that $\max_{i=1}^{k} \max_{p,q\in P_i} d(p,q)$ is minimized.

***k*-Median:** Given a weighted set P of points in a metric space (X,d), find a set $C \subseteq P$ of k centers that minimizes $median(P,C) = \sum_{p\in P} w(p)\cdot d(p,C)$.

***k*-Means:** Given a weighted set of points P in a metric space (X,d), find a set $C \subseteq P$ of k centers that minimizes $mean(P,C) = \sum_{p\in P} w(p)\cdot (d(p,C))^2$.

Random Sampling and Sublinear-Time Algorithms

Random sampling is perhaps the most natural approach to design sublinear-time algorithms for clustering. For the input set P, random sampling algorithm follows the following scheme:

1. Pick a random sample S of points
2. Run an algorithm (possibly an approximation one) for (given kind of) clustering for S
3. Return the clustering induced by the solution for S

The running time and the quality of this algorithm depends on the size of the random sample S and of the running time and the quality of the algorithm for clustering of S. Because almost all interesting clustering problems are computationally intractable ($\mathcal{NP}$-hard), usually the second step of the sampling scheme uses an approximation algorithm. (An algorithm for a minimization problem is called a λ-approximation if it always returns a solution whose cost is at most λ times the optimum.)

While random sampling approach gives very simple algorithms, depending on the clustering problem at hand, it often finds a clustering of poor quality and it is usually difficult to analyze. Indeed, it is easy to see that random sampling has some serious limitations to obtain clustering of good quality. Even the standard assumption that the input is in metric space is not sufficient to obtain good quality of clustering because of the small clusters which are "hidden" for random sampling approach. (If there is a cluster of size $o(|P|/|S|)$ then with high probability the random sample set S will contain no point from that cluster.) Therefore, another important parameters used in the analysis is the *diameter* of the metric space Δ, which is $\Delta = \max_{p,q\in P} d(p,q)$.

Quality of Uniformly Random Sampling: The quality of random sampling for three basic clustering problems

(k-means, k-median, and min-sum k-clustering) have been analyzed in Ben-David (2004), Czumaj and Sohler (2007), and Mishra, Oblinger, and Pitt (2001). In these papers, generic methods of analyzing uniform random sampling have been obtained. These results assume that the input point sets are in a metric space and are unweighted (i.e., when the weight function w is always 1).

Theorem 1 *Let $\epsilon > 0$ be an approximation parameter. Suppose that an α-approximation algorithm for the k-median problem in a metric space is used in step* (2) *of the uniform sampling, where $\alpha \geq 1$* (Ben-David 2004; Czumaj & Sohler 2007, Mishra et al., 2001). *If we choose S to be of size at least*

$$c\alpha\left(k + \frac{\Delta}{\epsilon}\left(\alpha + k\ln(k\Delta\alpha/\epsilon)\right)\right)$$

for an appropriate constant c, then the uniform sampling algorithm returns a clustering C^* (of S) such that with probability at least 0.99 *the normalized cost of clustering for S satisfies*

$$\frac{median(S, C^*)}{|S|} \leq \frac{2(\alpha + 0.01)OPT(P)}{|P|} + \epsilon,$$

where $OPT(S) = \min_C median(P, C)$ is the minimum cost of a solution for k-median for P.

Similar results can be shown for the k-means problem, and also for min-sum k-clustering (cf. Czumaj & Sohler, 2007). For example, for k-means, with a sample S of size at least $c\alpha\left(k + (\Delta^2/\epsilon)\left(\alpha + k\ln(k\Delta^2\alpha/\epsilon)\right)\right)$, with probability at least 0.99 the normalized cost of k-means clustering for S satisfy

$$\frac{mean(S, C^*)}{|S|^2} \leq \frac{4(\alpha + 0.01)OPT(P)}{|P|^2} + \epsilon,$$

where $OPT(S) = \min_C mean(P, C)$ is the minimum cost of a solution for k-means for P.

Improvements of these bounds for the case when the input consists of points in Euclidean space are also discussed in Mishra et al. (2001), Czumaj and Sohler (2007) discuss also . For example, for k-median, if one takes S of size at least $c\alpha\,(k + \Delta k\ln(\Delta/\epsilon)/\epsilon)$, then with probability at least 0.99 the normalized cost of k-median clustering for S satisfies

$$\frac{median(S, C^*)}{|S|} \leq \frac{(\alpha + 0.01)OPT(P)}{|P|} + \epsilon\ ,$$

and hence the approximation ratio is better than that in Theorem 1 by factor of 2.

The results stated in Czumaj and Sohler (2007) allow to parameterize the constants 0.99 and 0.01 in the claims above.

Property Testing of the Quality of Clustering: Since most of the clustering problems are computationally quite expensive, in some situations it might be interesting not to find a clustering (or its succinct representation), but just to test if the input set has a good clustering.

Alon, Dar, Parnas, and Ron (2003) introduced the notion of approximate testing of good clustering. A point set P is *c-clusterable* if it has a clustering of the cost at most c, that is, $OPT(P) \leq c$. To formalize the notion of having no good clustering, one says a point set is *ϵ-far from $(1 + \beta)c$-clusterable*, if more than an ϵ-fraction of the points in P must be removed (or moved) to make the input set $(1 + \beta)c$-clusterable. With these definitions, the goal is to design fast algorithms that accept the input point sets P, which are c-clusterable, and reject with probability at least 2/3 inputs are ϵ-far from $(1 + \beta)c$-clusterable. If neither holds, then the algorithms may either accept or reject. The bounds for the testing algorithms are phrased in terms of *sample complexity*, that is, the number of sampled input points which are considered by the algorithm (e.g., by using random sampling).

Alon et al. (2003) consider two classical clustering problems in this setting: radius and diameter k-clusterings. If the inputs are drawn from an arbitrary metric space, then they show that to distinguish between input points sets that are c-clusterable and are ϵ-far from $(1 + \beta)c$-clusterable with $\beta < 1$, the sample complexity must be at least $\Omega(\sqrt{|P|/\epsilon})$. However, to distinguish between inputs that are c-clusterable and are ϵ-far from $2c$-clusterable, the sample complexity is only $O(\sqrt{k/\epsilon})$.

A more interesting situation is for the input points drawn from the Euclidean d-dimensional space. In that case, even a constant-time algorithms are possible.

Theorem 2 *For the radius k-clustering, one can distinguish between points sets in R^d that are c-clusterable from those ϵ-far from c-clusterable with the sample complexity $\widetilde{O}(dk/\epsilon)$* (Alon et al., 2003) *(The $\widetilde{O}$-notation ignores logarithms in the largest occurrence of a variable; $\widetilde{O}(f(n)) = O(f(n) \cdot (\log f(n))^{O(1)})$.)*

Furthermore, for any $\beta > 0$, one can distinguish between points sets in R^d that are c-clusterable from those ϵ-far from $(1+\beta)$c-clusterable with the sample complexity $\widetilde{O}(k^2/(\beta^2\epsilon))$.

Theorem 3 *For the diameter k-clustering, one can distinguish between points sets in R^d that are c-clusterable from those ϵ-far from $(1+\beta)$c-clusterable with the sample complexity $\widetilde{O}(k^2 d(2/\beta)^{2d}/\epsilon)$* (Alon et al., 2003).

Core-Sets: Sublinear Space Representations with Applications

A *core-set* is a small weighted set of points S that provably approximates another point set P with respect to a given clustering problem (Bădoiu, Har-Peled, & Indyk, 2002). The precise definition of a core-set depends on the clustering objective function and the notion of approximation. For example, a coreset for the k-median problem can be defined as follows:

Definition 4 *A weighted point set S is called* ϵ-coreset *for a point set P for the k-median problem, if for every set C of k centers, we have $(1-\epsilon) \cdot median(P,C) \leq median(S,C) \leq (1+\epsilon) \cdot median(P,C)$* (Har-Peled & Mazumdar, 2004).

A core-set as defined above is also sometimes called a *strong* core-set, because the cost of the objective function is approximately preserved for any set of cluster centers. In some cases it can be helpful to only require a weaker notion of approximation. For example, for some applications it is sufficient that the cost is preserved for a certain discrete set of candidate solutions. Such a core-set is usually called a *weak* core-set. In high-dimensional applications it is sometimes sufficient that the solution is contained in the low-dimensional subspace spanned by the core-set points.

Constructing a Core-Set: There are deterministic and randomized constructions for core-sets of an unweighted set P of n points in the R^d. Deterministic core-set constructions are usually based on the *movement paradigm*. The input points are moved to a set of few locations such that the overall movement is at most ϵ times the cost of an optimal solution. Then the set of points at the same location are replaced by a single point whose weight equals the number of these points. Since for the k-median problem the cost of any solution changes by at most the overall movement, this immediately implies that the constructed weighted set is an ϵ-core-set. For other similar problems more involved arguments can be used to prove the core-set property. Based on the movement paradigm, for k-median a core-set of size $O(k \log n/\epsilon^d)$ can be constructed efficiently (Har-Peled & Mazumdar, 2004).

Randomized core-set constructions are based on non uniform sampling. The challenge is to define a randomized process for which the resulting weighted point set is with high probability a core-set. Most randomized coreset constructions first compute a bi-criteria approximation C'. Then every point is sampled with probability proportional to its distance to the nearest center of C'. A point q is assigned a weight proportional to $1/p_q$, where p_q is the probability that p is sampled. For every fixed set C of k centers, the resulting sample is an unbiased estimator for $median(P,C)$. If the sample set is large enough, it approximates the cost of every possible set of k centers within a factor of $(1 \pm \epsilon)$. The above approach can be used to obtain a weak core-set of size independent of the size of the input point set and the dimension of the input space (Feldman, Monemizadeh, & Sohler, 2007). A related construction has been previously used to obtain a strong core-set of size $\widetilde{O}(k^2 \cdot d \cdot \log n/\epsilon^2)$. Both constructions have constant success probability that can be improved by increasing the size of the core-set.

Applications Core-sets have been used to obtain improved approximation algorithms for different variants of clustering problems. Since the core-sets are of sublinear size and they can be constructed in sublinear time, they can be used to obtain sublinear-time approximation algorithms for a number of clustering problems.

Another important application is clustering of data streams. A data stream is a long sequence of data that typically does not fit into main memory, for example, a sequence of packet headers in IP traffic monitoring. To analyze data streams we need

algorithms that extract information from a stream without storing all of the observed data. Therefore, in the data streaming model algorithms are required to use $\log^{O(1)} n$ bits of memory. For core-sets, a simple but rather general technique is known, which turns a given construction of a strong core-set into a data streaming algorithm, i.e., an algorithm that processes the input points sequentially and uses only $\log^{O(1)}$ space (for constant k and ϵ) and computes a $(1+\epsilon)$-approximation for the optimal set of centers of the k-median clustering (Har-Peled & Mazumdar, 2004). Core-sets can also be used to improve the running time and stability of clustering algorithms like the k-means algorithm (Frahling & Sohler, 2006).

Recommended Reading

Alon, N., Dar, S., Parnas, M., & Ron, D. (2003). Testing of clustering. *SIAM Journal on Discrete Mathematics, 16*(3), 393–417.

Bădoiu, M., Har-Peled, S., & Indyk, P. (2002). Approximate clustering via core-sets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, (pp. 250–257).

Ben-David, S. (2004). A framework for statistical clustering with a constant time approximation algorithms for k-median clustering. In *Proceedings of the 17th Annual Conference on Learning Theory (COLT)*, (pp. 415–426).

Chen, K. (2006). On k-median clustering in high dimensions. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (pp. 1177–1185).

Czumaj, A., & Sohler, C. (2007). Sublinear-time approximation for clustering via random sampling. *Random Structures & Algorithms, 30*(1–2), 226–256.

Feldman, D., Monemizadeh, M., & Sohler, C. (2007). A PTAS for k-means clustering based on weak coresets. In *Proceedings of the 23rd Annual ACM Symposium on Computational Geometry (SoCG)*, (pp. 11–18).

Frahling, G., & Sohler, C. (2006). A fast k-means implementation using coresets. In *Proceedings of the 22nd Annual ACM Symposium on Computational Geometry (SoCG)*, (pp. 135–143).

Har-Peled, S. & Kushal, A. (2005). Smaller coresets for k-median and k-means clustering. In *Proceedings of the 21st Annual ACM Symposium on Computational Geometry (SoCG)*, (pp. 126–134).

Har-Peled, S., & Mazumdar, S. (2004). On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, (pp. 291–300).

Meyerson, A., O'Callaghan, L., & Plotkin S.(July 2004). A k-median algorithm with running time independent of data size. *Machine Learning, 56*(1–3), (pp. 61–87).

Mishra, N., Oblinger, D., & Pitt, L. (2001). Sublinear time approximate clustering. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (pp. 439–447).

Subspace Clustering

►Projective Clustering

Subsumption

Claude Sammut
The University of New South Wales,
Sydney NSW, Australia

Subsumption provides a syntactic notion of generality. Generality can simply be defined in terms of the cover of a concept. That is, a concept, C, is more general than a concept, C', if C covers at least as many examples as C' (see ►Learning as Search). However, this does not tell us how to determine, from their syntax, if one sentence in a concept description language is more general than another. When we define a *subsumption* relation for a language, we provide a syntactic analogue of generality (Lavrač & Džeroski, 1994). For example, *θ-subsumption* (Plotkin, 1970) is the basis for constructing generalization lattices in ►inductive logic programming (Shapiro, 1981). See ►Generality of Logic for a definition of *θ-subsumption*. An example of defining a subsumption relation for a domain specific language is in the LEX program (Mitchell, Utgoff, & Banerji, 1983), where an ordering on mathematical expressions is given.

Cross References

►Generalization
►Induction
►Learning as Search
►Logic of Generality

Recommended Reading

Lavrač, N., & Džeroski, S. (1994). *Inductive Logic Programming: Techniques and applications*. Chichester: Ellis Horwood.

Mitchell, T. M., Utgoff, P. E., & Banerji, R. B. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. Palo Alto: Tioga.

Plotkin, G. D. (1970). A note on inductive generalization. In B. Meltzer & D. Michie (Eds.), *Machine intelligence* (Vol. 5, pp. 153–163). Edinburgh University Press.

Shapiro, E. Y. (1981). An algorithm that infers theories from facts. In *Proceedings of the seventh international joint conference on artificial intelligence, Vancouver* (pp. 446–451). Los Altos: Morgan Kaufmann.

Supersmoothing

►Local Distance Metric Adaptation

Supervised Descriptive Rule Induction

Petra Kralj Novak[1], Nada Lavrač[1,2], Geoffrey I. Webb[3]
[1]Jožef Stefan Institute, Ljubljana, Slovenia
[2]University of Nova Gorica, Nova Gorica, Slovenia
[3]Monash University, Clayton, VIC, Australia

Synonyms

SDRI

Definition

Supervised descriptive rule induction (SDRI) is a machine learning task in which individual patterns in the form of rules (see ►Classification rule) intended for interpretation are induced from data, labeled by a predefined property of interest. In contrast to standard ►supervised rule induction, which aims at learning a set of rules defining a classification/prediction model, the goal of SDRI is to induce individual descriptive patterns. In this respect SDRI is similar to ►association rule discovery, but the consequents of the rules are restricted to a single variable – the property of interest – and, except for the discrete target attribute, the data is not necessarily assumed to be discrete.

Supervised descriptive rule induction assumes a set of training examples, described by attributes and their values and a selected attribute of interest (called the target attribute). Supervised descriptive rule induction induces rules that may each be interpreted independently of the others. Each rule is a ►local model, covering a subset of training examples, that captures a local relationship between the target attribute and the other attributes.

Induced descriptive rules are mainly aimed at human interpretation. More specifically, the purposes of supervised descriptive rule induction are to allow the user to gain insights into the data domain and to better understand the phenomena underlying the data.

Motivation and Background

Symbolic data analysis techniques aim at discovering comprehensible patterns or ►models in data. They can be divided into techniques for *predictive induction*, where models, typically induced from class labeled data, are used to predict the class value of previously unseen examples, and *descriptive induction*, where the aim is to find comprehensible patterns, typically induced from unlabeled data. Until recently, these techniques have been investigated by two different research communities: predictive induction mainly by the machine learning community, and descriptive induction mainly by the data mining community.

Data mining tasks where the goal is to find comprehensible patterns from labeled data have been addressed by both the machine learning and the data mining community independently. The data mining community, using the ►association rule learning perspective, adapted association rule learners like ►Apriori (Agrawal, Mannila, Srikant, Toivonon, & Inkeri Verkamo, 1996) to perform tasks on labeled data, like class association rule learning (Jovanovski & Lavrač, 2001; Liu, Hsu, & Ma, 1998), as well as ►contrast set mining (Bay & Pazzani, 2001) and ►emerging pattern mining (Dong & Li, 1999). On the other hand, the machine learning community, which traditionally focused on the induction of ►rule sets from labeled data for the purposes of classification, turned to building individual rules for exploratory data analysis and interpretation. This is the goal of the task named ►subgroup discovery (Wrobel, 1997). These are the main areas of supervised descriptive rule induction. All these areas deal with finding comprehensible rules from class labeled data. However, the methods used and the interpretation of the results differ slightly from approach to approach. Other related approaches include change mining, mining of closed sets for labeled data, exception rule mining, bump hunting, quantitative association rules, and impact rules. See Kralj Novak, Lavrač, and

Webb (2009) for a more detailed survey of supervised descriptive rule induction.

Structure of the Learning System

Supervised descriptive rule induction assumes that there is data with the property of interest defined by the user. Let us illustrate supervised descriptive rule induction using data from Table 1, a very small artificial sample data set, adapted from Quinlan (1986), which contains the results of a survey on 14 individuals, concerning the approval or disproval of an issue analyzed in the survey. Each individual is characterized by four attributes that encode rudimentary information about the sociodemographic background. The last column (Approved) is the designated property of interest, encoding whether the individual approved or disproved the issue. Unlike predictive induction, where the aim is to find a predictive model, the goal of supervised descriptive rule induction is to find local patterns in form of individual rules describing individuals that are likely to approve or disprove the issue, based on the four demographic characteristics.

Figure 1 shows six descriptive rules, found for the sample data using the Magnum Opus (Webb, 1995) rule learning software. These rules were found using

Supervised Descriptive Rule Induction. Table 1 A Sample Database

Education	Marital Status	Sex	Has Children	Approved
Primary	Single	Male	No	No
Primary	Single	Male	Yes	No
Primary	Married	Male	No	Yes
University	Divorced	Female	No	Yes
University	Married	Female	Yes	Yes
Secondary	Single	Male	No	No
University	Single	Female	No	Yes
Secondary	Divorced	Female	No	Yes
Secondary	Single	Female	Yes	Yes
Secondary	Married	Male	Yes	Yes
Primary	Married	Female	No	Yes
Secondary	Divorced	Male	Yes	No
University	Divorced	Female	Yes	No
Secondary	Divorced	Male	No	Yes

```
MaritalStatus=single AND Sex=male → Approved=no
Sex=male → Approved=no
Sex=female → Approved=yes
MaritalStatus=married → Approved=yes
MaritalStatus=divorced AND HasChildren=yes → Approved=no
MaritalStatus=single → Approved=no
```

Supervised Descriptive Rule Induction. Figure 1. Selected descriptive rules, describing individual patterns in the data of Table 1

the default settings except that the critical value for the statistical test was relaxed. This set of descriptive rules differs from a typical predictive rule set in several ways. The first rule is redundant with respect to the second. The first rule is included as a strong pattern (all three single males do not approve) whereas the second is weaker but more general (four out of seven males do not approve, which is not highly predictive, but accounts for four out of all five respondents who do not approve). Most predictive systems would include only one of these rules, but either or both of them may be of interest to someone trying to understand the data, depending on the specific application. This particular approach to descriptive pattern discovery does not attempt to guess which of the more specific or more general patterns will be more useful to the end user. Another difference between predictive and descriptive rules is that the predictive approach often includes rules for the sake of completeness, while some descriptive approaches make no attempt at completeness, as they assess each pattern on its individual merits.

Exactly which rules will be induced by a supervised descriptive rule induction algorithm depends on the task definition, the selected algorithm, as well as the user-defined constraints concerning minimal rule support, precision, etc. Different learning approaches and heuristics have been proposed to induce supervised descriptive rules.

Applications

Applications of supervised descriptive rule induction are widely spread. See Kralj Novak et al. (2009) for a detailed survey.

►Subgroup discovery has been used in numerous real-life applications. Medical applications include the analysis of coronary heart disease and brain ischemia data analysis, as well as profiling examiners for sonographic examinations. Spatial subgroup mining applications include mining of census data and mining of vegetation data. There are also applications in marketing and analysis of shop floor data.

►Contrast set mining has been used with retail sales data and for designing customized insurance programs. It has also been used in medical applications to identify patterns in synchrotron X-ray data that distinguish tissue samples of different forms of cancerous tumor and for distinguishing between groups of brain ischemia patients.

►Emerging pattern mining has been mainly applied to the field of bioinformatics, more specifically to microarray data analysis. For example, an interpretable classifier based on simple rules that is competitive to the state of the art black-box classifiers on the acute lymphoblastic leukemia (ALL) microarray data set was built from emerging patterns. Another application was about finding groups of genes by emerging patterns in a ALL/acute myeloblastic leukemia (AML) data set and a colon tumor data set. Emerging patterns were also used together with the unexpected change approach and the added/perished rule to mine customer behavior.

Future Directions

A direction for further research is to decompose SDRI algorithms, preprocessing and evaluation methods into basic components and their reimplementation as connectable web services, which includes the definition of interfaces between SDRI services. For instance, this can include the adaptation and implementation of subgroup discovery techniques to solving open problems in the area of contrast set mining and emerging patterns. This would allow for the improvement of algorithms due to the cross-fertilization of ideas from different SDRI subareas.

Another direction for further research concerns complex data types and the use of background knowledge. The SDRI attempts in this direction include relational subgroup discovery approaches like algorithms Midos (Wrobel, 2001), RSD (Relational Subgroup Discovery) (Železný & Lavrač, 2006), and SubgroupMiner (Klösgen & May, 2002), which is designed for spatial data mining in relational space databases. The search for enriched gene sets (SEGS) method (Trajkovski, Lavrac, & Tolar, 2008) supports building rules when using specialized biological knowledge in the form of ontologies. It is a step toward semantically enabled creative knowledge discovery in the form of descriptive rules.

Cross References

►Apriori
►Association Rule Discovery

►Classification Rule
►Contrast Set Mining
►Emerging Pattern Mining
►Subgroup Discovery
►Supervised Rule Induction

Recommended Reading

Agrawal, R., Mannila, H., Srikant, R., Toivonon, H., & Inkeri Verkamo, A. (1996). Fast discovery of association rules. In *Advances in knowledge discovery and data mining* (pp. 307–328). Menlo Park: American Association for Artificial Intelligence.

Bay, S. D., & Pazzani, M. J. (2001). Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery, 5*(3), 213–246.

Dong, G., & Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining (KDD-99)* (pp. 43–52). New York: ACM.

Jovanovski, V., & Lavrač, N. (2001). Classification rule learning with APRIORI-C. In *Proceedings of the tenth Portuguese conference on artificial intelligence* (pp. 44–51). London: Springer.

Klösgen, W., & May, M. (2002). Spatial subgroup mining integrated in an object-relational spatial database. In *Proceedings of the sixth European conference on principles and practice of knowledge discovery in databases (PKDD-02)* (pp. 275–286). London: Springer.

Kralj Novak, P. Lavrač, N., & Webb, G. I. (February 2009). Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research, 10*, 377–403. Available at: http://www.jmlr.org/papers/volume10/kralj-novak09a/kraljnovak09a.pdf.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. In *Proceedings of the fourth international conference on knowledge discovery and data mining (KDD-98)* (pp. 80–86).

Trajkovski, I., Lavrac, N., & Tolar, J. (2008). SEGS: Search for enriched gene sets in microarray data. *Journal of Biomedical Informatics*, 41(4), 588–601.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*(1), 81–106.

Webb, G. I. (1995). OPUS: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research*, 3, 431–465.

Wrobel, S. (1997). An algorithm for multi-relational discovery of subgroups. In *Proceedings of the first European conference on principles of data mining and knowledge discovery (PKDD-97)* (pp. 78–87). London: Springer.

Wrobel, S. (2001). Inductive logic programming for knowledge discovery in databases. In S. Dzeroski & N. Lavrac (Eds.), *Relational data mining* (Chap. 4, pp. 74–101). Berlin: Springer.

Železný, F., & Lavrac, N. (2006). Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, 62, 33–63.

Supervised Learning

Definition

Supervised learning refers to any machine learning process that learns a function from an input type to an output type using data comprising examples that have both input and output values. Two typical examples of supervised learning are ►classification learning and ►regression. In these cases, the output types are respectively categorical (the classes) and numeric. Supervised learning stands in contrast to ►unsupervised learning, which seeks to learn structure in data, and to ►reinforcement learning in which sequential decision-making policies are learned from reward with no examples of "correct" behavior.

Cross References

►Reinforcement Learning
►Unsupervised Learning

Support Vector Machines

Xinhua Zhang
Australian National University, NICTA London Circuit, Canberra, Australia

Definition

Support vector machines (SVMs) are a class of linear algorithms that can be used for ►classification, ►regression, density estimation, novelty detection, and other applications. In the simplest case of two-class classification, SVMs find a hyperplane that separates the two classes of data with as wide a margin as possible. This leads to good generalization accuracy on unseen data, and supports specialized optimization methods that allow SVM to learn from a large amount of data.

Motivation and Background

Over the past decade, maximum margin models especially SVMs have become popular in machine learning. This technique was developed in three major steps. First, assuming that the two classes of training examples can be separated by a hyperplane, Vapnik and Lerner

proposed in 1963 that the optimal hyperplane is the one that separates the training examples with the widest margin. From the 1960s to 1990s, Vapnik and Chervonenkis developed the Vapnik–Chervonenkis theory, which justifies the maximum margin principle from a statistical point of view. Similar algorithms and optimization techniques were proposed by Mangasarian in 1965.

Second, Boser, Guyon, and Vapnik (1992) incorporated kernel function into the maximum margin models, and their formulation is close to the currently popular form of SVMs. Before that, Wahba (1990) also discussed the use of kernels. Kernels allow SVM to implicitly construct the optimal hyperplane in the feature space, and the resulting nonlinear model is important for modeling real data.

Finally, in case the training examples are not linearly separable, Cortes and Vapnik (1995) showed that the soft margin can be applied, allowing some examples to violate the margin condition.

On the theoretical side, Shawe-Taylor, Bartlett, Williamson, and Anthony (1998) gave the first rigorous statistical bound on the generalization of hard margin SVMs. Shawe-Taylor and Cristianini (2000) gave statistical bounds on the generalization of soft margin algorithms and for the regression case.

In reality SVMs became popular thanks to its significantly better empirical performance than the neural networks. By incorporating transform invariances, the SVMs developed at AT&T achieved the highest accuracy on the MNIST benchmark set (a handwritten digit recognition problem). Joachims (1998) also showed clear superiority of SVMs on text categorization. Afterward, SVMs have been shown effective in many applications including computer vision, natural language, bioinformatics, and finance.

Theory

SVMs have a strong mathematical basis and are closely related to some well-established theories in statistics. They not only try to correctly classify the training data, but also maximize the margin for better generalization performance. This formulation leads to a separating hyperplane that depends only on the (usually small fraction of) data points that lie on the margin, which are called support vectors. Hence the whole algorithm is called *support vector machine*. In addition, since real–world data analysis problems often involve nonlinear dependencies, SVMs can be easily extended to model such nonlinearity by means of positive semidefinite kernels. Moreover, SVMs can be trained via quadratic programming, which (a) makes theoretical analysis easier, and (b) provides much convenience in designing efficient solvers that scale for large datasets. Finally, when applied to real-world data, SVMs often deliver state-of-the-art performance in accuracy, flexibility, robustness, and efficiency.

Optimal Hyperplane for Linearly Separable Examples

Consider the training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^p$ is the input feature vector for the i-th example, and $y_i \in \{1, -1\}$ is its corresponding label indicating whether the example is positive ($y_i = +1$) or negative ($y_i = -1$). To begin with, we assume that the set of positive and negative examples are linearly separable, that is, there exists a function $f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ where $\boldsymbol{w} \in \mathbb{R}^p$ (called the weight vector) and $b \in \mathbb{R}$ (called bias) such that

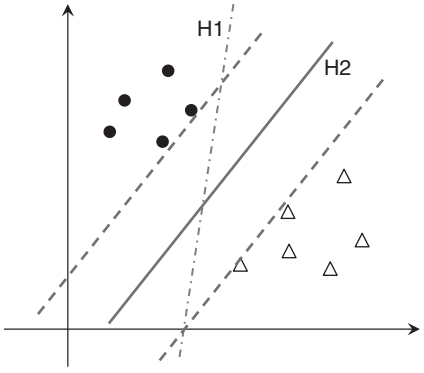
$$\begin{aligned} \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b > 0 \quad & \text{for } y_i = +1 \\ \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b < 0 \quad & \text{for } y_i = -1. \end{aligned}$$

We call $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0$ the decision hyperplane and in fact, there can exist multiple hyperplanes that separate the positive and negative examples, see Fig. 1. However, they are not created equal. Associated with each such hyperplane is a notion called *margin*, defined as the distance between the hyperplane and the closest example. SVM aims to find the particular hyperplane that maximizes the margin.

Mathematically, it is easy to check that the distance from a point $\boldsymbol{x}_i$ to a hyperplane $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0$ is $\|\boldsymbol{w}\|^{-1} |\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b|$. Therefore, SVM seeks for the optimal $\boldsymbol{w}, b$ of the following optimization problem:

$$\begin{aligned} & \underset{\boldsymbol{w} \in \mathbb{R}^p,\ b \in \mathbb{R}}{\text{maximize}} \ \min_{1 \le i \le n} \frac{|\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b|}{\|\boldsymbol{w}\|}, \\ & \text{s.t.} \begin{cases} \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b > 0 & \text{if } y_i = +1 \\ \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b < 0 & \text{if } y_i = -1 \end{cases} \quad \forall i \,. \end{aligned}$$

It is clear that if $(\boldsymbol{w}, b)$ is an optimal solution, then $(\alpha \boldsymbol{w}, \alpha b)$ is also an optimal solution for any $\alpha > 0$.



Support Vector Machines. Figure 1. Example of maximum margin separator. Both H1 and H2 correctly separate the examples from the two classes. But H2 has a wider margin than H1

Therefore, to fix the scale, we can equivalently set the numerator of the objective $\min_{1\le i\le n} |\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b|$ to 1, and minimize the denominator $\|\boldsymbol{w}\|$:

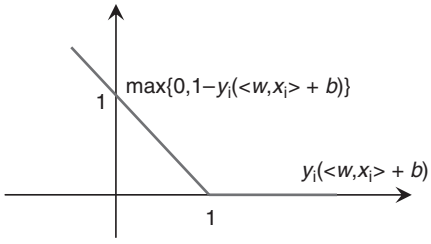
$$\underset{\boldsymbol{w}\in\mathbb{R}^p,\ b\in\mathbb{R}}{\text{minimize}}\ \|\boldsymbol{w}\|^2,$$
$$\text{s.t.}\begin{cases}\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b \ge 1 & \text{if } y_i = +1\\ \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b \le -1 & \text{if } y_i = -1.\end{cases} \quad \forall i\,. \qquad (1)$$

This is a linearly constrained quadratic program, which can be solved efficiently. Hence, it becomes the most commonly used (primal) form of SVM for the linearly separable case.

Soft Margins

In practice, most, if not all, datasets are not linearly separable, that is, no $\boldsymbol{w}$ and b can satisfy the constraints of the optimization problem (1). In this case, we will allow some data points to violate the margin condition, and penalize it accordingly. Mathematically, notice that the constraints in (1) can be equivalently written as $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \ge 1$. Now we introduce a new set of nonnegative slack variables ξ_i into the constraints:

$$y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \ge 1 - \xi_i\,,$$



Support Vector Machines. Figure 2. Graph of hinge loss

and incorporate a penalty into the original objective to derive the soft margin SVM:

$$\underset{\boldsymbol{w},b,\xi_i}{\text{minimize}}\ \lambda\|\boldsymbol{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i$$
$$\text{s.t. } y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \ge 1 - \xi_i, \text{ and } \xi_i > 0\ \ \forall i\,. \qquad (2)$$

$\lambda > 0$ is a trade-off factor. It is important to note that ξ_i can be written as $\xi_i = \max\{0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b)\}$, which is called hinge loss and is depicted in Fig. 2. This way, the optimization problem can be reformulated into an unconstrained non-smooth problem:

$$\underset{\boldsymbol{w}\in\mathbb{R}^p,\ b\in\mathbb{R}}{\text{minimize}}\ \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n}$$
$$\max\{0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b)\}\,. \qquad (3)$$

Notice that $\max\{0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b)\}$ is also a convex upper bound of $\delta(y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) > 0)$, where $\delta(x) = 1$ if x is true and 0 otherwise. Therefore, the penalty we use is a convex upper bound of the average training error. When the training set is actually separable, the soft margin problem (2) automatically recovers the hard margin problem (1) when λ is sufficiently large.

Dual Forms and Kernelization

As the constraints in the primal form (2) are not convenient to handle, people have conventionally resorted to the dual problem of (2). Following the standard procedures, one can derive the Lagrangian dual

$$\min_{\boldsymbol{\alpha}} \frac{1}{2\lambda}\sum_{i,j} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle - \sum_i \alpha_i,$$
$$\text{s.t. } \alpha_i \in [0, n^{-1}], \text{ and } \sum_i y_i \alpha_i = 0\,. \qquad (4)$$

which is again a quadratic program, but with much simpler constraints: box constraints plus a single linear equality constraint. To recover the primal solution $\boldsymbol{w}^*$ from the dual solution α_i^*, we have

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i \,,$$

and the optimal bias b can be determined by using the duality conditions.

The training examples can be divided into three categories according to the value of α_i^*. If $\alpha_i^* = 0$, it means the corresponding training example does not affect the decision boundary, and in fact it lies beyond the margin, that is, $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) > 1$. If $\alpha_i^* \in (0, n^{-1})$, then the training example lies on the margin, that is, $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) = 1$. If $\alpha_i^* = n^{-1}$ it means the training example violates the margin, that is, $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) < 1$. In the latter two cases where $\alpha_i^* > 0$, the i-th training example is called a support vector.

In many applications, most α_i^* in the optimal solution are 0, which gives a sparse solution. As the final classifier depends only on those support vectors, the whole algorithm is named support vector machines.

From the dual problem (4), a key observation can be drawn that the feature of the training examples $\{\boldsymbol{x}_i\}$ influences training only via the inner product $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$. Therefore, we can redefine the feature by mapping $\boldsymbol{x}_i$ to a richer feature space via $\phi(\boldsymbol{x}_i)$ and then compute the inner product there: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) := \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$. Furthermore, one can even directly define k without explicitly specifying ϕ. This allows us to (a) implicitly use a rich feature space whose dimension can be infinitely high, and (b) apply SVM to non-Euclidean spaces as long as a kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ can be defined on it. Examples include strings and graphs (Haussler, 1999), which have been widely applied in bioinformatics (Schölkopf, Tsuda, & Vert, 2004). Mathematically, the objective (4) can be kernelized into

$$\frac{1}{2\lambda} \sum_{i,j} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_i \alpha_i,$$
$$\text{s.t. } \alpha_i \in [0, n^{-1}], \text{ and } \sum_i y_i \alpha_i = 0 \,. \qquad (5)$$

However, now the $\boldsymbol{w}$ cannot be expressed just in terms of kernels because $\boldsymbol{w}^* = \sum_{i=1}^n \alpha_i^* y_i \phi(\boldsymbol{x}_i)$. Fortunately, when predicting on a new example $\boldsymbol{x}$ we again only require the inner product and hence use kernel only:

$$\langle \boldsymbol{w}^*, \boldsymbol{x} \rangle = \sum_{i=1}^{n} \alpha_i^* y_i \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}) \rangle = \sum_{i=1}^{n} \alpha_i^* y_i k(\boldsymbol{x}_i, \boldsymbol{x}) \,.$$

Commonly used kernels on $\mathbb{R}^n$ include polynomial kernels $(1 + \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle)^d$, Gaussian RBF kernels $\exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$, Laplace RBF kernels $\exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$, etc. Kernels on strings and trees are usually based on convolution, which requires involved algorithms for efficient evaluation (Borgwardt, 2007; Haussler, 1999). More details can be found in the kernel section.

Optimization Techniques and Toolkits

The main challenge of optimization lies in scaling for large datasets, that is, n and p are large. Decomposition method based on the dual problem is the first popularly used method for solving large scale SVMs. For example, sequential minimal optimization (SMO) optimizes two dual variables α_i, α_j analytically in each iteration (Platt, 1999a). An SMO-type implementation is available in the LibSVM package http://www.csie.ntu.edu.tw/~cjlin/libsvm. Another popular package using decomposition methods is the SVM-light, available at http://svmlight.joachims.org. Coordinate descent in the dual is also effective and converges at linear rate. An implementation can be downloaded from http://www.csie.ntu.edu.tw/~cjlin/liblinear.

Primal methods are also popular, most of which are based on formulating the objective as a non-smooth objective function like (3). An important type is the subgradient descent method, which is similar to gradient descent but uses a subgradient due to the non-smooth objective. When the dataset is large, one can further use a random subset of training examples to efficiently compute the (approximate) subgradient, and algorithms exist that guarantee the convergence in probability. This is called stochastic subgradient descent, and in practice, it can often quickly find a reasonably good solution. A package that implements this idea can be found at http://leon.bottou.org/projects/sgd.

Finally, cutting plane and bundle methods are also effective (Tsochantaridis, Joachims, Hofmann, & Altun, 2005; Smola, Vishwanathan, & Le, 2007), and they are especially useful for generalized SVMs with structured

outputs. An implementation is the bundle method for risk minimization (BMRM), available for download at http://users.rsise.anu.edu.au/~chteo/BMRM.html.

Applications

The above description of SVM focused on binary class classification. In fact, SVM, or the ideas of maximum margin and kernel, have been widely used in many other learning problems such as regression, ranking and ordinal regression, density estimation, novelty detection, quantile regression, and etc. Even in classification, SVM has been extended to the case of multi-class, multi-label, and structured output (Taskar, 2004; Tsochantaridis et al., 2005).

For multi-class classification and structured output classification where the possible label set $\mathcal{Y}$ can be large, maximum margin machines can be formulated by introducing a joint feature map ϕ on pairs of (x_i, y) $(y \in \mathcal{Y})$. Letting $\Delta(y_i, y)$ be the discrepancy between the true label y_i and the candidate label y, the primal form can be written as

$$\begin{aligned} &\text{minimize}_{w,\xi_i} \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i, \\ &\text{s.t. } \langle w, \phi(x_i, y_i) - \phi(x_i, y)\rangle \geq \Delta(y_i, y) - \xi_i, \ \forall\, i, y, \end{aligned}$$

and the dual form is

$$\begin{aligned} \text{minimize}_{\alpha_{i,y}} \frac{1}{2\lambda} &\sum_{(i,y),(i',y')} \alpha_{i,y}\alpha_{i',y'} \langle \phi(x_i, y_i) - \phi(x_i, y) \\ &\phi(x_{i'}, y_{i'}) - \phi(x_{i'}, y')\rangle - \sum_{i,y} \Delta(y_i, y)\alpha_{i,y} \\ \text{s.t.} \quad &\alpha_{i,y} \geq 0, \ \forall\, i, y; \quad \sum_{y} \alpha_{i,y} = \frac{1}{n}, \ \forall i. \end{aligned}$$

Again kernelization is convenient, by simply replacing all the inner products $\langle \phi(x_i, y), \phi(x_{i'}, y')\rangle$ with a joint kernel $k((x_i, y), (x_{i'}, y'))$. Further factorization using graphical models is possible, see Taskar (2004). Notice when $\mathcal{Y} = \{1, -1\}$, setting $\phi(x_i, y) = y\phi(x_i)$ recovers the binary SVM formulation. Effective methods to optimize the dual objective include SMO, exponentiated gradient descent, mirror descent, cutting plane, or bundle methods.

In general, SVMs are not trained to output the odds of class membership, although the posterior probability is desired to enable post-processing. Platt (1999b) proposed training an SVM, and then train the parameters of an additional sigmoid function to map the SVM outputs into probabilities. A more principled approach is the relevance vector machine, which has an identical functional form to the SVMs and uses Bayesian inference to obtain sparse solutions for probabilistic classification.

As mentioned above, the hinge loss used in SVM is essentially a convex surrogate of the misclassification loss, that is, 1 if the current weight $\boldsymbol{w}$ misclassifies the training example and 0 otherwise. Minimizing the misclassification loss is proved NP-hard, so for computational convenience continuous convex surrogates are used, including hinge loss, exponential loss, and logistic loss. Their statistical properties are studied by Jordan, Bartlett, and McAuliffe (2003). For hinge loss, it has the significant merit of sparsity in the dual, which leads to robustness and good generalization performance.

SVMs have been widely applied in real-world problems. In history, its first practical success was gained in handwritten digit recognition. By incorporating transform invariances, the SVMs developed at AT&T achieved the highest accuracy on the MNIST benchmark set. It has also been very effective in computer vision applications such as object recognition and detection. With the special advantage in handling high-dimensional data, SVMs have witnessed wide application in bioinformatics such as microarray processing (Schölkopf et al., 2004), and natural language processing like named entity recognition, part-of-speech tagging, parsing, and chunking (Joachims, 1998; Taskar, 2004).

Cross References

►Kernel Methods
►Radial Basis Function Networks

Further Reading

A comprehensive treatment of SVMs can be found in Schölkopf and Smola (2002) and Shawe-Taylor and Cristianini (2004). Some important recent developments of SVMs for structured output are collected in Bakir, Hofmann, Schölkopf, Smola, Taskar, and Vishwanathan (2007). As far as applications are concerned,

see Lampert (2009) for computer vision and Schölkopf et al. (2004) for bioinformatics. Finally, Vapnik (1998) provides the details on statistical learning theory.

Recommended Reading

Bakir, G., Hofmann, T., Schölkopf, B., Smola, A., Taskar, B., & Vishwanathan, S. V. N. (2007). *Predicting structured data*. Cambridge: MIT Press.

Borgwardt, K. M. (2007). *Graph Kernels*. Ph.D. thesis, Ludwig-Maximilians-University, Munich, Germany.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In D. Haussler (Ed.), *Proceedings of annual conference computational learning theory* (pp. 144–152). Pittsburgh: ACM Press.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*(3), 273–297.

Haussler, D. (1999). *Convolution kernels on discrete structures* (Tech. Rep. UCS-CRL-99-10). University of California, Santa Cruz.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European conference on machine learning* (pp. 137–142). Berlin: Springer.

Jordan, M. I., Bartlett, P. L., & McAuliffe, J. D. (2003). *Convexity, classification, and risk bounds* (Tech. Rep. 638). University of California, Berkeley.

Lampert, C. H. (2009). Kernel methods in computer vision. *Foundations and Trends in Computer Graphics and Vision*, *4*(3), 193–285.

Platt, J. C. (1999a). Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods—support vector learning* (pp. 185–208). Cambridge, MA: MIT Press.

Platt, J. C. (1999b). Probabilities for sv machines. In A. J. Smola, P. L. Bartlett, B. Schölkopf, & D. Schuurmans, (Eds.), *Advances in large margin classifiers* (pp. 61–74). Cambridge: MIT Press.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels*. Cambridge: MIT Press.

Schölkopf, B., Tsuda, K., & Vert, J.-P. (2004). *Kernel methods in computational biology*. Cambridge: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2000). Margin distribution and soft margin. In A. J. Smola, P. L. Bartlett, B. Schölkopf, & D. Schuurmans, (Eds.), *Advances in large margin classifiers* (pp. 349–358). Cambridge: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.

Shawe-Taylor, J., Bartlett, P. L., Williamson, R. C., & Anthony, M. (1998). Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, *44*(5), 1926–1940.

Smola, A., Vishwanathan, S. V. N., & Le, Q. (2007). Bundle methods for machine learning. In D. Koller, & Y. Singer, (Eds.), *Advances in neural information processing systems* (Vol. 20). Cambridge: MIT Press.

Taskar, B. (2004). *Learning structured prediction models: A large margin approach*. Ph.D. thesis, Stanford University.

Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, *6*, 1453–1484.

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.

Wahba, G. (1990). *Spline models for observational data*. *CBMS-NSF regional conference series in applied mathematics* (Vol. 59). Philadelphia: SIAM.

Swarm Intelligence

Swarm intelligence is the discipline that studies the collective behavior of systems composed of many individuals that interact locally with each other and with their environment and that rely on forms of decentralized control and self-organization. Examples of such systems are colonies of ants and termites, schools of fish, flocks of birds, herds of land animals, and also some artifacts, including swarm robotic systems and some computer programs for tackling optimization problems such as ►ant colony optimization and ►particle swarm optimization.

Symbolic Dynamic Programming

Scott Sanner[1], Kristian Kersting[2]
[1]Statistical Machine Learning Group, NICTA, Canberra, ACT, Australia
[2]Fraunhofer IAIS, Sankt Augustin, Germany

Synonyms

Dynamic programming for relational domains; Relational dynamic programming; Relational value iteration; SDP

Definition

Symbolic dynamic programming (SDP) is a generalization of the ►dynamic programming technique for solving ►Markov decision processes (MDPs) that exploits the symbolic structure in the solution of relational and

first-order logical MDPs through a lifted version of dynamic programming.

Motivation and Background

Decision-theoretic planning aims at constructing a policy for acting in an uncertain environment that maximizes an agent's expected utility along a sequence of steps. For this task, Markov decision processes (MDPs) have become the standard model. However, classical dynamic programming algorithms for solving MDPs require explicit state and action enumeration, which is often impractical: the number of states and actions grows very quickly with the number of domain objects and relations. In contrast, SDP algorithms seek to avoid explicit state and action enumeration through the symbolic representation of an MDP and a corresponding symbolic derivation of its solution, such as a value function. In essence, SDP algorithms exploit the symbolic structure of the MDP representation to construct a minimal logical partition of the state space required to make all necessary value distinctions.

Theory and Solution

Consider an agent acting in a simple variant of the BoxWorld problem. There are several cities such as *London*, *Paris* etc., trucks $truck_1$, $truck_2$ etc., and boxes box_1, box_2 etc. The agent can load a box onto a truck or unload it and can drive a truck from one city to another. Only when a particular box, say box box_1, is in a particular city, say *Paris*, the agent receives a positive reward. The agent's learning task is now to find a policy for action selection that maximizes its reward over the long term.

A great variety of techniques for solving such decision-theoretic planning tasks have been developed over the last decades. Most of them assume atomic representations, which essentially amounts to enumerating all unique configurations of trucks, cities, and boxes. It might then be possible to learn, for example, that taking action *action234* in state *state42* is worth 6.2 and leads to state *state654321*. Atomic representations are simple, and learning can be implemented using simple lookup tables. These lookup tables, however, can be intractably large as atomic representations easily explode. Furthermore, they do not easily generalize across different numbers of domain objects (We use the term *domain* in the first-order logical sense of an object universe. The term should not be confused with a planning *problem* such as BoxWorld or BlocksWorld.).

In contrast, SDP assumes a relational or first-order logical representation of an MDP (as given in Fig. 1) to exploit the existence of domain objects, relations over these objects, and the ability to express objectives and action effects using quantification.

It is then possible to learn that to get box b to *paris*, the agent drives a truck to the city of b, loads box_1 on the truck, drives the truck to *Paris*, and finally unloads the box box_1 in *Paris*. This is essentially encoded in the symbolic value function shown in Fig. 2, which was computed by discounting rewards t time steps into the future by 0.9^t. The key features to note here are the state and action abstraction in the value and policy representation that are afforded by the first-order specification and solution of the problem. That is, this solution does not refer to any specific set of domain objects, such as $City = \{paris, berlin, london\}$, but rather it provides a solution for *all possible domain object instantiations*. And while classical dynamic programming techniques could never solve these problems for large domain instantiations (since they would have to enumerate all states and actions), a domain-independent SDP solution to this particular problem is quite simple due to the power of state and action abstraction.

Background: Markov Decision Processes (MDPs)

In the MDP (Puterman, 1994) model, an agent is assumed to fully observe the current state and choose an action to execute from that state. Based on that state and action, nature then chooses a next state according to some fixed probability distribution. In an infinite-horizon MDP, this process repeats itself indefinitely. Assuming there is a reward associated with each state and action, the goal of the agent is to maximize the expected sum of discounted rewards received over an infinite horizon (Although we do not discuss it here, there are other alternatives to discounting such as averaging the reward received over an infinite horizon.). This criterion assumes that a reward received t steps in the future is discounted by γ^t, where γ is a discount factor satisfying $0 \leq \gamma < 1$. The goal of the agent is to choose its actions in order to maximize the expected, discounted future reward in this model.

- *Domain Object Types (i.e., sorts)*: *Box*, *Truck*, *City* = $\{paris, \ldots\}$
- *Relations (with parameter sorts)*:
  $BoxIn(Box, City)$, $TruckIn(Truck, City)$, $BoxOn(Box, Truck)$
- *Reward*: if $\exists b.BoxIn(b, paris)$ 10 else 0
- *Actions (with parameter sorts)*:
  - $load(Box : b, Truck : t, City : c)$:
    - Success Probability: if $(BoxIn(b, c) \wedge TruckIn(t, c))$ then .9 else 0
    - Add Effects on Success: $\{BoxOn(b, t)\}$
    - Delete Effects on Success: $\{BoxIn(b, c)\}$
  - $unload(Box : b, Truck : t, City : c)$:
    - Success Probability: if $(BoxOn(b, t) \wedge TruckIn(t, c))$ then .9 else 0
    - Add Effects on Success: $\{BoxIn(b, c)\}$
    - Delete Effects on Success: $\{BoxOn(b, t)\}$
  - $drive(Truck : t, City : c_1, City : c_2)$:
    - Success Probability: if $(TruckIn(t, c_1))$ then 1 else 0
    - Add Effects on Success: $\{TruckIn(t, c_2)\}$
    - Delete Effects on Success: $\{TruckIn(t, c_1)\}$
  - *noop*
    - Success Probability: 1
    - Add Effects on Success: $\varnothing$
    - Delete Effects on Success: $\varnothing$

Symbolic Dynamic Programming. Figure 1. A formal description of the BoxWorld adapted from Boutilier, Reiter, and Price (2001). We use a simple STRIPS (Fikes & Nilsson, 1971) add and delete list representation of actions and, as a simple probabilistic extension in the spirit of PSTRIPS (Kushmerick, Hanks, & Weld, 1995), we assign probabilities that an action successfully executes conditioned on various state properties

if $(\exists b.BoxIn(b, paris))$ then do *noop* (value = 100.00)
else if $(\exists b,t.TruckIn(t, paris) \wedge BoxOn(b, t))$ then do $unload(b, t)$ (value = 89.0)
else if $(\exists b,c,t.BoxOn(b, t) \wedge TruckIn(t, c))$ then do $drive(t, c, paris)$ (value = 80.0)
else if $(\exists b,c,t.BoxIn(b, c) \wedge TruckIn(t, c))$ then do $load(b, t)$ (value = 72.0)
else if $(\exists b, c_1, t, c_2.BoxIn(b, c_1) \wedge TruckIn(t, c_2))$ then do $drive(t, c_2, c_1)$ (value = 64.7)
else do *noop* (value = 0.0)

Symbolic Dynamic Programming. Figure 2. A decision-list representation of the optimal policy and expected discounted reward value for the BoxWorld problem

Formally, a finite state and action MDP is a tuple: $\langle S, A, T, R \rangle$, where S is a finite state space, A is a finite set of actions, T is a transition function: $T : S \times A \times S \to [0, 1]$, where $T(s, a, \cdot)$ is a probability distribution over S for any $s \in S$ and $a \in A$, and R is a bounded reward function $R : S \times A \to \mathbb{R}$.

As stated earlier, our goal is to find a policy that maximizes the infinite horizon, discounted reward criterion: $E_\pi[\sum_{t=0}^{\infty} \gamma^t \cdot r_t | s_0]$, where r_t is a reward obtained at time t, γ is a discount factor as defined earlier, π is the policy being executed, and s_0 is the initial starting state. Based on this reward criterion, we define the value function

for a policy π as the following:

$$V_\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t \;\middle|\; s_0 = s\right] \quad (1)$$

Intuitively, the value function for a policy π is the expected sum of discounted rewards accumulated while executing that policy when starting from state s.

For the MDP model discussed here, the optimal policy can be shown to be stationary (Puterman, 1994). Consequently, we use a stationary policy representation of the form $\pi : S \to A$, with $\pi(s)$ denoting the action to be executed in state s. An optimal policy π^* is the policy that maximizes the value function for all states. We denote the optimal value function over an indefinite horizon as $V^*(s)$ and note that it satisfies the following equality:

$$V^*(s) = \max_{a \in A}\left\{R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \cdot V^*(s')\right\} \quad (2)$$

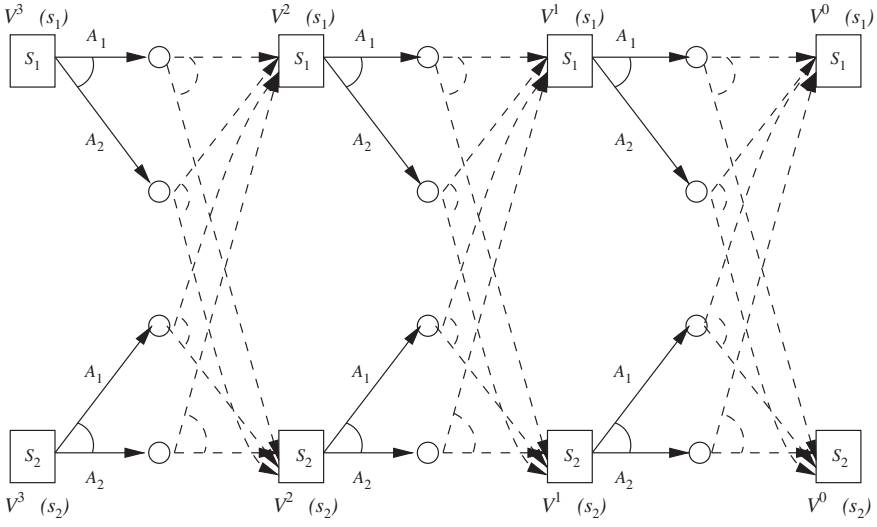
Bellman's *principle of optimality* (Bellman, 1957) establishes the following relationship between the optimal value function $V^t(s)$ with a finite horizon of t steps remaining and the optimal value function $V^{t-1}(s)$ with a finite horizon of $t-1$ steps remaining:

$$V^t(s) = \max_{a \in A}\left\{R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \cdot V^{t-1}(s')\right\} \quad (3)$$

A *dynamic programming* approach for computing the optimal value function over an indefinite horizon is known as value iteration and directly implements (3) to compute 1 by successive approximation. As sketched in Fig. 3, we start with arbitrary $V^0(s)$ (e.g., $\forall s\, V^0(s) = 0$) and perform the Bellman backup given in (3) for each state $V^1(s)$ using the value of $V^0(s)$. We repeat this process for each t to compute $V^t(s)$ from the memorized values for $V^{t-1}(s)$ until we have computed the intended t-stages-to-go value function. $V^t(s)$ will converge to $V^*(s)$ as $t \to \infty$ (Puterman, 1994).

Often, the Bellman backup is rewritten in two steps to separate out the action regression and maximization steps. In this case, we first compute the t-stages-to-go Q-function for every action and state:

$$Q^t(s,a) = R(s,a) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \cdot V^{t-1}(s') \quad (4)$$



Symbolic Dynamic Programming. Figure 3. A diagram demonstrating a *dynamic programming* regression-based evaluation of the MDP value function. Dashed lines are used in the expectation computation of the Q-function: for each action, take the expectation over the values of possible successor states. Solid lines are used in the max computation: determine the highest valued action to be taken in each state

Then we maximize over each action to determine the value of the regressed state:

$$V^t(s) = \max_{a \in A} \{Q^t(s,a)\} \quad (5)$$

This is clearly equivalent to (3) but is in a form that we refer to later, since it separates the algorithm into its two conceptual components: decision-theoretic regression and maximization.

First-Order Markov Decision Processes

A first-order MDP (FOMDP) can be thought of as a universal MDP that abstractly defines the state, action, transition, and reward tuple $\langle S, A, T, R \rangle$ for an infinite number of ground MDPs. To make this idea more concrete, consider the BoxWorld problem defined earlier. While we have not yet formalized the details of the FOMDP representation, it should be clear that the BoxWorld dynamics hold for any instantiation of the domain objects: *Box*, *Truck*, and *City*. For instance, assume that the domain instantiation consists of two boxes $Box = \{box_1, box_2\}$, two trucks $Truck = \{truck_1, truck_2\}$ and two cities $City = \{paris, berlin\}$. Then the resulting ground MDP has 12 state-variable atoms (each atom being *true* or *false* in a state), four atoms for *BoxIn* such as $BoxIn(box_1, paris)$, $BoxIn(box_2, paris), \ldots$, four atoms for *TruckIn* such as $TruckIn(truck_2, paris), \ldots$ and four atoms for *BoxOn* such as $BoxOn(box_2, truck_1), \ldots$. There are also 24 possible actions (eight for each of *load*,*unload*, and *drive*) such as $load(box_1, truck_1, paris)$, $load(box_1, truck_1, berlin)$, $drive(truck_2, paris, paris)$, $drive(truck_2, paris, berlin)$, etc., where the transition function directly follows from the ground instantions of the corresponding PSTRIPS operators. The reward function looks like: if $(BoxIn(box_1, paris) \vee BoxIn(box_2, paris))$ 10 else 0.

Therefore, to solve an FOMDP, we could ground it for a specific domain instantiation to obtain a corresponding ground MDP. Then we could apply classical MDP solution techniques to solve this ground MDP. However, the obvious drawback to this approach is that the number of state variables and actions in the ground MDP grow at least linearly as the domain size increases. And even if the ground MDP could be represented within memory constraints, the number of distinct ground states grows exponentially with the number of state variables, thus rendering solutions that scale with state size intractable even for moderately small numbers of domain objects.

An alternative idea to solving an FOMDP at the ground level is to solve the FOMDP directly at the first-order level using symbolic dynamic programming, thereby obtaining a solution that applies universally to all possible domain instantiations. While the exact representation and SDP solution of FOMDPs differ among the variant formalisms, they all share the same basic first-order representation of rewards, probabilities, and values that we outline next. To highlight this, we introduce a graphical *case notation* to allow first-order specifications of the rewards, probabilities, and values required for FOMDPs:

$$case = \begin{cases} \phi_1 : t_1 \\ \vdots \quad : \; : \\ \phi_n : t_n \end{cases}$$

Here the ϕ_i are *state formulae* and the t_i are terms. Often the t_i are constants and the ϕ_i partition state space. To make this concrete, we represent our BoxWorld FOMDP reward function as the following *rCase* statement:

$$rCase = \begin{cases} \exists b.BoxIn(b, paris) & : 10 \\ \neg \exists b.BoxIn(b, paris) & : 0 \end{cases}$$

Here we see that the first-order formulae in the case statement divide all possible ground states into two regions of constant value: when there exists a box in Paris, a reward of 10 is achieved, otherwise a reward of 0 is achieved. Likewise, the value function *case* that we derive through SDP can be represented in exactly the same manner. Indeed, as we will see shortly, $case^0 = rCase$ in the first-order version of value iteration.

To state the FOMDP transition function for an action, we decompose stochastic "agent" actions into a *collection* of deterministic actions, each corresponding to a possible outcome of the stochastic action. We then specify a distribution according to which "nature" may

choose a deterministic action from this set whenever the stochastic action is executed.

Letting $A(\vec{x})$ be a stochastic action with nature's choices (i.e., deterministic actions) $n_1(\vec{x}), \ldots, n_k(\vec{x})$, we represent the distribution over $n_i(\vec{x})$ given $A(\vec{x})$ using the notation $pCase(n_j(\vec{x}), A(\vec{x}))$. Continuing our logistics example, if the success of driving a truck depends on whether the destination city is *paris* (perhaps due to known traffic delays), then we decompose the stochastic *drive* action into two deterministic actions *driveS* and *driveF*, respectively denoting success and failure. Then we can specify a distribution over nature's choice deterministic outcome for this stochastic action:

$$pCase(driveS(t, c_1, c_2), drive(t, c_1, c_2)) = \begin{array}{|l|}\hline c_2 = paris : 0.6 \\ \hline c_2 \neq paris : 0.9 \\ \hline \end{array}$$

$$pCase(driveF(t, c_1, c_2), drive(t, c_1, c_2)) = \begin{array}{|l|}\hline c_2 = paris : 0.4 \\ \hline c_2 \neq paris : 0.1 \\ \hline \end{array}$$

Intuitively, to perform an operation on case statements, we simply perform the corresponding operation on the intersection of all case partitions of the operands. Letting each ϕ_i and ψ_j denote generic first-order formula, we can perform the "cross-sum" $\oplus$ of case statements in the following manner:

$$\begin{array}{|l|}\hline \phi_1 : 10 \\ \hline \phi_2 : 20 \\ \hline \end{array} \oplus \begin{array}{|l|}\hline \psi_1 : 1 \\ \hline \psi_2 : 2 \\ \hline \end{array} = \begin{array}{|l|}\hline \phi_1 \wedge \psi_1 : 11 \\ \hline \phi_1 \wedge \psi_2 : 12 \\ \hline \phi_2 \wedge \psi_1 : 21 \\ \hline \phi_2 \wedge \psi_2 : 22 \\ \hline \end{array}$$

Likewise, we can perform $\ominus$, $\otimes$, and max operations by respectively subtracting, multiplying, or taking the max of partition values (as opposed to adding them) to obtain the result. Some partitions resulting from the application of the $\oplus$, $\ominus$, and $\otimes$ operators may be inconsistent; we simply discard such partitions (since they can obviously never correspond to any world state).

We define another operation on case statements $\max \exists \vec{x}$ that is crucial for SDP. Intuitively, the meaning of $\max \exists \vec{x}\, case(\vec{x})$ is a case statement where the maximal value is assigned to each region of state space where there exists a satisfying instantiation of $\vec{x}$. To make these ideas concrete, following is an exposition of how the $\max \exists \vec{x}$ may be explicitly computed:

$$\max \exists \vec{x} \begin{array}{|l|}\hline \psi_1(\vec{x}) : 1 \\ \hline \psi_2(\vec{x}) : 2 \\ \hline \psi_3(\vec{x}) : 3 \\ \hline \end{array} = \begin{array}{|ll|}\hline \exists \vec{x}\, \psi_3(\vec{x}) & : 3 \\ \hline \neg(\exists \vec{x}\, \psi_3(\vec{x})) \wedge \exists \vec{x}\, \psi_2(\vec{x}) & : 2 \\ \hline \neg(\exists \vec{x}\, \psi_3(\vec{x})) \wedge \neg(\exists \vec{x}\, \psi_2(\vec{x})) \wedge \exists \vec{x}\, \psi_1(\vec{x}) & : 1 \\ \hline \end{array}$$

Here we have simply sorted partitions in order of values and have ensured that the highest value is assigned to partitions in which there exists a satisfying instantiation of $\vec{x}$ by rendering lower value partitions disjoint from their higher-value antecedents.

Symbolic Dynamic Programming

SDP is a dynamic programming solution to FOMDPs that produces a logical case description of the optimal value function. This is achieved through the operations of first-order decision-theoretic regression (FODTR) and symbolic maximization that perform the traditional dynamic programming Bellman backup at an abstract level without explicit enumeration of either the state or action spaces of the FOMDP. Among many uses, the application of SDP leads to a domain-independent value iteration solution to FOMDPs.

Suppose that we are given a value function in the form *case*. The FODTR (Boutilier et al., 2001) of this value function through an action $A(\vec{x})$ yields a case statement containing the logical description of states and values that would give rise to *case* after doing action $A(\vec{x})$. This is analogous to classical goal regression, the key difference being that action $A(\vec{x})$ is stochastic. In MDP terms, the result of FODTR is a case statement representing a Q-function.

We define the *FODTR* operator in the following manner:

$$FODTR[vcase, A(\vec{x})] = rCase\oplus \quad (6)$$
$$\gamma\,[\oplus_j\{pCase(n_j(\vec{x}))\otimes$$
$$Regr[vcase, A(\vec{x})]\}]$$

Note that we have not yet defined the regression operator $Regr[vcase, A(\vec{x})]$. As it turns out, the implementation of this operator is specific to a given FOMDP language and SDP implementation. We simply remark that the regression of a formula ψ through an action $A(\vec{x})$ is a formula ψ' that holds prior to $A(\vec{x})$ being performed iff ψ holds after $A(\vec{x})$. However, regression is a deterministic operator and thus FODTR takes the expectation of the regression over all possible outcomes of a stochastic action according to their respective probabilities.

It is important to note that the case statement resulting from FODTR contains free variables for the action parameters $\vec{x}$. That is, for any constant binding $\vec{c}$ of these action parameters such that $\vec{x} = \vec{c}$, the case statement specifies a well-defined logical description of the value that can be obtained by taking action $A(\vec{c})$ and following a policy so as to obtain the value given by *vcase* thereafter. However, what we really need for symbolic dynamic programming is a logical description of a Q-function that tells us the highest value that can be achieved for *any* action instantiation. This leads us to the following $qCase(A(\vec{x}))$ definition of a first-order Q-function that makes use of the previously defined max $\exists\vec{x}$ operator:

$$qCase^t(A(\vec{x})) = \max \exists\vec{x}.\, FODTR[vcase^{t-1}, A(\vec{x})] \quad (7)$$

Intuitively, $qCase^t(A(\vec{x}))$ is a logical description of the Q-function for action $A(\vec{x})$ indicating the best value that could be achieved by *any* instantiation of $A(\vec{x})$. And by using the case representation and action quantification in the max $\exists\vec{x}$ operation, FODTR effectively achieves *both* action and state abstraction.

At this point, we can regress the value function through a *single* action, but to complete the dynamic programming step, we need to know the maximum value that can be achieved by *any* action (e.g., in the BoxWorld FOMDP, our possible action choices are $unload(b,t,c)$, $load(b,t,c)$, and $drive(t,c_1,c_2)$). Fortunately, this turns out to be quite easy. Assuming we have m actions $\{A_1(\vec{x}_1),\ldots,A_m(\vec{x}_m)\}$, we can complete the SDP step in the following manner using the previously defined max operator:

$$vcase^t = \max_{a\in\{A_1(\vec{x}_1),\ldots,A_m(\vec{x}_m)\}} qCase^t(a) \quad (8)$$

While the details of SDP may seem very abstract at the moment, there are several examples for specific FOMDP languages that implement SDP as described earlier, for which we provide references next. Nonetheless, one should note that the SDP equations given here are exactly the "lifted" versions of the traditional dynamic programming solution to MDPs given previously in (4) and (5). The reader may verify — on a conceptual level — that applying SDP to the 0-stages-to-go value function (i.e., $case^0 = rCase$, given previously) yields the following 1- and 2-stages-to-go value functions in the BoxWorld domain (¬" indicating the conjunction of the negation of all higher value partitions):

$case^1 =$

$\exists b.BoxIn(b, paris)$	: 19.0
$\neg" \wedge \exists b,t.TruckIn(t, paris) \wedge BoxOn(b,t)$	: 9.0
$\neg"$	: 0.0

$case^2 =$

$\exists b.BoxIn(b, paris)$	: 27.1
$\neg" \wedge \exists b,t.TruckIn(t, paris) \wedge BoxOn(b,t)$	: 17.1
$\neg" \wedge \exists b,c,t.BoxOn(b,t) \wedge TruckIn(t,c)$	: 8.1
$\neg"$	: 0.0

After sufficient iterations of SDP, the t-stages-to-go value function converges, giving the optimal value function (and corresponding policy) from Fig. 2.

Applications

Variants of SDP have been successfully applied in decision-theoretic planning domains such as BlocksWorld, BoxWorld, ZenoWorld, Elevators, Drive, PitchCatch, and Schedule. The

first-order approximate linear programming (FOALP) system (Sanner & Boutilier, 2005) was runner-up at the probabilistic track of the 5th International Planning Competition (IPC-6). Related techniques have been used to solve path planning problems within robotics and instances of real-time strategy games, Tetris, and Digger.

Future Directions

The original *SDP* (Boutilier et al., 2001) approach is a value iteration algorithm for solving FOMDPs based on Reiter's situations calculus. Since then, a variety of exact algorithms have been introduced to solve MDPs with relational (RMDP) and first-order (FOMDP) structure (We use the term *relational MDP* to refer to models that allow implicit existential quantification, and *FOMDP* for those with explicit existential and universal quantification.). *First-order value iteration (FOVIA)* (Hölldobler & Skvortsova, 2004; Karabaev & Skvortsova, 2005) and the *relational Bellman algorithm (ReBel)* (Kersting, van Otterlo, & de Raedt, 2004) are value iteration algorithms for solving RMDPs. In addition, *first-order decision diagrams (FODDs)* have been introduced to compactly represent case statements and to permit efficient application of SDP operations to solve RMDPs via value iteration (Wang, Joshi, & Khardon, 2007) and policy iteration (Wang & Khardon, 2007). All of these algorithms have some form of guarantee on convergence to the (ϵ-)optimal value function or policy. The expressiveness of FOMDPs has been extended to support indefinitely factored reward and transition functions in FOMDPs (Sanner & Boutilier, 2007).

A class of linear-value approximation algorithms have been introduced to approximate the value function as a linear combination of weighted basis functions. FOALP (Sanner & Boutilier, 2005) directly approximates the FOMDP value function using a linear program. *First-order approximate policy iteration (FOAPI)* (Sanner & Boutilier, 2006) approximately solves for the FOMDP value function by iterating between policy and value updates in a policy-iteration style algorithm. Somewhat weak error bounds can be derived for a value function approximated via FOALP (Sanner & Boutilier, 2005) while generally stronger bounds can be derived from the FOAPI solution (Sanner & Boutilier, 2006).

Finally, there are a number of heuristic solutions to FOMDPs and RMDPs. *Approximate policy iteration* (Fern, Yoon, & Givan, 2003) induces rule-based policies from sampled experience in small-domain instantiations of RMDPs and generalizes these policies to larger domains. In a similar vein, *inductive policy selection using first-order regression* (Gretton & Thiebaux, 2004) uses the action regression operator in the situation calculus to provide the first-order hypothesis space for an inductive policy learning algorithm. *Approximate linear programming (for RMDPs)* (Guestrin, Koller, Gearhart, & Kanodia, 2003) is an approximation technique using linear program optimization to find a best-fit value function over a number of sampled RMDP domain instantiations.

Cross References

►Dynamic Programming
►Markov Decision Processes

Recommended Reading

Bellman, R. E. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *IJCAI-01* (pp.690–697) Seattle.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2*, 189–208.

Fern, A., Yoon, S., & Givan, R. (2003). Approximate policy iteration with a policy language bias. In *NIPS-2003*. Vancouver.

Gretton, C., & Thiebaux, S. (2004). Exploiting first-order regression in inductive policy selection. In *UAI-04*. (pp.217–225) Banff, Canada.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. (2003). Generalizing plans to new environments in relational MDPs. In *IJCAI-03*. Acapulco, Mexico.

Hölldobler, S., & Skvortsova, O. (2004). A logic-based approach to dynamic programming. In *AAAI-04 Workshop on Learning and Planning in MDPs* (pp.31–36). Menlo Park, CA.

Karabaev, E., & Skvortsova, O. (2005). A heuristic search algorithm for solving first-order MDPs. In *UAI-2005* (pp.292–299). Edinburgh, Scotland.

Kersting, K., van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. In *ICML-04*. New York ACM Press.

Kushmerick, N., Hanks, S., & Weld, D. (1995). An algorithm for probabilistic planning. *Artificial Intelligence, 76*, 239–286.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: Wiley.

Sanner, S., & Boutilier, C. (2005). Approximate linear programming for first-order MDPs. In *UAI-2005*. Edinburgh, Scotland.

Sanner, S., & Boutilier, C. (2006) Practical linear evaluation techniques for first-order MDPs. In *UAI-2006*. Boston.

Sanner, S., & Boutilier, C. (2007). Approximate solution techniques for factored first-order MDPs. In *ICAPS-07*. Providence, RI. pp. 288–295.

Wang, C., Joshi, S., & Khardon, R. (2007). First order decision diagrams for relational MDPs. In *IJCAI*. Hyderabad, India.

Wang, C., & Khardon, R. (2007). Policy iteration for relational MDPs. In *UAI*. Vancouver, Canada.

Symbolic Regression

►Equation Discovery

Symmetrization Lemma

Synonyms

Basic lemma

Definition

Given a distribution P over a sample space $\mathcal{Z}$, a finite sample $\mathbf{z} = (z_1, \ldots, z_n)$ drawn i.i.d. from P and a function $f : \mathcal{Z} \to \mathbb{R}$ we define the shorthand $\mathbb{E}_P f = \mathbb{E}_P[f(z)]$ and $\mathbb{E}_{\mathbf{z}} f = \frac{1}{n}\sum_{i=1}^{n} f(z_i)$ to denote the true and empirical average of f. The symmetrization lemma is an important result in the learning theory as it allows the true average $\mathbb{E}_P f$ found in ►generalization bounds to be replaced by a second empirical average $\mathbb{E}_{\mathbf{z}'} f$ taken over an independent *ghost sample* $\mathbf{z}' = (z'_1, \ldots z'_n)$ drawn i.i.d. from P. Specifically, the symmetrization lemma states that for any $\epsilon > 0$ whenever $n\epsilon^2 \geq 2$

$$P^n\left(\sup_{f\in\mathbb{F}} |\mathbb{E}_P f - \mathbb{E}_{\mathbf{z}} f| > \epsilon\right) \leq 2P^{2n}\left(\sup_{f\in\mathbb{F}} |\mathbb{E}_{\mathbf{z}'} f - \mathbb{E}_{\mathbf{z}} f| > \frac{\epsilon}{2}\right).$$

This means the typically difficult to analyze behavior of $\mathbb{E}_P f$ – which involves the entire sample space $\mathcal{Z}$ – can be replaced by the evaluation of functions from $\mathbb{F}$ over the points in $\mathbf{z}$ and $\mathbf{z}'$.

Synaptic E.Cacy

►Weight

T

Tagging

►POS Tagging

TAN

►Tree Augmented Naive Bayes

Taxicab Norm Distance

►Manhattan Distance

TD-Gammon

Definition

TD-Gammon is a world-champion strength backgammon program developed by Gerald Tesauro. Its development relied heavily on machine learning techniques, in particular on ►Temporal-Difference Learning. Contrary to successful game programs in domains such as chess, which can easily out-search their human opponents but still trail these ability of estimating the positional merits of the current board configuration, TD-Gammon was able to excel in backgammon for the same reasons that humans play well: its grasp of the positional strengths and weaknesses was excellent. In 1998, it lost a 100-game competition against the world champion with only 8 points. Its sometimes unconventional but very solid evaluation of certain opening strategies had a strong impact on the backgammon community and was soon adapted by professional players.

Description of the Learning System

TD-Gammon is a conventional game-playing program that uses very shallow search (the first versions only searched one ply) for determining its move. Candidate moves are evaluated by a ►Neural Network, which is trained by TD(λ), a well-known algorithm for Temporal-Difference Learning (Tesauro, 1992). This evaluation function is trained on millions of games that the program played against itself. At the end of each game, a reinforcement signal that indicates whether the game has been lost or won is passed through all moves of the game. TD-Gammon developed useful concepts in the hidden layer of its network. Tesauro (1992) shows examples for two hidden units of TD-Gammon that he interpreted as a race-oriented feature detector and an attack-oriented feature detector.

TD-Gammon clearly surpassed its predecessors, in particular the Computer Olympiad champion Neurogammon, which was trained with ►Preference Learning (Tesauro, 1989). In fact, early versions of TD-Gammon, which only used the raw board information as features, already learned to play as well as Neurogammon, which used a sophisticated set of features. Adding more sophisticated features to the input representation further improved TD-Gammon's playing strength. Over time, TD-Gammon not only that increase the number of training games that it played against itself, but Tesauro also increased the search depth and changed the network architecture, so that TD-Gammon eventually reached world-championship strength (Tesauro, 1995).

Cross References

►Machine Learning and Game Playing

Recommended Reading

Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. In D. Touretzky (Ed.), *Proceedings of the advances in neural information processing systems 1 (NIPS-88)* (pp. 99–106). San Francisco: Morgan Kaufmann.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8*, 257–278. http://mlis.www.wkap.nl/mach/abstracts/absv8p257.htm.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM, 38*(3), 58–68. http://www.research.ibm.com/massdist/tdl.html.

TDIDT Strategy

►Divide-and-Conquer Learning

Temporal Credit Assignment

►Credit Assignment

Temporal Data

►Time Series

Temporal Difference Learning

William Uther
NICTA and the University of New South Wales

Definition

Temporal Difference Learning, also known as TD-Learning, is a method for computing the long term utility of a pattern of behavior from a series of intermediate rewards (Sutton, 1984, 1988; Sutton and Barto, 1998). It uses differences between successive utility estimates as a feedback signal for learning. The Temporal Differencing approach to model-free ►reinforcement learning was introduced by, and is often associated with, R.S. Sutton. It has ties to both the artificial intelligence and psychological theories of reinforcement learning as well as ►dynamic programming and operations research from economics (Bellman, 1957; Bertsekas & Tsitsiklis, 1996; Puterman, 1994; Samuel, 1959; Watkins, 1989).

While TD learning can be formalised using the theory of ►Markov Decision Processes, in many cases it has been used more as a heuristic technique and has achieved impressive results even in situations where the formal theory does not strictly apply, e.g., Tesauro's TD-Gammon (Tesauro, 1995) achieved world champion performance in the game of backgammon. These heuristic results often did not transfer to other domains, but over time the theory behind TD learning has expanded to cover large areas of reinforcement learning.

Formal Definitions

Consider an agent moving through a world in discrete time steps, $t_1, t_2, \ldots$. At each time step, t, the agent is informed of both the current state of the world, $s_t \in \mathcal{S}$, and its reward, or utility, for the previous time step, $r_{t-1} \in \mathbb{R}$.

As the expected long term utility of a pattern of behavior can change depending upon the state, the utility is a function of the state, $V : \mathcal{S} \to \mathbb{R}$. V is known as the *value function* or *state-value function*. The phrase "long term utility" can be formalized in multiple ways.

Undiscounted sum of reward:

The simplest definition is that long term reward is the sum of all future rewards.

$$\begin{aligned} V(s_t) &= r_t + r_{t+1} + r_{t+2} + \cdots \\ &= \sum_{\delta=0}^{\infty} r_{t+\delta} \end{aligned}$$

Unfortunately, the undiscounted sum of reward is only well defined if this sum converges. Convergence is usually achieved by the addition of a constraint that the agent's experience terminates at some, finite, point in time and all rewards after that point are zero.

Discounted sum of reward:

The *discounted* utility measure discounts rewards exponentially into the future.

$$\begin{aligned} V(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \quad \gamma \in [0,1] \\ &= \sum_{\delta=0}^{\infty} \gamma^{\delta} r_{t+\delta} \end{aligned}$$

Note that when $\gamma = 1$ the discounted and undiscounted regimes are identical. When $\gamma < 1$, the discounted reward scheme does not require that the agent experience terminates at some finite time for convergence. The *discount factor* γ can be interpreted as an inflation rate, a probability of failure for each time step, or simply as a mathematical trick to achieve convergence.

Average reward:

Rather than consider a sum of rewards, the *average reward* measure of utility estimates both the expected reward per future time step, also known as the *gain*, and the current difference from that long-term average, or *bias*.

$$G(s_t) = \lim_{n\to\infty} \frac{1}{n} \sum_{\delta=0}^{n} r_{t+\delta}$$
$$B(s_t) = \sum_{\delta=0}^{\infty} [r_{t+\delta} - G(s_{t+\delta})]$$

A system where any state has a nonzero probability of being reached from any other state is known as an ergodic system. For such a system the gain, $G(s)$, will have the same value for all states and the bias, $B(s)$, serves a similar purpose to $V(s)$ above in indicating the relative worth of different states. While average reward has a theoretical advantage in that there is no discount factor to choose, historically average reward has been considered more complex to use than the discounted reward regimes and so has been less used in practice. There is a strong theoretical relationship between average reward and discounted reward in the limit as the discount factor approaches one.

Here we focus on discounted reward.

Estimating Discounted Sum of Reward The temporal differencing estimation procedure is based on recursive reformulation of the above definitions. For the discounted case:

$$\begin{aligned} V(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots \\ &= r_t + \gamma[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots] \\ &= r_t + \gamma V(s_{t+1}) \end{aligned}$$

From the recursive formulation we can see that the long term utility for one time step is closely related to the long term utility at the next time step. If there is already an estimate of the long term utility at s_t, $V(s_t)$, then we could calculate a change in that value given a new trajectory as follows:

$$\Delta_t = [r_t + \gamma V(s_{t+1})] - V(s_t)$$

If we are dealing with a stochastic system, then we may not want to update $V(s_t)$ to the new value in one jump, but rather only move part way toward the new value:

$$\Delta_t = \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

where α is a learning rate between 0 and 1. As an assignment, this update can be written in a number of equivalent ways, the two most common being:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t)) \quad \text{or,}$$
$$V(s_t) \leftarrow (1-\alpha)V(s_t) + \alpha(r_t + \gamma V(s_{t+1}))$$

This *update*, *error*, *learning* or *delta* rule is the core of temporal difference learning. It is from this formulation, which computes a delta based on the difference in estimated long term utility of the world at two consecutive time steps, that we get the term temporal differencing.

Having derived this update rule, we can now apply it to finding the long term utility of a particular agent. In the simplest case we will assume that there are a finite number of Markov states of the world, and that these can be reliably detected by the agent at run time. We will store the function V as an array of real numbers, with one number for each world state.

After each time step, t, we will use the knowledge of the previous state, s_t, the instantaneous reward for the time step, r_t, and the resulting state, s_{t+1}, to update the value of the previous state, $V(s_t)$, using the delta rule above:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

Eligibility Traces and TD(λ)

Basic temporal differencing as represented above can be quite slow to converge in many situations. Consider, for example, a simple corridor with a single reward at the end, and an agent that walks down the corridor. Assume that the value function was initialized to a uniform zero value. On each walk down the corridor, useful information is only pushed one step back toward the start of the corridor.

Eligibility traces try to alleviate this problem by pushing information further back along the trajectory of the agent with each update to V. An algorithm incorporating eligibility traces can be seen as a mixture of "pure" TD, as described above, and ▶Monte-Carlo estimation of the long term utility. In particular, the λ parameter to the TD(λ) family of algorithms

specifies where in the range from pure TD, when $\lambda = 0$, to pure Monte-Carlo, when $\lambda = 1$, a particular algorithm falls.

Eligibility traces are implemented by keeping a second function of the state space, $\epsilon : \mathcal{S} \to \mathbb{R}$. The ϵ function represents how much an experience now should affect the value of a state the agent has previously passed through. When the agent performs an update, the values of all states are changed according to their eligibility.

The standard definition of the eligibility of a particular state uses an exponential decay over time, but this is not a strict requirement and other definitions of eligibility could be used. In addition, each time a state is visited, its eligibility increases. Formally, on each time step,

$$\forall_{s \in \mathcal{S}}\ \epsilon(s) \leftarrow \gamma\lambda\epsilon(s) \qquad \text{and then,}$$
$$\epsilon(s_t) \leftarrow \epsilon(s_t) + 1$$

This eligibility is used to update all state values by first calculating the delta for the current state as above, but then applying it to all states according to the eligibility values:

$$\Delta_t = \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$
$$\forall_{s \in \mathcal{S}}\ V(s) \leftarrow V(s) + \Delta_t \epsilon(s)$$

Convergence

TD value function estimation has been shown to converge under many conditions, but there are also well known examples where it does not converge at all, or does not converge to the correct long term reward (Tsitsiklis & Van Roy, 1997).

In particular, temporal differencing has been shown to converge to the correct value of the long term discounted reward if,

- The world is finite.
- The world state representation is Markovian.
- The rewards are bounded.
- The representation of the V function has no constraints (e.g., a tabular representation with an entry for each state).
- The learning rate, α, is reduced according to the Robbins-Monro conditions: $\sum_{t=0}^{\infty} \alpha_t = \infty$, and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$.

Much of the further work in TD learning since its invention has been in finding algorithms that provably converge in more general cases.

These convergence results require that a Markovian representation of state be available to the agent. There has been research into how to acquire such a representation from a sequence of observations. The approach of the Temporal Differencing community has been to use TD-Networks (Sutton & Tanner, 2004).

Control of Systems

Temporal Difference Learning is used to estimate the long term reward of a pattern of behavior. This estimation of utility can then be used to improve that behavior, allowing TD to help solve a reinforcement learning problem. There are two common ways to achieve this: An *Actor-Critic* setup uses value function estimation as one component of a larger system, and the *Q-learning* and *SARSA* techniques can be viewed as slight modifications of the TD method which allow the extraction of control information more directly from the value function.

First we will formalise the concept of a pattern of behavior. In the preceding text it was left deliberately vague as TD can be applied to multiple definitions. Here we will focus on discrete action spaces.

Assume there is a set of allowed actions for the agent, $\mathcal{A}$. We define a *Markov policy* as a function from world states to actions, $\pi : \mathcal{S} \to \mathcal{A}$. We also define a *stochastic* or *mixed* Markov policy as a function from world states to probability distributions over actions, $\pi : \mathcal{S} \to \mathcal{A} \to [0,1]$. The goal of the control algorithm is to find an optimal policy: a policy that maximises long term reward in each state. (When function approximation is used (see section "Approximation"), this definition of an optimal policy no longer suffices. One can then either move to average reward if the system is ergodic, or give a, possibly implicit, weighting function specifying the relative importance of different states.)

Actor-Critic Control Systems Actor-Critic control is closely related to *mixed policy iteration* from Markov Decision Process theory. There are two parts to an actor-critic system; the *actor* holds the current policy for the agent, and the *critic* evaluates the actor and suggests improvements to the current policy.

There are a number of approaches that fall under this model. One early approach stores a preference value for each world state and action pair, $p : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The actor then uses a stochastic policy based on the Gibbs softmax function applied to the preferences:

$$\pi(s,a) = \frac{e^{p(s,a)}}{\sum_{x \in \mathcal{A}} e^{p(s,x)}}$$

The critic then uses TD to estimate the long term utility of the current policy, and also uses the TD update to change the preference values. When the agent is positively surprised it increases the preference for an action, when negatively surprised it decreases the preference for an action. The size of the increase or decrease is modulated by a parameter, β:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta \Delta_t$$

Convergence of this algorithm to an optimal policy is not guaranteed.

A second approach requires the agent to have an accurate model of its environment. In this approach the critic uses TD to learn a value function for the current behavior. The actor uses model based forward search to choose an action likely to lead to a state with a high expected long term utility. This approach is common in two player, zero sum, alternating move games such as Chess or Checkers where the forward search is a deterministic game tree search.

More modern approaches which guarantee convergence are related to *policy gradient* approaches to reinforcement learning (Castro & Meir, 2010). These store a stochastic policy in addition to the value function, and then use the TD updates to estimate the gradient of the long term utility with respect to that policy. This allows the critic to adjust the policy in the direction of the negative gradient with respect to long term value, and thus improve the policy.

Other Value Functions The second class of approaches to using TD for control relies upon extending the value function to estimate the value of multiple actions. Instead of V we use a *state-action value function*, $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The update rule for this function is minimally modified from the TD update defined for V above.

Once these state-action value functions have been estimated, a policy can be selected by choosing for each state the action that maximizes the state-action value function, and then adding some exploration.

In order for this extended value function to be learned, the agent must explore each action in each state infinitely often. Traditionally this has been assured by making the agent select random actions occasionally, even when the agent believes that action would be sub-optimal. In general the choice of when to explore using a sub-optimal action, the *exploration/exploitation trade-off*, is difficult to optimize. More recent approaches to optimizing the exploration/exploitation trade-off in reinforcement learning estimate the variance of the value function to decide where they need to explore (Auer & Ortner, 2007).

The requirement for exploration leads to two different value functions that could be estimated. The agent could estimate the value function of the pattern of behavior currently being executed, which includes the exploration. Or, the agent could estimate the value function of the current best policy, excluding the exploration currently in use. These are referred to as *on-policy* and *off-policy* methods respectively.

Q-Learning is an off-policy update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

$$\text{where} \qquad V(s_{t+1}) = \max_{a \in \mathcal{A}} Q(s_{t+1}, a)$$

SARSA is an on-policy update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Then for both:

$$\pi(s) = argmax_{a \in \mathcal{A}} Q(s,a)$$

and some exploration.

As can be seen above, the update rules for SARSA and Q-learning are very similar – they only differ in the value used for the resulting state. Q-learning uses the value of the best action, whereas SARSA uses the value of the action that will actually be chosen.

Q-Learning converges to the best policy to use once you have converged and can stop exploring. SARSA converges to the best policy to use if you want to keep exploring as you follow the policy (Lagoudakis & Parr, 2003).

Approximation

A major problem with many state based algorithms, including TD learning, is the so-called ▶curse of dimensionality. In a factored state representation, the number of states increases exponentially with the number of factors. This explosion of states produces two problems: it can be difficult to store a function over the state space, and even if the function can be stored, so much data is required to learn the function that learning is impractical.

The standard response to the curse of dimensionality is to apply function approximation to any function of state. This directly attacks the representation size, and also allows information from one state to affect another "similar" state allowing generalisation and learning.

While the addition of function approximation can significantly speed up learning, it also causes difficulty with convergence. Some types of function approximation will stop TD from converging at all. The resulting algorithms can either oscillate forever or approach infinite values. Other forms of approximation cause TD to converge to a estimate of long term reward which is only weakly related to the true long term reward (Baird, 1995; Boyan & Moore, 1995; Gordon, 1995).

Most styles of function approximation used in conjunction with TD learning are parameterized, and the output is differentiable with respect to those parameters. Formally we have $V : \Theta \to \mathcal{S} \to \mathbb{R}$, where Θ is the space of possible parameter vectors, so that $V_\theta(s)$ is the value of V at state s with parameter vector θ, and $\nabla V_\theta(s)$ is the gradient of V with respect to θ at s. The TD update then becomes:

$$\Delta_t = \alpha \left(r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \right)$$
$$\theta \leftarrow \theta + \Delta_t \nabla V_\theta(s_t)$$

We describe three styles of approximation: state abstraction, linear approximation, and smooth general approximators (e.g., neural networks).

State abstraction refers to grouping states together and thereafter using the groups, or *abstract states*, instead of individual states. This can significantly reduce the amount of storage required for the value function as only values for abstract states need to be stored. It also preserves convergence results. A slightly more advanced form of state abstraction is the tile coding or CMAC (Albus, 1981). In this type of function approximation, the state representation is assumed to be factored, i.e., each state is represented by a vector of values rather than a single scalar value. The CMAC represents the value function as the sum of separate value functions; one for each dimension of the state. Those individual dimensions can each have their own state abstraction. Again, TD has been shown to converge when used with a CMAC value function representation.

In general, any form of function approximation that forms a contraction mapping will converge when used with TD (see the entry on ▶Markov Decision Processes). Linear interpolation is a contraction mapping, and hence converges. Linear extrapolation is not a contraction mapping and care needs to be taken when using general linear functions with TD. It has been shown that general linear function approximation used with TD will converge, but only when complete trajectories are followed through the state space (Tsitsiklis & Van Roy, 1997).

It is not uncommon to use various types of back-propagation neural nets with TD, e.g., Tesauro's TD-gammon. More recently, TD algorithms have been proposed that converge for arbitrary differentiable function approximators (Maei et al., 2009; Papavassiliou and Russell, 1999). These use more complex update techniques than those shown above.

Related Differencing Systems

TD learning was originally developed for use in environments where accurate models were unavailable. It has a close relationship with the theory of Markov Decision Processes where an accurate model is assumed. Using the notation $V(s_t) \rightsquigarrow V(s_{t+1})$ for a TD-style update that moves the value at $V(s_t)$ closer to the value at $V(s_{t+1})$ (including any discounting and intermediate rewards), we can now consider many possible updates.

As noted above, one way of applying TD to control is to use forward search. Forward search can be implemented using dynamic programming, and the result is closely related to TD. Let state $c(s)$ be the best child of state s in the forward search. We can then consider an update, $V(s) \rightsquigarrow V(c(s))$. If we let $l(s)$ be the best leaf in the forward search, we could then consider an update $V(s) \rightsquigarrow V(l(s))$. Neither of these updates consider the world after an actual state transition, only simulated state transitions, and so neither is technically a TD update.

Some work has combined both simulated time steps and real time steps. The TD-Leaf learning algorithm for alternative move games uses the $V(l(s_t)) \rightsquigarrow V(l(s_{t+1}))$ update rule (Baxter et al., 1998).

An important issue to consider when using forward search is whether the state distribution where learning takes place is different to the state distribution where the value function is used. In particular, if updates only occur for states the agent chooses to visit, but the search is using estimates for states that the agent is not visiting, then TD may give poor results. To combat this, the TreeStrap(α-β) algorithm for alternating move games updates all nodes in the forward search tree to be closer to the bound information provided by their children (Veness et al., 2009).

Biological Links

There are strong relationships between TD learning and the Rescorla–Wagner model of Pavlovian conditioning. The Rescorla–Wagner model is one way to formalize the idea that learning occurs when the co-occurence of two events is surprising rather than every time a co-occurence is experienced. The Δ_t value calculated in the TD update can be viewed as a measure of surprise. These findings appear to have a neural substrate in that dopamine cells react to reward when it is unexpected and to the predictor when the reward is expected (Schultz et al., 1997; Sutton & Barto, 1990).

Cross References

►Curse of Dimensionality
►Markov Decision Processes
►Monte-Carlo Simulation
►Reinforcement Learning

Recommended Reading

Albus, J. S. (1981). *Brains, behavior, and robotics*. Peterborough: BYTE, ISBN: 0070009759.

Auer, P., & Ortner, R. (2007). Logarithmic online regret bounds for undiscounted reinforcement learning. *Neural and Information Processing Systems (NIPS)*.

Baird, L. C. (1995). Residual algorithms: reinforcement learning with function approximation. In A. Prieditis & S. Russell (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference (ICML95)* (pp. 30–37). San Mateo: Morgan Kaufmann.

Baxter, J., Tridgell, A., & Weaver, L. (1998). KnightCap: a chess program that learns by combining TD(lambda) with game-tree search. In J. W. Shavlik (Ed.), *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)* (pp. 28–36). San Francisco: Morgan Kaufmann.

Bellman, R. E. (1957). *Dynamic programming*. Princeton: Princeton University Press.

Bertsekas, D. P., & Tsitsiklis, J. (1996). Neuro-dynamic programming. Belmont: Athena Scientific.

Boyan, J. A., & Moore, A. W. (1995). Generalization in reinforcement learning: safely approximating the value function. In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), *Advances in neural information processing systems* (Vol. 7). Cambridge: MIT Press.

Di Castro, D., & Meir, R. (2010). A convergent online single time scale actor critic algorithm. *Journal of Machine Learning Research, 11*, 367–410. http://jmlr.csail.mit.edu/papers/v11/dicastro10a.html

Gordon, G. F. (1995). *Stable function approximation in dynamic programming* (Technical report CMU-CS-95-103). School of Computer Science, Carnegie Mellon University.

Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research 4*, 1107–1149. http://www.cs.duke.edu/~parr/jmlr03.pdf

Maei, H. R. et al. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. *Neural and Information Processing Systems (NIPS)*, pp. 1204–1212. http://books.nips.cc/papers/files/nips22/NIPS2009_1121.pdf

Mahadevan, S. (1996). Average reward reinforcement learning: foundations, algorithms, and empirical results. *Machine Learning, 22*, 159–195, doi: 10.1023/A:1018064306595.

Papavassiliou, V. A., & Russell, S. (1999). Convergence of reinforcement learning with general function approximators. *International Joint Conference on Artificial Intelligence*, Stockholm.

Puterman, M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming. Wiley series in probability and mathematical statistics. Applied probability and statistics section*. New York: Wiley.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development, 3*(3), 210–229.

Schultz, W., Dayan, P., & Read Montague, P. (1997). A neural substrate of prediction and reward. *Science, 275*(5306), 1593–1599, doi: 10.1126/science.275.5306.1593.

Sutton, R., & Tanner, B. (2004). Temporal difference networks. *Neural and Information Processing Systems (NIPS)*.

Sutton, R. S. (1984). Temporal credit assignment in reinforcement learning. Ph.D. thesis, University of Massachusetts, Amherst.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine learning, 3*, 9–44, doi: 10.1007/BF00115009.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: an introduction*. Cambridge: MIT Press.

Sutton, R. S., & Barto, A. G. (1990). Time-derivative models of Pavlovian reinforcement. In M. Gabriel & J. Moore (Eds.), *Learning and computational neuroscience: foundations of adaptive networks* (pp. 497–537). Cambridge: MIT Press.

Tesauro, G. (1995). Temporal difference learning and TD-gammon. *Communications of the ACM, 38*(3), 58–67.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control, 42*(5), 674–690.

Veness, J., et al. (2009). Bootstrapping from game tree search. *Neural and Information Processing Systems (NIPS)*.
Watkins, C. J. C. H. (1989). *Learning with delayed rewards*. Ph.D. thesis, Cambridge University Psychology Department, Cambridge.

Test Data

Synonyms
Evaluation data; Test instances

Definition
Test data are data to which a ►model is applied for the purposes of ►evaluation. When ►holdout evaluation is performed, test data are also called *out-of-sample data*, *holdout data*, or the *holdout set*.

Cross References
►Test Set

Test Instances

►Test Data

Test Set

Synonyms
Evaluation data; Evaluation set; Test data

Definition
A test set is a ►data set containing data that are used for ►evaluation by a ►learning system. Where the ►training set and the test set contain disjoint sets of data, the test set is known as a ►holdout set.

Cross References
►Data Set

Test Time

A learning algorithm is typically applied at two distinct times. Test time refers to the time when an algorithm is applying a learned model to make predictions. ►Training time refers to the time when an algorithm is learning a model from ►training data. ►Lazy learning usually blurs the distinction between these two times, deferring most learning until test time.

Test-Based Coevolution

Synonyms
Competitive coevolution

Definition
A coevolutionary system constructed to simultaneously develop solutions to a problem and challenging tests for candidate solutions. Here, individuals represent complete solutions or their tests. Though not precisely the same as *competitive coevolution*, there is a significant overlap.

Text Clustering

►Document Clustering

Text Learning

►Text Mining

Text Mining

Dunja Mladenić
Jožef Stefan Insitute, Ljubljana, Slovenia

Synonyms
Analysis of text; Data mining on text; Text learning

Definition
The term *text mining* is used analogous to ►data mining when the data is text. As there are some data specificities when handling text compared to handling data from databases, text mining has a number of specific methods and approaches. Some of these are extensions of data mining and machine learning methods, while other are rather text-specific. Text mining approaches

combine methods from several related fields, including machine learning, data mining, ►information retrieval, ►natural language processing, ►statistical learning, and the Semantic Web. Basic text mining approaches are also extended to enable handling different natural languages (►cross-lingual text mining) and are combined with methods for handling different data types, such as links and graphs (►link mining and link discovery, ►graph mining), images and video (multimedia mining).

Cross References

►Cross-Lingual Text Mining
►Feature Construction In Text Mining
►Feature Selection In Text Mining
►Semi-Supervised Text Processing
►Text Mining For Advertising
►Text Mining For News and Blogs Analysis
►Text Mining for the Semantic Web
►Text Mining For Spam Filtering
►Text Visualization

Text Mining for Advertising

Massimiliano Ciaramita
Yahoo! Research Barcelona,
Barcelona, Spain

Synonyms

Content match; Contextual advertising; Sponsored search; Web advertising

Definition

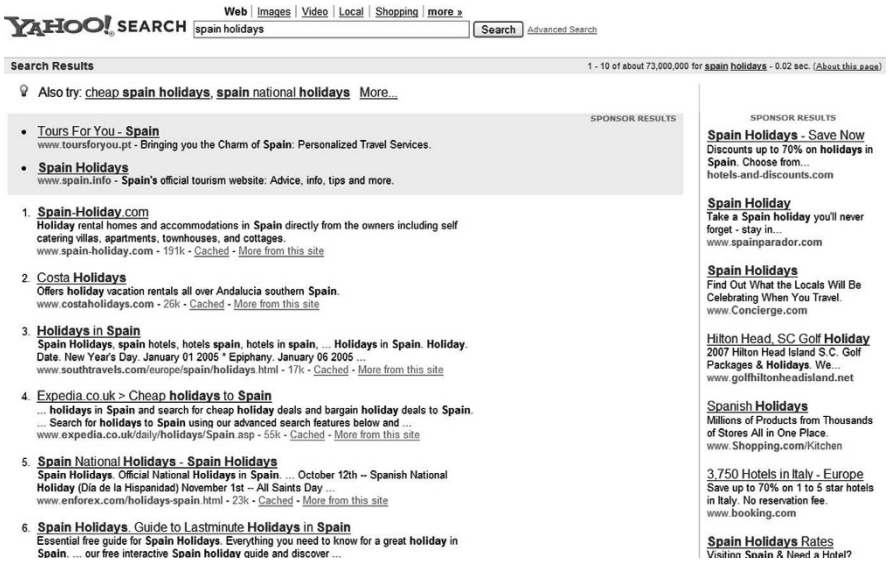
Text mining for advertising is an area of investigation and application of text mining and machine learning methods to problems such as Web advertising; e.g., automatically selecting the most appropriate ads with respect to a Web page, or query submitted to a search engine. Formally, the task can be framed as a ranking or matching problem where the unit of retrieval, rather than a Web page, is an advertisement. Most of the time ads have simple and homogeneous predefined textual structures, however, formats can vary and include audio and visual information. Advertising is a challenging problem due to several factors such as the economic nature of the transactions involved, engineering issues concerning scalability, and the inherent complexity of modeling the linguistic and multimedia content of advertisements.

Motivation and Background

The role of advertising in supporting and shaping the development of the Web has substantially increased over the past years. According to the Interactive Advertising Bureau (IAB), Internet advertising revenues in the USA totaled almost $8 billion in the first 6 months of 2006, a 36.7% increase over the same period in 2005, the last in a series of consecutive growths. Search, i.e., ads placed by Internet companies in Web pages or in response to specific queries, is the largest source of revenue, accounting for 40% of total revenue (Internet Advertising Bureau, 2006). The most important categories of Web advertising are *keyword match*, also known as *sponsored search* or *paid listing*, which places ads in the search results for specific queries (see Fain & Pedersen, 2006 for a brief history of sponsored search), and *content match*, also called *content-targeted advertising* or *contextual advertising*, which places ads in Web pages based on the page content. Figure 1 shows an example of sponsored search and ads are listed on the right side of the page.

Currently, most of the focus in Web advertising involves sponsored search, because matching based on keywords is a well-understood problem. Content match has greater potential for content providers, publishers, and advertisers, because users spend most of their time on the Web on content pages as opposed to search engine result pages. However, content match is a harder problem than sponsored search. Matching ads with query terms is to a certain degree straightforward, because advertisers themselves choose the keywords that characterize their ads that are matched against keywords chosen by users while searching. In contextual advertising, matching is determined automatically by the page content, which complicates the task considerably.

Advertising touches challenging problems concerning how ads should be analyzed, and how the accurately and efficiently systems select the best ads. This area of research is developing rapidly in information retrieval.



Text Mining for Advertising. Figure 1. Ads ranked next to a search results page for the query "Spain holidays"

How best to model the structure and components of ads, and the interaction between the ads and the contexts in that they appear are open problems. Information retrieval systems are designed to capture relevance, which is a basic concept in advertising as well. Elements of an ad such as text and images tend to be mutually relevant, and often (in print media for example) ads are placed in contexts that match a certain product at a topical level; e.g., an ad for sneakers placed on a sport news page. However, topical relevance is only one the basic parameters which determine a successful advertisement placement. For example, an ad for sneakers might be appropriate and effective on a page comparing MP3 players, because they may share a target audience (e.g., joggers) although they arguably refer to different topics, and it is possible they share no common vocabulary. Conversely, there may be ads that are topically similar to a Web page, but cannot be placed there because they are inappropriate. An example might be placing ads for a product in the page of a competitor.

The language of advertising is rich and sophisticated and can rely considerably on complex inferential processes. A picture of a sunset in an ad for life insurance carries a different implication than a picture of a sunset in an ad for beer. Layout and visual content are designed to elicit inferences, possibly hinging on cultural elements; e.g., the age, appearance, and gender of people in an ad affect its meaning. Adequate automatic modeling will likely involve, to a substantial degree, understanding the language of advertisement and the inferential processes involved (Vestergaard & Schroeder, 1985). Today this seems beyond what traditional information retrieval systems are designed to cope with. In addition, the global context can be captured only partially by modeling the text alone. As the Web evolves into an immense infrastructure for social interaction and multimedia information sharing the need for modeling structured "content" becomes more and more crucial. This applies to information retrieval and specifically to advertising. For this reason, the problem of content match is of particular interest and opens new problems and opportunities for interdisciplinary research.

Today, contextual advertising, the most interesting sub-task from a mining perspective, consists mostly in selecting ads from a pool to match the textual content of a particular Web page. Ads provide a limited amount of text: typically a few keywords, a title, and brief description. The ad-placing system needs to identify relevant ads, from huge ad inventories, quickly and efficiently based on this very limited amount of information. Current approaches have focused on augmenting the representation of the page to increase the chance of a match (Ribeiro-Neto, Cristo, Golgher, and de Moura, 2005), or by using machine learning to find complex ranking functions (Lacerda et al., 2006), or

by reducing the problem of content match to that of sponsored search by extracting keywords from the Web page (Yih et al., 2006). All these approaches are based on methods that quantify the similarity between the ad and the target page on the basis of traditional information retrieval notions such as cosine similarity and *tf.idf* features. The relevance of an ad for a page depends on the number of overlapping words, weighted individually and independently as a function of their individual distributional properties in the collection of documents or ads.

Structure of Learning Problem

The typical elements of an advertisement are a set of *keywords*, a *title*, a *textual description* and a hyperlink pointing to page, the *landing page*, relative to a product or service, etc. In addition, an ad has an *advertiser id* and can be part of a *campaign*, i.e., a subset of all the ads with same advertiser id. This latter information can be used, for example, to impose constraints on the number of ads to display relative to the campaign or advertiser. While this is possibly the most common layout, it is important to realize that ads structure can vary significantly and include relevant audio and visual content.

In general, the learning problem for an ad-placing system can be formalized as a ranking task. Let $\mathcal{A}$ be a set of ads, $\mathcal{P}$ the set of possible pages, and $\mathcal{Q}$ the set of possible queries. In keyword match, the goal is to find a function $F : \mathcal{A} \times \mathcal{Q} \to \mathbb{R}$; e.g., a function that counts the number of individual common terms or n-grams of such terms. In content match, the objective is to find a function $F : \mathcal{A} \times \mathcal{P} \to \mathbb{R}$. The keyword match problem is to a certain extent straightforward and amounts to matching small set of terms defined manually by both the user and the advertiser. The content match task shares with the former task the peculiarities of the units of retrieval (the ads), but introduces new and interesting issues for text mining and learning because of the more complex conditioning environment, the Web page content, which needs to modeled automatically.

In general terms, an ad can be represented as a feature vector $\mathbf{x} = \Phi(a_i, p_j)$ such that $a_i \in \mathcal{A}$, $p_j \in \mathcal{P}$, and given a d-dimensional feature space $\mathcal{X} \subset \mathbb{R}^d$, $\Phi : \mathcal{A} \times \mathcal{P} \to \mathcal{X}$. In the traditional machine learning setting, one introduces a weight vector $\alpha \in \mathbb{R}^d$ which quantifies each feature's contribution individually. The vector's weights can be learned from manually edited rankings (Lacerda et al., 2006; Ribeiro-Neto et al., 2005) or from click-through data as in search results optimization (Joachims, 2002). In the case of a linear classifier the score of an ad-target page pair $\mathbf{x}_i$ would be:

$$F(\mathbf{x}; \alpha) = \sum_{s=1}^{d} \alpha_s x_s. \quad (1)$$

Several methods can be used to learn similar or related models such as perceptron, SVM, boosting, etc. Constraints on the number of advertisers or campaigns could be easily implemented as post-ranking filters on the top of the ranked list of ads or included in a suitable objective function.

A basic model for ranking ads can be defined in the vector space model for information retrieval, using a ranking function based on cosine similarity, where ads and target pages are represented as vectors of terms weighted by fixed schemes such as *tf.idf*. If only one feature is used, the cosine based on *tf.idf* between the ad and the page, a standard vector space model baseline is obtained, which is at the base of the ad-placing ranking functions variants proposed by (Ribeiro-Neto et al., 2005) Recent work has shown that machine learning-based models are considerably more accurate than such baselines. However, as in document retrieval, simple feature maps which include mostly coarse-grained statistical properties of the ad-page pairs, such as *tfidf-based* cosine, are the most desirable for efficiency and bias reasons. Properties of the different components of the ad can be used and weighted in different ways, and soft or hard constraints introduced to model the presence of the ads keyword in the Web page. The design space for ad-place systems is vast and still little explored. All systems presented so far in the literature make use of manually annotated data for training and/or evaluating a model.

Structure of Learning Systems

Web advertising presents peculiar engineering and modeling challenges and has motivated research in different areas. Systems need to be able to deal in real time with huge volumes of data and transactions involving billions of ads, pages, and queries. Hence several engineering constraints need to be taken into account; efficiency and computational costs are crucial factors in the

choice of matching algorithms (The Yahoo! Research Team, 2006). Ad-placing systems might require new global architecture design; e.g., Attardi et al. (2004) proposed an architecture for information retrieval systems that need to handle large-scale targeted advertising based on an information filtering model. The ads that appear on Web pages or search results pages will ultimately be determined taking into account the expected revenues and the price of the ads. Modeling the microeconomics factors of such processes is a complex area of investigation in itself (Feng et al., 2005).

Another crucial issue is the evaluation of the effectiveness of the ad-placing systems. Studies have emphasized the impact of the quality of the matching on the success of the ad in terms of click-through rates (Gallagher et al., 2001; Sherman & Deighton, 2001). Although ►click-through rates (CTRs) provide a traditional measure of effectiveness, it has been found that ads can be effective even when they do not solicit any conscious response and that the effectiveness of the ad is mainly determined by the level of congruency between the ad and the context in which it appears (Yoo, 2006).

Keyword Extraction Approaches

Since the query-based ranking problem is better understood than contextual advertising, one way of approaching the latter would be to represent the content page as a set of keywords and then ranking the ads based on the keywords extracted from the content page. Carrasco et al. (2003) proposed clustering of bi-partite advertiser-keyword graphs for keyword suggestion and identifying groups of advertisers. Yih, Goodman, & Carvalho (2006) proposed a system for keyword extraction from content pages. The goal is to determine which keywords, or key phrases, are more relevant in a Web page. Yih et al. develop a supervised approach to this task from a corpus of pages where keywords have been manually identified. They show that a model learned with ►logistic regression outperforms traditional vector models based on fixed *tf.idf* weights. The most useful features to identify good keywords efficiently are, in this case, term frequency and document frequency of the candidate keywords, and particularly the frequency of the candidate keyword in a search engine query log. Other useful features include the similarity of the candidate with the page's URL and the length, in number of words, of the candidate keyword. In terms of feature representation thus, they propose a feature map $\Phi : \mathcal{A} \rightarrow \mathcal{Q}$, which represent a Web page as a set of keywords. The accuracy of the best learned system is 30.06%, in terms of the top predicted keyword being in the set of manually generated keywords for a page, against 13.01% of the simpler *tf.idf* based model. While this approach is simple to apply, it remains to be seen how accurate it is at identifying good ads for a page. It identifies potentially useful sources of information in automatically-generated keywords. An interesting related finding concerning keywords is that longer keywords, about four words long, lead to increased click-through rates (OneUpWeb, 2005).

The Vocabulary Impedance Problem

(Ribeiro-Neto et al., 2005) introduced an approach to content match which focuses on the vocabulary mismatch problem. They notice that there tends to be not enough overlap in the text of the ad and the target page to guarantee good accuracy; they call this the *vocabulary impedance problem*. To overcome this limitation they propose to generate an augmented representation of the target page by means of a Bayesian model previously applied to document retrieval (Ribeiro-Neto & Muntz, 1996). The expanded vector representation of the target page includes a significant number of additional words which can potentially match some of the terms in the ad. They find that such a model improves over a standard vector space model baseline, evaluated by means of 11-point average precision on a test bed of 100 Web pages, from 0.168 to 0.253. One possible shortcoming of such an approach is that generating the augmented representation involves crawling a significant number of additional related pages. It has also been argued (Yih et al., 2006) that this model complicates pricing of the ads because the keywords chosen by the advertisers might not be present in the content of the matching page.

Learning with Genetic Programming

Lacerda et al. (2006) proposed to use machine learning to find good ranking functions for contextual advertising. They use the same data-set described in Ribeiro-Neto et al. (2005), but use part of the data for training a model and part for evaluation purposes. They use a genetic programming algorithm to select a

ranking function which maximizes the average precision on the training data. The resulting ranking function is a nonlinear combination of simple components based on the frequency of ad terms in the target page, document frequencies, document length, and size of the collections. Thus, in terms of the feature representation defined earlier, they choose a feature map which extracts traditional features from the ad-page pair, but then apply then genetic programming methods to select complex nonlinear combinations of such features that maximize a fitness function based on average precision. Lacerda et al. (2006) find that the ranking functions selected in this way are considerably more accurate than the baseline proposed in Ribeiro-Neto et al. (2005); in particular, the best function selected by genetic programming achieves an average precision at position three of 0.508, against 0.314 of the baseline.

Semantic Approaches to Contextual Advertising

The most common approaches to contextual advertising are based on matching terms between the ad and the content page. Broder, Fontoura, Josifovski, and Riedel (2007) notice that this approach (which they call the "syntactic—" model), can be improved by adopting a matching model which additionally takes into account topical proximity; i.e., a "semantic" model. In their model the target page and the ad are classified with respect to a taxonomy of topics. The similarity of ad and target page estimated by means of the taxonomy provides an additional factor in the ads ranking function. The taxonomy, which has been manually built, contains approximately 6,000 nodes, where each node represents a set of queries. The concatenation of all queries at each node is used as a meta-document, ads and target pages are associated with a node in the taxonomy using a nearest neighbor classifier and *tf.idf* weighting. The ultimate score of an ad a_i for a page p is a weighted sum of the taxonomy similarity score and the similarity of a_i and p based on standard syntactic measures (vector cosine). On evaluation, Broder et al. (2007) report a 25% improvement for mid-range recalls of the syntactic-semantic model over the pure syntactic one.

Cross References

►Boosting
►Genetic Programming
►Information Retrieval
►Perceptron
►SVM
►TF–IDF
►Vector Space Model

Recommended Reading

Attardi, G., Esuli, A., & Simi, M. (2004). Best bets, thousands of queries in search of a client. In *Proceedings of the 13th international conference on World Wide Web, alternate track papers & posters*. New York: ACM Press.

Broder, A., Fontoura, M., Josifovski, V., & Riedel, L. (2007). A semantic approach to contextual advertising. In *Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval*. New York: ACM Press.

Carrasco, J. J., Fain, D., Lang, K., & Zhukov, L. (2003). Clustering of bipartite advertiser-keyword graph. In *Workshop on clustering large datasets, IEEE conference on data mining*. New York: IEEE Computer Society Press.

Fain, D., & Pedersen, J. (2006). Sponsored search: A brief history. In *Proceedings of the 2nd workshop on sponsored search auctions*. Web Publications.

Feng, J., Bhargava, H., & Pennock, D. (2005). Implementing sponsored search in Web search engines: Computational evaluation of alternative mechanisms. *Informs Journal on Computing* (forthcoming).

Gallagher, K., Foster, D., & Parsons, J. (2001). The medium is not the message: Advertising effectiveness and content evaluation in print and on the Web. *Journal of Advertising Research, 41*(4), 57–70.

Internet Advertising Bureau (IAB). (2006). *IAB Internet Advertising Revenue Report*. http://www.iab.net/resources/adrevenue/pdf/IAB_PwC%202006Q2.pdf

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the ACM conference on knowledge discovery and data mining (KDD)*. New York: ACM Press.

Lacerda, A., Cristo, M., Gonçalves, M. A., Fan, W., Ziviani, N., & Ribeiro-Neto, B. (2006). Learning to advertise. In *Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 549–556). New York: ACM Press.

OneUpWeb (2005). *How keyword length affects conversion rates*. http://www.oneupweb.com/landing/keywordstudy_landing.htm.

Ribeiro-Neto, B., Cristo, M., Golgher, P. B., & de Moura, E. S. (2005). Impedance coupling in content-targeted advertising. In *Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 496–503). New York: ACM Press.

Ribeiro-Neto, B., & Muntz, R. (1996). A belief network model for IR. In *Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 253–260). New York: ACM Press.

Sherman, L., & Deighton, J. (2001). Banner advertising: Measuring effectiveness and optimizing placement. *Journal of Interactive Marketing, 15*(2), 60–64.

The Yahoo! Research Team. (2006). Content, metadata, and behavioral information: Directions for Yahoo! Research. *IEEE Data Engineering Bulletin, 29*(4), 10–18.

Vestergaard, T., & Schroeder, T. (1985). *The language of advertising.* Oxford: Blackwell.

Yih, W., Goodman, J., & Carvalho, V. R. (2006). Finding advertising keywords on web pages. In *Proceedings of the 15th international conference on World Wide Web* (pp. 213–222). New York: ACM Press.

Yoo, C. Y. (2006). *Preattentive processing of web advertising.* PhD thesis, University of Texas at Austin.

Text Mining for News and Blogs Analysis

Bettina Berendt
Katholieke Universiteit Leuven, Heverlee, Belgium

Definition

News and blogs are two types of media that generate and offer informational resources. *News* is any information whose revelation is anticipated to have an intellectual or actionable impact on the recipient. The dominant type of news in text analysis is that pertaining to current events. Originally referring to print-based news from press agencies or end-user news providers (like individual newspapers or serials), it now increasingly refers to Web-based news in the online editions of the same providers or in online-only news media. The term is generally understood to denote only the reports in news media, not opinion or comment pieces. A *blog* is a (more or less) frequently updated publication on the Web, sorted in (usually reverse) chronological order of the constituent *blog posts*. The content may reflect any interest including personal, journalistic, or corporate. Blogs were originally called weblogs. To avoid confusion with web server log files that are also known by this term, the abbreviation "blog" was coined and is now commonly used.

News and blogs consist of textual and (in some cases) pictorial content, and, when Web-based, may contain additional content in any other format (e.g., video, audio) and hyperlinks. They are indexed by time and structured into smaller units: news media into articles, blogs into blog posts. In most news and blogs, textual content dominates. Therefore, text analysis is the most often applied form of knowledge discovery. This comprises tasks and methods from data/text mining, ►information retrieval, and related fields. In accordance with the increasing convergence of these fields, this article refers to all of them as ►*text mining*.

Motivation and Background

News and blogs are today's most common sources for learning about current events and also, in the case of blogs, for uttering opinions about current events. In addition, they may deal with topics of more long-term interest. Both reflect and form societies', groups' and individuals' views of the world, fast or even instantaneous with the events triggering the reporting.

However, there are differences between these two types of media regarding authoring, content, and form. News is generally authored by people with journalistic training who abide by journalistic standards regarding the style and language of reporting. Topics and ways of reporting are circumscribed by general societal consensus and the policies of the news provider. In contrast, everybody with Internet access can start a blog, and there are no restrictions on content and style (beyond the applicable types of censorship). Thus, blogs offer end users a wider range of topics and views on them. On the one hand, this implies that journalistic blogs, which correspond most closely to news, are only one type of blogs. Other frequent types are diary-like personal blogs, corporate blogs for public relations, and blogs focusing on special-interest topics. On the other hand, their comparative lack of restrictions has helped to establish blogs as an important alternative source of information, as a form of grassroots journalism that may give rise to a counterpublic. An example are the *warblogs* published during the early years of the Iraq War (2003+) by independent sources (often civilian individuals) both in the West and in the Middle East.

These application characteristics lead to various linguistic and computational challenges for text-mining analyses of news and blogs:

- *Indexing, taxonomic categorization, partial redundancy, and data streams:* News is indexed by time

and by source (news agency or provider). In a multisource corpus, many articles published at about the same time (in the same or in other languages) describe the same events. Over time, a story may develop in the articles. Such multiple reporting and temporal structures are also observed for popular topics in blogs.

- *Language and meaning:* News is written in clear, correct, "objective," and somewhat schematized language. Usually, the start of a news article summarizes the whole article (feeds are a partial analogue of this in blogs). Information from external sources such as press agencies is generally reused rather than referenced. In sum, news makes fewer assumptions about the reader's background and context knowledge than many other texts.
- *Nonstandard language and subjectivity:* The language in blogs ranges from high-quality "news-like" language through poor-quality, restricted-code language with many spelling and grammatical errors to creative, sometimes literary, language. Jargon is very common in blogs, and new linguistic developments are adopted far more quickly than could be reflected in external resources such as lexica. Many blog authors strive not for objectivity, but for subjectivity and emotionality.
- *Thematic diversity and new forms of categorization:* News are generally categorized by topic area ("politics," "business," etc.). In contrast, a blog author may choose to write about differing, arbitrary topics. When blogs are labeled, it is usually not with reference to a stable, taxonomic system, but with an arbitrary number of *tags*: free-form, often informal labels chosen by the user.
- *Social structure and its impact on content and meaning:* The content of a blog (post) is often not contained in the text alone. Rather, blog software supports "Social Web" behavior, and bloggers practice it: multiway communication rather than broadcasting, and semantics-inducing referencing of both content and people. Specifically, hyperlinks to other resources provide not only context but also content; "blogrolls" (hyperlinks to other blogs) supply context in terms of other blogs/bloggers recommended by the author; comments to blog posts are integral part of the communication that the post triggered. "Trackback" links, indicating hyperlinks set to the blog, may be automatically added by blogging software and thus, create a dynamic citation context.

Structure of the Learning System

Tasks

News and blogs may serve many different interests, for example, those of:

- End users who want to know what is happening in given universes of discourse, to follow developments within these areas, or to identify sources that are of long-term interest to them. These users differ by their preferences, their educational level, the purposes of their searches, and other factors. This calls for search engines, temporal analyses, topic identification, personalization, and related functionalities.
- Companies that want to learn about their target groups' views and opinions of their products and activities, detect trends and make predictions. Similar market research may be carried out by nonprofit organizations or politicians.
- People who use blogs to gain insights about specific blog author(s) as background knowledge for decisions on befriending, hiring, or insuring these individuals (see Nowson & Oberlander 2007) on the textual analysis of blogs for determining personality features).

The literature on news and blogs analysis reflects these and other possible uses. A number of standard tasks are emerging, furthered by the competitions at events such as the Topic Detection and Tracking (TDT) research program and workshops (http://www.itl.nist.gov/iad/mig/tests/tdt, Allan, 2002), Text Retrieval Conference (TREC, http://trec.nist.gov/, e.g., MacDonald, Ounis, & Soboroff, 2007), and Document Understanding/Text Analysis Conference (DUC/TAC, http://www.nist.gov/tac/). Other initiatives also provide and encourage the usage of standardized real-world ►datasets, but instigate research on novel questions by standardizing neither tasks nor ►algorithm evaluation. Prominent examples are the Reuters-21578 dataset, which is not only a collection of newswire articles but also the most classical dataset for text mining in general (http://kdd.ics.uci/edu/databases/reuters21578/reuters21578.html), and the blog datasets provided by

International Conference on Weblogs and Social Media (ICWSM, http://www.icwsm.org) and its precursors.

Tasks can be grouped by different criteria:

- *Use case and type of result:* description and prediction (supervised or unsupervised, may include topic identification, tracking, and/or novelty detection); search (ad hoc or filtering); recommendation (of blogs, blog posts, or tags); summarization
- *Higher-order characterization to be extracted:* topic; opinion
- *Time dimension:* nontemporal; temporal (stream mining); multiple streams (e.g., in different languages, see ▶cross-lingual text mining)
- *User adaptation:* none (no explicit mention of user issues and/or general audience); customizable; personalized

Since the beginning of news mining in the 1990s and of blog mining in the early 2000s, more complex combinations of these dimensions have been explored. Examples include (a) the TDT research program (1997/1998–2004) required an explicit focus on temporal analyses and called for topic description and prediction in a news stream; (b) "bursty" topics in a stream of blogs were used to predict peaks in a stream of online sales (Gruhl, Guha, Kumar, Novak, & Tomkins, 2005); (c) the role of opinion mining as a key question in blog analysis was manifested by the first TREC blog track in 2006 (see also MacDonald et al., 2007); it is now a standard task, also for analysing microblogs (Jansen, Zhang, Sobel, & Chowdury, 2009); (d) a recommendation method on a document stream based on tracking multiple topics over time, personalized to a user whose interests may change over time was developed in (Pon, Cardenas, Buttler, & Critchlow, 2007); (e) in (Lu & Zhai, 2008), opinions were summarized in a set of non–time-indexed texts, for a general audience; and (f) in (Subašić & Berendt, 2008), bursty topics in a news stream were summarized into graph patterns that can be interactively explored and customized.

Another important task is spam detection and blocking (Kolari, Java, Finin, Oates, & Joshi, 2006). While basically nonexistent in news mining (news are identified by their sources, which are "white-listed" and thus credible), spamming has become a severe problem in the blogosphere, ranging from comment spam via "flogs" (e.g., ghostwritten by a marketing department but pretending to be an end user), to "splogs" (artificially created blogs used to increase the visibility and search engine rankings of associated sites). (cf. ▶text mining for spam detection).

Solution Approaches

Solution approaches are based on general ▶data-mining methods and adapted to the conceptual specifics of news and blogs and their mining tasks (see list of tasks above). Methods include (▶document) classification and ▶clustering, latent-variable techniques such as (P)LSA or LDA (cf. ▶feature construction), ▶mixture models, ▶time series, and ▶stream mining methods. Named-entity recognition may be an important part or companion of topic detection (cf. ▶information extraction). Opinion mining often relies on word class identification and ▶part-of-speech tagging, and it generally employs lexica (e.g., of typical opinionated words and their positive or negative polarity). Data cleaning is similar to that of other Web documents; in particular, it requires the provision or learning of wrappers for removing markup elements.

In addition, many solution approaches exploit the specific formatting and/or linguistic features of blogs. For example, to improve the retrieval of blogs about a queried event, the format elements "timestamp" and "number of comments" can be treated as indicators of increased topical relevance and likelihood of being opinionated, respectively (Mishne, 2007). Structural elements of blogs such as length and representation in post title versus post body have been used for blog distillation (filtering out those blogs that are principally devoted to a topic rather than just mentioning it in passing) (Weerkamp, Balog, & de Rijke, 2008). Text-based statistical topic modeling can be enhanced by ▶regularizing it with the (e.g., social) network structure associated with blog data (Mei, Cai, Zhang, & Zhai, 2008) (cf. ▶link mining and link discovery). However, many blogs are not strongly hyperlinked – but tags also carry "Social Web" information: A combination of text clustering and tag analysis can serve to identify topics as well as the blogs that are on-topic and likely to retain this focus over time (Hayes, Avesani, & Bojars, 2007).

Due to blog writing style, standard indicators of relevance may not be applicable. For example, a term's

TF.IDF score, which is commonly used as a ►weight in a ►feature vector representing the document, assumes that important terms are mentioned (frequently) in the document and infrequently elsewhere. However, blogs often rely on implicit context – established by hyperlinks or by the history of the discussion. Solution proposals include the integration of the text from previous blog posts with the same tag (Hayes et al., 2007); in addition, terms from hyperlinked documents could be taken into account. In addition, while blogs may be more opinionated than news texts, their language may make it more difficult to extract topics and argumentation vis-à-vis that topic. Specifically, blogs often contain irony and other indirect uses of language for expressing appreciation or discontent. The "emotional charge" of a text has, therefore, been proposed as a better target for blog classification (Gamon et al., 2008).

Viewed in relation to each other, news and blogs pose some additional challenges for automated analysis and text mining. Several studies (e.g., Adamic & Glance 2005) address questions such as: How are blogs linked to news media (and possibly vice versa)? Do they form a coherent whole, "the blogosphere," or rather a loose connection of mutually unrelated, political, national, linguistic, etc., blogospheres? What are the topics investigated in blogs versus news? Are stories reported by news or blogs first, and how does the other side follow up reporting? In general, how do blogs and news refer to and contextualize each other (e.g., Gamon et al. 2008; Berendt & Trümper 2009; Leskovec, Backstrom, & Kleinberg 2009)?

Finally, text mining faces a further challenge: while news are always meant to be read, many blogs are not (e.g., because they are a personal journal) – even if they are accessible over the Web. This raises the question of whether text mining could or should become privacy-aware (cf. ►privacy-related aspects and techniques).

Recommended Reading

Adamic, L., & Glance, N. (2005). The political blogosphere and the 2004 U.S. election: Divided they blog. In E. Adar, N. Glance, & M. Hurst (Eds.), *Proceedings of the WWW 2005 2nd annual workshop on the weblogging ecosystem: Aggregation, analysis and dynamics.* Chiba, Japan. http://doi.acm.org/10.1145/1134271.1134277 New York, NY, 36–43.

Allan, J. (Ed.). (2002). *Topic detection and tracking: Event-based information organization*. Norwell, MA: Kluwer Academic Publishers.

Berendt, B., & Trümper, D. (2009). Semantics-based analysis and navigation of heterogeneous text corpora: The *Porpoise* news and blogs engine. In I.-H. Ting & H.-J. Wu (Eds.), *Web mining applications in e-commerce and e-services* Berlin: Springer.

Gamon, M., Basu, S., Belenko, D., Fisher, D., Hurst, M., & König, A. C. (2008). BLEWS: Using blogs to provide context for news articles. In E. Adar, M. Hurst, T. Finin, N. Glance, N. Nicolov, B. Tseng, & F. Salvetti (Eds.), *Proceedings of the second international conference on weblogs and social media (ICWSM'08).* Seattle, WA. Menlo Park, CA. http://www.aaai.org/Papers/ICWSM/2008/ICWSM08-015.pdf

Gruhl, D., Guha, R., Kumar, R., Novak, J., & Tomkins, A. (2005). The predictive power of online chatter. In R. Grossman, R. J. Bayardo, & K. P. Bennett (Eds.), *Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery and data mining.* Chicago, IL. New York, NY.

Hayes, C., Avesani, P., & Bojars, U. (2007). An analysis of bloggers, topics and tags for a blog recommender system. In B. Berendt, A. Hotho, D. Mladenič, & G. Semeraro (Eds.), *From web to social web: Discovering and deploying user and content profiles. LNAI 4737.* Berlin: Springer.

Jansen, B.J., Zhang, M., Sobel, K., & Chowdury, A. (2009). Twitter Power: Tweets as electronic word of mouth. *Journal of the American Society for Information Science and Technology, 60*(11): 2169–2188.

Kolari, P., Java, A., Finin, T., Oates, T., & Joshi, A. (2006). Detecting spam blogs: A machine learning approach. In *Proceedings of the 21st national conference on artificial intelligence.* Boston: AAAI.

Leskovec, J., Backstrom, L., & Kleinberg, J. (2009). Meme-tracking and the dynamics of the news cycle. In J.F. Elder IV, F. Fogelman-Soulié, P.A. Flach, & M.J. Zaki (Eds.), *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining,* Paris, France. New York, NY.

Lu, Y., & Zhai, C. (2008). Opinion integration through semi-supervised topic modeling. In J. Huai & R. Chen (Eds.), *Proceeding of the 17th international conference on world wide web (WWW'08).* Beijing, China. New York, NY.

MacDonald, C., Ounis, I., & Soboroff, I. (2007). Overview of the TREC-2007 blog track. In E. M. Voorhees & L. P. Buckland (Eds.), *NIST special publication: SP 500-274: The sixteenth text REtrieval conference (TREC 2007) Proceedings.* Gaithersburg, MD. http://trec.nist.gov/pubs/trec16/papers/BLOG.OVERVIEW16.pdf

Mei, Q., Cai, D., Zhang, D., & Zhai, C. (2008). Topic modeling with network regularization. In J. Huai & R. Chen (Eds.), *Proceeding of the 17th international conference on world wide web (WWW'08)* Beijing, China. New York, NY.

Mishne, G. (2007). Using blog properties to improve retrieval. In N. Glance, N. Nicolov, E. Adar, M. Hurst, M. Liberman, & F. Salvetto (Eds.), *Proceedings of the international conference on weblogs and social media (ICWSM).* Boulder, CO. http://www.icwsm.org/papers/paper25.html

Nowson, S., & Oberlander, J. (2007). Identifying more bloggers: Towards large scale personality classification of personal weblogs. In N. Glance, N. Nicolov, E. Adar, M. Hurst, M. Liberman, & F. Salvetto (Eds.), *Proceedings of the international conference on weblogs and social media (ICWSM).* Boulder, CO. http://www.icwsm.org/papers/paper4.html.

Pon, R. K., Cardenas, A. F., Buttler, D., & Critchlow, T. (2007). Tracking multiple topics for finding interesting articles. In P. Berkhin,

R. Caruana, & X. Wu (Eds.), *Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining*. San Jose, CA. New York, NY.

Subašić, I., & Berendt, B. (2008). Web mining for understanding stories through graph visualisation. In F. Giannotti, D. Gunopoulos, F. Turini, C. Zaniolo, N. Ramakrishnan, & X. Wu (Eds.), *Proceedings of the IEEE international conference on data mining (ICDM 08)*. Pisa, Italy. Los Alamitos, CA.

Weerkamp, W., Balog, K., & de Rijke, M. (2008). Finding key bloggers, one post at a time. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, & N. Avouris (Eds.), *Proceedings of the 18th European conference on artificial intelligence (ECAI 2008)*. Greece: Patras. Amsterdam, The Netherlands.

Text Mining for Spam Filtering

Aleksander Kołcz
Microsoft One Microsoft Way,
Redmond, WA, USA

Synonyms

Commerical email filtering; Junk email filtering; Spam detection; Unsolicited commercial email filtering

Definition

Spam filtering is the process of detecting unsolicited commercial email (UCE) messages on behalf of an individual recipient or a group of recipients. Machine learning applied to this problem is used to create discriminating models based on labeled and unlabeled examples of spam and nonspam. Such models can serve populations of users (e.g., departments, corporations, ISP customers) or they can be personalized to reflect the judgments of an individual. An important aspect of spam detection is the way in which textual information contained in email is extracted and used for the purpose of discrimination.

Motivation and Background

Spam has become the bane of existence for both Internet users and entities providing email services. Time is lost when sifting through unwanted messages and important emails may be lost through omission or accidental deletion. According to various statistics, spam constitutes the majority of emails sent today and a large portion of emails actually delivered. This translates to large costs related to bandwidth and storage use. Spam detection systems help to alleviate these issues, but they may introduce problems of their own, such as more complex user interfaces, delayed message delivery, and accidental filtering of legitimate messages. It is not clear if any one approach to fighting spam can lead to its complete eradication and a multitude of approaches have been proposed and implemented. Among existing techniques are those relying on the use of supervised and unsupervised machine learning techniques, which aim to derive a model differentiating spam from legitimate content using textual and nontextual attributes. These methods have become an important component of the antispam arsenal and draw from the body of related research such as text classification, fraud detection and cost-sensitive learning. The text mining component of these techniques is of particular prominence given that email messages are primarily composed of text. Application of machine learning and data mining to the spam domain is challenging, however, due, among others, to the adversarial nature of the problem (Dalvi, Domingos, Sanghai, & Verma, 2004; Fawcett, 2003).

Structure of the Learning System

Overview

A machine-learning approach to spam filtering relies on the acquisition of a learning sample of email data, which is then used to induce a classification or scoring model, followed by tuning and setup to satisfy the desired operating conditions. Domain knowledge may be injected at various stages into the induction process. For example, it is common to *a priori* specific features that are known be highly correlated with the spam label, e.g., certain patterns contained in email headers or certain words or phrases. Depending on the application environment, messages classified as spam are prevented from being delivered (e.g., are blocked or "bounced"), or are delivered with a mechanism to alert users to their likely spam nature. Filter deployment is followed by continuous evaluation of its performance, often accompanied by the collection of error feedback from its users.

Data Acquisition

A spam filtering system relies on the presence of labeled training data, which are used to induce a model of what constitutes spam and what is legitimate email. Spam detection represents a two-class problem, although it may sometimes be desired to introduce special handling

of messages for which a confident decision, either way, cannot be made. Depending on the application environment, the training data may represent emails received by one individual or a group of users. Ideally, the data should correspond to a uniform sample acquired over some period of time preceding filter deployment. Typical problems with data collection revolve around privacy issues, whereby users are unwilling to donate emails of personal or sensitive nature. Additionally, labeling mistakes are common where legitimate emails may be erroneously marked as spam or vice versa. Also, since for certain types of emails, the spam/legitimate distinction is personal, one may find that the same message content is labeled in a conflicting manner by different users (or even by the same user at different times). Therefore, data cleaning and conflict resolution techniques may need to be deployed, especially when building filters that serve a large and diverse user population.

Due to privacy concerns, few large publicly email corpora exist. The ones created for the TREC Spam Track (TREC data is available from: http://plg.uwaterloo.ca/~gvcormac/treccorpus/). stand out in terms of size and availability of published comparative results.

Content Encoding and Deobfuscation

Spam has been evolving in many ways over the course of time. Some changes reflect the shift in content advertised in such messages (e.g., from pornography and pharmaceuticals to stock schemes and *phish*). Others reflect the formatting of content. While early spam was sent in the form of plain text, it subsequently evolved into more complex HTML, with deliberate attempts to make extraction of meaningful textual features as difficult as possible. Typically, obfuscation (a list of obfuscation techniques is maintained at http://www.jgc.org/tsc.html) aims at

(a) Altering the text extracted from the message for words visible to the user (e.g., by breaking up the characters in message source by HTML tags, encoding the characters in various ways, using character look-alikes, wrapping the display of text using script code executed by the message viewer). This tactic is used to hide the message "payload."
(b) Adding content that is not visible to the user (e.g., using the background color or zero-width font to render certain characters/words). This tactic typically attempts to add "legitimate content."
(c) Purposeful misspelling of words known to be fairly incriminating (e.g., *Viagra* as *V1agr@*), in a way that allows the email recipient to still understand the spammer's message.

The line of detection countermeasures aiming at preventing effective content extraction continues in the form of image spam, where the payload message is encoded in the form of an image that is easily legible to a human but poses challenges to an automatic content extraction system. To the extent that rich and multimedia content gets sent out by legitimate users in increasing proportions, spammers are likely to use the complexity of these media to obfuscate their messages even further. The very fact that obfuscation is attempted, however, provides an opportunity for machine learning techniques to use obfuscation presence as a feature. Thus, even if payload content cannot be faithfully decoded, the very presence of elaborate encoding may help in identifying spam.

Feature Extraction and Selection

An email message represents a semistructured document, commonly following the rfc822 standard (www.faqs.org/rfcs/rfc822.html). Its header consists of fields indicative of formatting, authorship, and delivery information, while its body contains the actual content being sent. There can be little correctness enforcement of the header fields and spamming techniques often rely on spoofing and forging of the header data, although this may provide evidence of tempering. Many early approaches to detect spam depended predominantly on hand-crafted rules identifying inconsistencies and peculiarities of spam email headers. Manually or automatically generated header features continue to be relevant even when other features (e.g., message text) are considered.

Given that an email message tends to be primarily text, features traditionally useful in text categorization have also been found useful in spam detection. These include individual words, phrases, character n-grams, and other textual components (Siefkes, Assis, Chhabra, & Yerazunis, 2004). Natural language processing (NLP) techniques such as stemming, stop-word removal, and case folding are also sometimes applied to

normalize the features further. Text extraction is often nontrivial due to the application of content obfuscation techniques. For example, standard lexical feature extractors may need to be strengthened to correctly identify word boundaries (e.g., in cases where groups of characters within a word are separated by zero-width HTML tags).

Extraction of features from nontextual attachments (e.g., images, audio, and video) is also possible but tends to be more computationally demanding. Other types of features capture the way a message if formatted, encoded in HTML, composed of multiple parts, etc.

Although nontextual features have different properties than text, it is common practice to combine them with textual features and present a single unified representation to the classifier. Indeed, some approaches make no distinction between text and formatting even during the process of feature extraction, and apply pattern discovery techniques to identifying complex features automatically (Rigoutsos & Huynh, 2004). The advantage of such techniques is that they do not require rich domain knowledge and can discover new useful patterns. Due to the large space of possible patterns they can potentially be computationally expensive. However, even the seemingly simplistic treatment of an email message as a plain-text document with "words" delimited by white space often leads to very good results.

Even though typical text documents are already very sparse, the problem is even more pronounced for the email medium due to frequent misspelling and deliberate randomization performed by spammers. Insisting on using all such variations may lead to overfitting for some classifiers, and it leads to large filter memory footprints that are undesirable from an operational standpoint. However, due to the constantly changing distribution of content, it may be dangerous to rely on very few features. Traditional approaches to feature selection based on measures such as Information Gain have been reported as successful in the spam filtering domain, but even simple rudimentary attribute selection based on removing very rare and/or very frequent features tends to work well.

There are a number of entities that can be extracted from message text and that tend to be of relevance in spam detection. Among others, there are telephone numbers and URLs. In commercial email and in spam, these provide a means of ordering products and services and thus, offer important information for vendor and campaign tracking. Detection of signature and mailing address blocks can also be of interest, even if only to indicate their presence or absence.

Learning Algorithms

A variety of learning algorithms have been applied in the spam filtering domain. These range from linear classifiers such as Naive Bayes (Metsis, Androutsopoulos, & Paliouras, 2006), logistic regression (Goodman & Yih, 2006), or linear support vector machines (Drucker, Wu, & Vapnik, 1999; Kołcz & Alspector, 2001; Sculley & Wachman, 2007) to nonlinear ones such as boosted decision trees (Carreras & Màrquez, 2001). Language modeling and statistical compression techniques have also been found quite effective (Bratko, Cormack, Filipic, Lynam, & Zupan, 2006). In general, due to the high dimensionality of the feature space, the classifier chosen should be able to handle tens of thousand, or more, attributes without overfitting the training data.

It is usually required that the learned model provides a scoring function, such that for email message x $\text{score}(x) \in R$, with higher score values corresponding to higher probability of the message being spam. The score function can also be calibrated to represent the posterior probability $P(\text{spam}|x) \in [0,1]$, although accurate calibration is difficult due to constantly changing class and content distributions. The scoring function is used to establish a decision rule:

$$\text{score}(x) \geq th \rightarrow \text{spam}$$

where the choice of the decision threshold th is driven by the minimization of the expected cost. In the linear case, the scoring function takes the form

$$\text{score}(x) = w \cdot x + b$$

where w is the weight vectors, and x is a vector representation of the message. Sometimes scores are normalized with a monotonic function, e.g., to give an estimate of the probability of x being spam.

Linear classifiers tend to provide sufficiently high accuracy, which is also consistent with other application domains involving the text medium. In particular,

many variants of the relatively simple Naive Bayes classifier have been found successful in detecting spam, and Naive Bayes often provides a baseline for systems employing more complex classification algorithms (Metsis et al., 2006).

One Model versus Multiple Models

It often pays off to combine different types of classifiers (even different linear ones) in a sequential or parallel fashion to benefit from the fact that different classifiers may provide an advantage in different regions of the feature space. Stacking via ▶linear regression has been reported to be effective for this purpose (Sakkis et al., 2001; Segal, Crawford, Kephart, & Leiba, 2004). One can generally distinguish between cases where all classifiers are induced over the same data and cases where several different datasets are used. In the former case, the combination process exploits the biases of different learning algorithms. In the latter case, one can consider building a multitude of detectors, each targeting a different subclass of spam (e.g., phish, pharmaceutical spam, "Nigerian" scams, etc.). Datasets can also be defined on a temporal basis, so that different classifiers have shorter or longer memory spans. Other criteria of providing different datasets are also possible (e.g., based on the language of the message).

Additional levels of complexity in the classifier combination process can be introduced by considering alternative feature representations for each dataset. For example, a single data collection and a single learning method can be used to create several different classifiers, based upon alternative representations of the same data (e.g., using just the header features or just the message text features).

The method of classifier combination will necessarily depend on their performance and intended area of applications. The combination regimes can range from simple logical-OR through linear combinations to complex nonlinear rules, either derived automatically to maximize the desired performance or specified manually with the guidance of expert domain knowledge.

Off-line Adaptation Versus Online Adaptation

A spam filtering system can be configured to receive instant feedback from its users, informing it whenever certain messages get misdelivered (this necessarily does not include cases where misclassified legitimate messages are simply blocked). In the case of online filters, the feedback information may be immediately used to update the filtering profile. This allows a filter to adjust to the changing distribution of email content and to detection countermeasures employed by spammers. Not all classifiers are easily amenable to the online learning update, although online versions of well-known learners such as logistic regression (Goodman & Yih, 2006) and linear SVMs (Sculley & Wachman, 2007) have been proposed. The distinguishing factor is the amount of the original training data that needs to be retained in addition to the model itself to perform future updates. In this respect, Naive Bayes is particularly attractive since it does not require any of the original data for adaptation, with the model itself providing all the necessary information.

One issue with the user feedback signal, however, is its bias toward current errors of the classifier, which for learners depending on the training data being an unbiased sample drawn from the underlying distribution may lead to overcompensation rather than an improvement in filtering accuracy. As an alternative, unbiased feedback can be obtained by either selectively querying users about the nature of uniformly sampled messages or by deriving the labels implicitly.

In the case where off-line adaptation is in use, the feedback data is collected and saved for later use, whereby the filtering models are retrained periodically or only as needed using the data collected. The advantage of off-line adaptation is that it offers more flexibility in terms of the learning algorithm and its optimization. In particular, model retraining can take advantage of a larger quantity of data, and does not have to be constrained to be an extension of the current version of the model. As a result, it is, e.g., possible to redefine the features from one version of the spam filter to the next. One disadvantage is that model updates are likely to be performed less frequently and may be lagging behind the most recent spam trends.

User-specific Versus User-independent Spam Detection

What constitutes a spam message tends to be personal, at least for some types of spam. Various commercial messages, such as promotions and advertisements, e.g., may be distributed in a solicited or unsolicited manner, and sometimes only the end recipient may be able to

judge which. In that sense, user-specific spam detection has the potential of being most accurate, since a user's own judgments are used to drive the training process. Since the nonspam content received by any particular user is likely to be more narrowly distributed when compared a larger user population, this makes the discrimination problem much simpler. Additionally, in the adversarial context, a spammer should find it more difficult to measure the success of penetrating personalized filter defenses, which makes it more difficult to craft a campaign that reaches sufficiently many mail inboxes to be profitable.

One potential disadvantage of such solutions is the need for acquiring labeled data on a user by user basis, which may be challenging. For some users historical data may not yet exist (or has already been destroyed), for others even if such data exist, labeling may seem too much of a burden for the users. Aside from the data collection issues, personal spam filtering faces maintainability issues, as the filter is inherently controlled by its user. This may result in less-than-perfect performance, e.g., if the user misdirects filter training.

From the perspective of institutions and email service providers, it is often more attractive to maintain just one set of spam filters that service a larger user population. This makes them simpler to operate and maintain, but their accuracy may depend on the context of any particular user. The advantage of centralized filtering when serving large user populations is that global trends can be more readily spotted and any particular user may be automatically protected against spam, affecting other users. Also, the domain knowledge of the spam-filtering analysts can be readily injected into the filtering pipeline.

To the extent that a service provider maintains personal filters for its population of users, there are potential large system costs to account for, so that a complete cost-benefit analysis needs to be performed to assess the suitability of such as solution as opposed to a user-independent filtering complex. More details on the nature of such trade-offs can be found in (Kołcz, Bond, & Sargent, 2006).

Clustering and Volumetric Techniques

Content clustering can serve as an important data understanding technique in spam filtering. For example, large clusters can justify the use of specialized classifiers and/or the use of cost-sensitive approaches in classifier learning and evaluation (e.g., where different costs are assigned to different groups of content within each class (Kołcz & Alspector, 2001).

Both spam and legitimate commercial emails are often sent in large campaigns, where the same or highly similar content is sent to a large number of recipients, sometimes over prolonged periods of time. Detection of email campaigns can therefore play an important role in spam filtering. Since individual messages of a campaign are highly similar to one another, this can be considered a variant of near-replica document detection (Kołcz, 2005). It can also be seen as relying on identification of highly localized spikes in the content density distribution. As found in (Yoshida et al., 2004), density distribution approaches can be highly effective, which is especially attractive given that they do not require the explicitly labeled training data. Tracking of spam campaigns may be made difficult due to content randomization, and some research has been directed at making the detection methods robust in the presence such countermeasures (Kołcz, 2005; Kołcz & Chowdhury, 2007).

Misclassification Costs and Filter Evaluation

An important aspect of spam filtering is that the costs of misclassifying spam as legitimate email are not the same as the costs of making the opposite mistake. It is thus commonly assumed that the costs of a false positive mistake (i.e., a legitimate email being misclassified as spam) are much higher than the cost of mistaking spam for legitimate email. Given the prevalence of spam π and the false-spam (FS) and false-legitimate (FL) rates of the classifier, the misclassification cost c can be expressed as

$$c = C_{FS} \cdot (1 - \pi) \cdot FS + C_{FL} \cdot \pi \cdot FL$$

where C_{FS} and C_{FL} are the costs of making a false-spam and false-legitimate mistake, respectively (there is no penalty for making the correct decision). Since actual values of C_{FS} and C_{FL} are difficult to quantify, one typically sees them combined in the form of a ratio, $\lambda = C_{FS}/C_{FL}$, and the overall cost can be expressed as relative to the cost of a false-legitimate misclassification e.g.,

$$c_{\text{rel}} = \lambda \cdot (1 - \pi) \cdot \text{FS} + \pi \cdot \text{FL}$$

Practical choices of λ tend to range from 1 to 1,000. Nonuniform misclassification costs can be used during the process of model induction or in postprocessing when setting up the operating parameters of a spam filter, e.g., using the receiver operating characteristic (ROC) analysis.

Since the costs and cost ratios are sometimes hard to define, some approaches to evaluation favor direct values of the false-spam and false-legitimate error rates. This captures the intuitive requirement that an effective spam filter should provide high detection rate at a close-to-zero false-spam rate. Alternatively, threshold independent metrics such as the area under the ROC (AUC) can be used (Bratko et al., 2006; Cormack & Lynam, 2006), although other measures have also been proposed (Sakkis et al., 2001).

Adaptation to Countermeasures

Spam filtering is an inherently adversarial task, where any solution deployed on a large scale is likely to be met with a response on the part of the spammers. To that extent that the success of a spam filter can be pinpointed to any particular component (e.g., the type of features used), that prominent component is likely to be attacked directly and may become a victim of its own success. For example, the use of word features in spam filtering encourages countermeasures in the form of deliberate misspellings, word fragmentation, and "invisible ink" in HTML documents. Also, since some words are considered by a model inherently more legitimate than others, "word stuffing" has been used to inject large blocks of potentially legitimate vocabulary into an otherwise spammy message in the hope that this information outweighs the evidence provided by the spam content (Lowd & Meek, 2005).

Some authors have attempted to put the adversarial nature of spam filtering in the formal context of game theory (Dalvi et al., 2004). One difficulty of drawing broad conclusion based on such analyses is the breadth of the potential attack/defense front, of which only small sections have been successfully captured in the game-theory formalism. The research on countering the countermeasures points at using multiple diverse filtering components, normalization of features to keep them invariant to irrelevant alterations. A key point is that frequent filter retraining is likely to help in keeping up with the shifts in content distribution, both natural and due to countermeasures.

Future Directions

Reputation Systems and Social Networks

There has been a growing interest in developing reputation systems capturing the trustworthiness of a sender with respect to a particular user or group of users. To this end however, the identity of the sender needs to be reliably verified, which poses challenges and presents a target for potential abuses of such systems. Nevertheless, reputation systems are likely to grow in importance, since they are intuitive from the user perspective in capturing the communication relationships between users. Sender reputation can be hard or soft. In the hard variant, the recipient always accepts or declines messages from a given sender. In the soft variant, the reputation reflects the level of trustworthiness of the sender in the context of the given recipient. When sender identities resolve to individual email addresses, the reputation system can be learned via analysis of a large social network that documents who exchanges email with whom. The sender identities can also be broader however, e.g., assigning reputation to a particular mail server or all mail servers responsible for handling the outbound traffic for a particular domain. On the recipient side, reputation can also be understood globally to represent the trustworthiness of the sender with respect to all recipients hosted by the system. Many open questions remain with regard to computing and maintaining reputations as well as using them effectively to improve spam detection. In the context of text mining, one such question is the extent to which email content analysis can be used to aid the process of reputation assessment.

Cross References

►Cost-Sensitive Learning
►Logistic Regression
►Naive Bayes
►Support Vector Machines
►Text Categorization

Recommended Reading

Bratko, A., Cormack, G. V., Filipic, B., Lynam, T. R., & Zupan, B. (2006). Spam filtering using statistical data compression models. *Journal of Machine Learning Research, 7*, 2673–2698.

Carreras, X., & Màrquez, L. (2001). Boosting trees for anti-spam email filtering. In *Proceedings of RANLP-01, the 4th international conference on recent advances in natural language processing*. New York: ACM.

Cormack, G. V., & Lynam, T. R. (2006). On-line supervised spam filter evaluation. *ACM Transactions on Information Systems, 25(3)*, 11.

Dalvi, N., Domingos, P., Sanghai, M. S., & Verma, D. (2004). Adversarial classification. In *Proceedings of the tenth international conference on knowledge discovery and data mining* (Vol. 1, pp. 99–108). New York: ACM.

Drucker, H., Wu, D., & Vapnik, V. N. (1999). Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, *5*(10), 1048–1054.

Fawcett, T. (2003). In vivo' spam filtering: A challenge problem for data mining. *KDD Explorations*, *5*(2), 140–148.

Goodman, J., & Yih, W. (2006). Online discriminative spam filter training. In *Proceedings of the third conference on email and anti-spam*.Mountain View, CA. (CEAS-2006).

Kołcz, A. (2005). Local sparsity control for naive bayes with extreme misclassification costs. In *Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery and data mining*. New York: ACM.

Kołcz, A., & Alspector, J. (2001). *SVM-based filtering of e-mail spam with content-specific misclassification costs*. TextDM'2001 (IEEE ICDM-2001 workshop on text mining), San Jose, CA.

Kołcz, A., Bond, M., & Sargent, J. (2006). The challenges of service-side personalized spam filtering: Scalability and beyond. In *Proceedings of the first international conference on scalable information systems (INFOSCALE)*. New York: ACM.

Kołcz, A. M., & Chowdhury, A. (2007). Hardening fingerprinting by context. In *Proceedings of the fourth international conference on email and anti-spam*.

Lowd, D., & Meek, C. (2005). Good word attacks on statistical spam filters. In *Proceedings of the second conference on email and anti-spam*. Mountain View, CA. (CEAS-2005).

Metsis, V., Androutsopoulos, I., & Paliouras, G. (2006). Spam filtering with naive bayes - which naive bayes? In *Proceedings of the third conference on email and anti-spam*. (CEAS-2006).

Rigoutsos, I., & Huynh, T. (2004). Chung-Kwei: a pattern-discovery-based system for the automatic identification of unsolicited e-mail messages (SPAM). In *Proceedings of the first conference on email and anti-spam*. (CEAS-2004).

Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (1998). *A Bayesian approach to filtering junk email*. AAAI workshop on learning for text categorization, Madison, Wisconsin. AAAI Technical Report WS-98-05.

Sakkis, G., Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Spyropoulos, C. D., & Stamatopoulos, P. (2001). Stacking classifiers for anti-spam filtering of e-mail. In L. Lee & D. Harman (Eds.). *Proceedings of empirical methods in natural language processing (EMNLP 2001)* (pp. 44–50). http://www.cs.cornell.edu/home/llee/emnlp/proceeding.html.

Sculley, D., & Wachman, G. (2007). Relaxed online support vector machines for spam filtering. In *Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval*. New York: ACM.

Segal, R., Crawford, J., Kephart, J., & Leiba, B. (2004). SpamGuru: An enterprise anti-spam filtering system. In *Proceedings of the first conference on email and anti-spam*. (CEAS-2004).

Siefkes, C., Assis, F., Chhabra, S., & Yerazunis, W. (2004). Combining winnow and orthogonal sparse bigrams for incremental spam filtering. In *Proceedings of the european conference on principle and practice of knowledge discovery in databases*. New York: Springer.

Yoshida, K., Adachi, F., Washio, T., Motoda, H., Homma, T., Nakashima, A., et al. (2004). Densitiy-based spam detection. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 486–493). New York: ACM.

Text Mining for the Semantic Web

Marko Grobelnik[1], Dunja Mladenić[1],
Michael Witbrock[2]
[1]Jožef Stefan Institute
[2]Cycorp Inc
Executive Center Drive, Austin, TX, USA

Definition

►Text mining methods allow for the incorporation of textual data within applications of semantic technologies on the Web. Application of these techniques is appropriate when some of the data needed for a Semantic Web use scenario are in textual form. The techniques range from simple processing of text to reducing vocabulary size, through applying shallow natural language processing to constructing new semantic features or applying information retrieval to selecting relevant texts for analysis, through complex methods involving integrated visualization of semantic information, semantic search, semiautomatic ontology construction, and large-scale reasoning.

Motivation and Background

Semantic Web applications usually involve deep structured knowledge integrated by means of some kind of ontology. Text mining methods, on the other hand, support the discovery of structure in data and effectively support semantic technologies on data-driven tasks such as, (semi)automatic ontology acquisition, extension, and mapping. Fully automatic text mining

approaches are not always the most appropriate because often it is too difficult or too costly to fully integrate the available background domain knowledge into a suitable representation. For such cases semiautomatic methods, such as ►active learning and ►semisupervised text processing (►Semisupervised Learning), can be applied to make use of small pieces of human knowledge to provide guidance toward the desired ontology or other model. Application of these semiautomated techniques can reduce the amount of human effort required to produce training data by an order of magnitude while preserving the quality of results.

To date, Semantic Web applications have typically been associated with data, such as text documents and corresponding metadata that have been designed to be relatively easily manageable by humans. Humans are, for example, very good at reading and understanding text and tables. General semantic technologies, on the other hand, aim more broadly at handling data modalities including multimedia, signals from emplaced or remote sensors, and the structure and content of communication and transportation graphs and networks. In handling such multimodal data, much of which is not readily comprehensible by unaugmented humans, there must be significant emphasis on fully- or semiautomatic methods offered by knowledge discovery technologies whose application is not limited to a specific data representation (Grobelnik & Mladenic, 2005).

Data and the corresponding semantic structures change over time, and semantic technologies also aim at adapting the ontologies that model the data accordingly. For most such scenarios extensive human involvement in building models and adapting them according to the data is too costly, too inaccurate, and too slow. Stream mining (Gaber, Zaslavsky, & Krishnaswamy, 2005) techniques (►Data Streams: Clustering) allow text mining of dynamic data (e.g., notably in handling a stream of news or of public commentary).

Structure of Learning System

Ontology is a fundamental method for organizing knowledge in a structured way, and is applied, along with formalized reasoning, in areas from philosophy to scientific discovery to knowledge management and the Semantic Web. In computer science, an ontology generally refers to a graph or network structure consisting of a set of concepts (vertices in a graph), a set of relationships connecting those concepts (directed edges in a graph) and, possibly, a set of distinguished instance concepts assigned to particular class concepts (data records assigned to vertices in a graph). In many cases, knowledge is structured in this way to allow for automated inference based on a logical formalism such as the predicate calculus (Barwise & Etchemendy, 2002); for these applications, an ontology often further comprises a set of rules or produces new knowledge within the representation from existing knowledge. An ontology containing instance data and rules is often referred to as a knowledge base (KB) (e.g., Lenat, 1995).

Machine learning practitioners refer to the task of constructing these ontologies as *ontology learning*. From this point of view, an ontology is seen a class of models – somewhat more complex than most used in machine learning – which need to be expressed in some ►hypothesis language. This definition of ontology learning (from Grobelnik & Mladenic, 2005) enables a decomposition into several machine learning tasks, including learning concepts, identifying relationships between existing concepts, populating an existing ontology/structure with instances, identifying change in dynamic ontologies, and inducing rules over concepts, background knowledge, and instances.

Following this scheme, text mining methods have been applied to extending existing ontologies based on Web documents, learning semantic relations from text based on collocations, semiautomatic data driven ontology construction based on document clustering and classification, extracting semantic graphs from text, transforming text into RDF triples (a commonly used Semantic Web data representation), and mapping triplets to semantic classes using several kinds of lexical and ontological background knowledge. Text mining is also intensively used in the effort to produce a Semantic Web for annotation of text with concepts from ontology. For instance, a text document is split into sentences, each sentence is represented as a word-vector, sentences are clustered, and each cluster is labeled by the most characteristic words from its sentences and mapped upon the concepts of a general ontology. Several approaches that integrate ontology management, knowledge discovery, and human language technologies are described in (Davies, Grobelnik, & Mladenić, 2009).

Extending the text mining paradigm, current efforts are beginning to aim at giving machines an approximation of the full human ability to acquire knowledge from text. Machine reading aims at full text understanding by integrating knowledge-based construction and use into syntactically sophisticated natural language analysis, leading to systems that autonomously improve their ability to extract further knowledge from text (e.g., Curtis et al., 2009; Etzioni, Banko, & Cafarella, 2007; Mitchell, 2005).

Cross References

►Active Learning
►Classification
►Document Clustering
►Semisupervised Learning
►Semisupervised Text Processing
►Text Mining
►Text Visualization

Recommended Reading

Barwise, J., & Etchemendy, J. (2002). *Language proof and logic.* Center for the Study of Language and Information. ISBN, 157586374X.

Buitelaar, P., Cimiano, P., & Magnini, B. (2005). *Ontology learning from text: Methods, applications and evaluation, frontiers in artificial intelligence and applications*. The Netherlands: IOS Press.

Curtis, J., Baxter, D., Wagner, P., Cabral, J., Schneider, D., & Witbrock, M. (2009). Methods of rule acquisition in the TextLearner system. In *Proceedings of the 2009 AAAI spring symposium on learning by reading and learning to read* (pp. 22–28). Palo Alto, CA: AAAI Press.

Davies, J., Grobelnik, M., & Mladenić, D. (2009). *Semantic knowledge management*. Berlin: Springer.

Etzioni, O., Banko, M., & Cafarella, M. J. (2007). Machine Reading. In Proceedings of the 2007 AAAI spring symposium on machine reading.

Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2005).Mining data streams: A review. *ACM SIGMOD Record, 34*(1), 18–26. ISSN: 0163-580

Grobelnik, M., & Mladenic, D. (2005). Automated knowledge discovery in advanced knowledge management. *Journal of Knowledge Management*, *9*, 132–149.

Lenat, D. B. (1995). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM, 38*(11), 33–38.

Mitchell, T. (2005). Reading the web: A breakthrough goal for AI. celebrating twenty-five years of AAAI: Notes from the AAAI-05 and IAAI-05 conferences. *AI Magazine*, *26*(3), 12–16.

Text Spatialization

►Text Visualization

Text Visualization

John Risch[1], Shawn Bohn[1], Steve Poteet[2], Anne Kao[2], Lesley Quach[2], Jason Wu[2]
[1]Pacific Northwest National Laboratory
[2]Boeing Phantom Works, Seattle, WA, USA

Synonyms

Semantic mapping; Text spatialization; Topic mapping

Definition

The term *text visualization* describes a class of knowledge discovery techniques that use interactive graphical representations of textual data to enable knowledge discovery via recruitment of human visual pattern recognition and spatial reasoning capabilities. It is a subclass of *information visualization*, which more generally encompasses visualization of nonphysically based (or "abstract") data of all types. Text visualization is distinguished by its focus on the unstructured (or *free text*) component of information. While the term "text visualization" has been used to describe a variety of graphical methods for deriving knowledge from text, it is most closely associated with techniques for depicting the semantic characteristics of large document collections. Text visualization systems commonly employ unsupervised machine learning techniques as part of broader strategies for organizing and graphically representing such collections.

Motivation and Background

The Internet enables universal access to vast quantities of information, most of which (despite admirable efforts (Berners-Lee, Hendler, & Lassila, 2001)) exists in the form of unstructured and unorganized text. Advancements in search technology make it possible to retrieve large quantities of this information with reasonable precision; however, only a tiny fraction of the information available on any given topic can be effectively exploited.

Text visualization technologies, as forms of computer-supported knowledge discovery, aim to improve our ability to understand and utilize the wealth of text-based information available to us. While the term "text visualization" has been used to describe a variety of techniques for graphically depicting the characteristics of free-text data (Havre, Hetzler, Whitney, & Nowell, 2002; Small, 1996), it is most closely associated with the so-called *semantic clustering* or *semantic mapping* techniques (Chalmers & Chitson, 1992; Kohonen et al., 2000; Lin, Soergel, & Marchionini, 1991; Wise et al., 1995). These methods attempt to generate summary representations of document collections that convey information about their general topical content and similarity structure, facilitating general domain understanding and analytical reasoning processes.

Text visualization methods are generally based on vector-space models of text collections (Salton, 1989), which are commonly used in information retrieval, clustering, and categorization. Such models represent the text content of individual documents in the form of vectors of frequencies of the terms (*text features*) they contain. A document collection is therefore represented as a collection of vectors. Because the number of unique terms present in a document collection generally is in the range of tens of thousands, a dimensionality reduction method such as singular value decomposition (SVD) (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990) or other matrix decomposition method (Kao, Poteet, Ferng, Wu, & Quach, 2008; Booker et al., 1999) is typically used to eliminate noise terms and reduce the length of the document vectors to a tractable size (e.g., 50–250 dimensions). Some systems attempt to first identify discriminating features in the text and then use mutual probabilities to specify the vector space (York, Bohn, Pennock, & Lantrip, 1995).

To enable visualization, the dimensions must be further reduced to two or three. The goal is a graphical representation that employs a "spatial proximity means conceptual similarity" metaphor where topically similar text documents are represented as nearby points in the display. Various regions of the semantic map are subsequently labeled with descriptive terms that convey the primary concepts described by nearby documents. The text visualization can thus serve as a kind of graphical "table of contents" depicting the conceptual similarity structure of the collection.

Text visualization systems therefore typically implement four key functional components, namely,

1. A *tokenization* component that characterizes the lexical content of text units via extraction, normalization, and selection of key terms
2. A *vector-space modeling* component that generates a computationally tractable vector space representation of a collection of text units
3. A *spatialization* component that uses the vector space model to generate a 2D or 3D spatial configuration that places the points representing conceptually similar text units in near spatial proximity
4. A *labeling* component that assigns characteristic text labels to various regions of the semantic map

Although machine learning techniques can be used in several of these steps, their primary usage is in the spatialization stage. An unsupervised learning algorithm is typically used to find meaningful low-dimensional structures hidden in high-dimensional document feature spaces.

Structure of Learning System

Spatialization is a term generically used in ▶information visualization to describe the process of generating a spatial representation of inherently nonspatial information. In the context of text visualization, this term generally refers to the application of a nonlinear dimensionality reduction algorithm to a collection of text vectors in order to generate a visually interpretable two- or three-dimensional representation of the similarity structure of the collection. The goal is the creation of a *semantic similarity map* that positions graphical features representing text units (e.g., documents) conceptually similar to one another near one another in the visualization display. These maps may be further abstracted to produce more general summary representations of text collections that do not explicitly depict the individual text units themselves (Wise et al., 1995).

A key assumption in text visualization is that text units which express similar concepts will employ similar word patterns, and that the existence of these

word correlations creates coherent structures in high-dimensional text feature spaces. A further assumption is that text feature spaces are nonlinear, but that their structural characteristics can be approximated by a smoothly varying low-dimensional manifold. The text spatialization problem thus becomes one of finding an embedding of the feature vectors in a two- or three-dimensional manifold that best approximates this structure. Because the intrinsic dimensionality of the data is invariably much larger than two (or three), significant distortion is unavoidable. However, because the goal of text visualization is not necessarily the development of an accurate representation of interdocument similarities, but rather the depiction of broad (and ambiguously defined) semantic relationships, this distortion is generally considered acceptable.

Text vector spatialization therefore involves the fitting of a model into a collection of observations. Most text visualization systems developed to date have employed some type of unsupervised learning algorithm for this purpose. In general, the desired characteristics of an algorithm used for text spatialization include that it (1) preserves global properties of the input space, (2) preserves the pairwise input distances to the greatest extent possible, (3) supports out-of-sample extension (i.e., the incremental addition of new documents), and (4) has low computational and memory complexity. Computational and memory costs are key considerations, as a primary goal of text visualization is the management and interpretation of extremely large quantities of textual information.

A leading approach is to iteratively adapt the nodes of a fixed topology mesh to the high-dimensional feature space via adaptive refinement. This is the basis of the well-known Kohonen feature mapping algorithm, more commonly referred to as the ▶self-organizing map (SOM) (Kohonen, 1997). In a competitive learning process, text vectors are presented one at a time to a (typically triangular) grid, the nodes of which have been randomly initialized to points in the term space. The Euclidean distance to each node is computed, and the node closest to the sample is identified. The position of the winning node, along with those of it's topologically nearest neighbors, is incrementally adjusted toward the sample vector. The magnitude of the adjustments is gradually decreased over time. The process is generally repeated using every vector in the set for many hundreds or thousands of cycles until the mesh converges on a solution. At the conclusion, the samples are assigned to their nearest nodes, and the results are presented as a uniform grid. In the final step, the nodes of the grid are labeled with summary terms which describe the key concepts associated with the text units that have been assigned to them.

Although self-organizing maps can be considered primarily a clustering technique, the grid itself theoretically preserves the topological properties of the input feature space. As a consequence, samples that are nearest neighbors in the feature space generally end up in topologically adjacent nodes. However, while SOMs are topology-preserving, they are not distance-preserving. Vectors that are spatially distant in the input space may be presented as proximal in the output, which may be semantically undesirable. SOMs have a number of attractive characteristics, including straightforward out-of-sample extension and low computational and memory complexity. Examples of the use of SOMs in text visualization applications can be found in (Lin et al., 1991; Kaski, Honkela, Lagus, & Kohonen, 1998; Kohonen et al., 2000).

Often, it is considered desirable to attempt to preserve the distances among the samples in the input space to the greatest extent possible in the output. The rationale is that the spatial proximities of the text vectors capture important and meaningful characteristics of the associated text units: spatial "nearness" corresponds to conceptual "nearness." As a consequence, many text visualization systems employ distance-preserving dimensionality reduction algorithms. By far the most commonly used among these is the class of algorithms known as *multidimensional scaling* (MDS) algorithms.

Multidimensional scaling is "a term used to describe any procedure which starts with the 'distances' between a set of points (or individuals or objects) and finds a configuration of the points, preferably in a smaller number of dimensions, usually 2 or 3" ((Chatfield & Collins, 1980), quoted in Chalmers & Chitson, 1992). There are two main subclasses of MDS algorithms. Metric (quantitative, also known as classical) MDS algorithms attempt to preserve the pairwise input distances to the greatest extent possible in the output

configuration, while nonmetric (qualitative) techniques attempt only to preserve the rank order of the distances. Metric techniques are most commonly employed in text visualization.

Metric MDS maps the points in the input space to the output space while maintaining the pairwise distances among the points to the greatest extent possible. The quality of the mapping is expressed in a stress function which is minimized using any of a variety of optimization methods, e.g., via eigen decomposition of a pairwise dissimilarity matrix, or using iterative techniques such as generalized Newton–Raphson, simulated annealing, or genetic algorithms. A simple example of a stress function is the raw stress function (Kruskal, 1964) defined by

$$\phi(Y) = \sum_{ij} (||x_i - x_j|| - ||y_i - y_j||)^2$$

in which $||x_i - x_j||$ is the Euclidean distance between points x_i and x_j in the high-dimensional space, and $||y_i - y_j||$ is the distance between the corresponding points in the output space. A variety of alternative stress functions have been proposed (Cox & Cox, 2001). In addition to its distance-preserving characteristics, MDS has the added advantage of preserving the global properties of the input space. A major disadvantage of MDS, however, is its high computational complexity, which is approximately $O(kN^2)$, where N is the number of data points and k is the dimensionality of the embedding. Although computationally expensive, MDS can be used practically on data sets of up to several hundred documents in size. Another disadvantage is that out-of-core extension requires reprocessing of the full data set if an optimization method which computes the output coordinates all at once is used.

The popularity of MDS methods has led to the development of a range of strategies for improving on its computational efficiency to enable scaling of the technique to text collections of larger size. One approach is to use either cluster centroids or a randomly sampled subset of input vectors as surrogates for the full set. The surrogates are down-projected independently using MDS, then the remainder of the data is projected relative to this "framework" using a less expensive algorithm, e.g., distance-based triangulation. This is the basis for the *anchored least stress* algorithm used in the SPIRE text visualization system (York et al., 1995), as well as the more recently developed Landmark MDS (de Silva & Tenenbaum, 2003) algorithm.

While self-organizing maps and multidimensional scaling techniques have received the most attention to date, a number of other machine learning techniques have also been used for text spatialization. The Starlight system (Risch et al., 1999) uses *stochastic proximity embedding* (Agrafiotis, 2003), a high-speed nonlinear manifold learning algorithm. Other approaches have employed methods based on graph layout techniques (Fabrikant, 2001). Generally speaking, any of a number of techniques for performing dimensionality reduction in a correlated system of measurements (classified under the rubric of factor analysis in statistics) may be employed for this purpose.

Machine learning algorithms can also be used in text visualization for tasks other than text vector spatialization. For example, generation of descriptive labels for semantic maps requires partitioning of the text units into related sets. Typically, a partitioning-type ►clustering algorithm such as K-means is used for this purpose (see ►Partitional Clustering), either as an element of the spatialization strategy (see York et al., 1995), or as a postspatialization step. The labeling process itself may also employ machine learning algorithms. For instance, the TRUST system (Booker et al., 1999; Kao et al., 2008) employed by Starlight generates meaningful labels for document clusters using a kind of ►unsupervised learning. By projecting a cluster centroid defined in the reduced dimensional representation (e.g., 50–250 dimensions) back into the full term space, terms related to the content of the documents in the cluster are identified and used as summary terms. Machine learning techniques can also be applied indirectly during the tokenization phase of text visualization. For example, information extraction systems commonly employ rule sets that have been generated by a supervised learning algorithm (Mooney & Bunescu, 2006). Such systems may be used to identify tokens that are most characteristic of the overall topic of a text unit, or are otherwise of interest (e.g., the names of people or places). In this way, the dimensionality of the input space can be drastically reduced, accelerating downstream processing while

simultaneously improving the quality of the resulting visualizations.

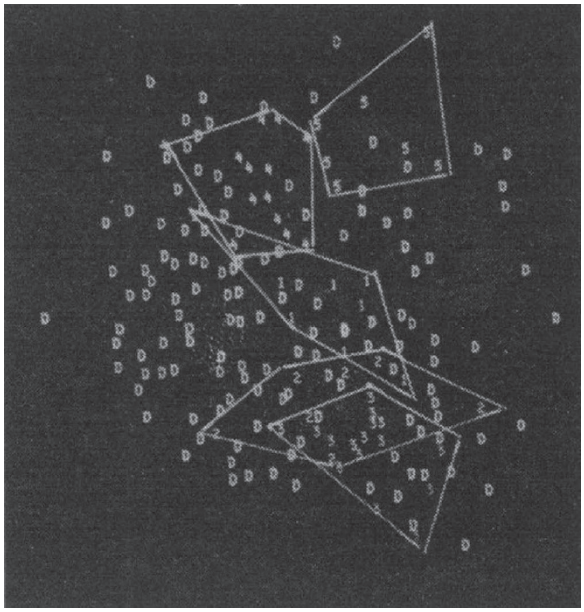
Applications

Sammon

The first text visualization system based on a text vector space model was likely a prototype developed in the 1960s by John Sammon's "nonlinear mapping," or *NLM*, algorithm (today referred to as organizing text data). The configuration depicted here is the result of applying Sammon's algorithm to a collection of 188 documents represented as 17-dimensional vectors determined according to document relevance to 1,125 key words and phrases. Among other interesting and prescient ideas, Sammon describes techniques for interacting with text visualizations depicted on a "CRT display" using a light pen (Fig. 1).

Lin

Lin's 1991 prototype (Lin et al., 1991) was one of the first to demonstrate the use of self-organizing maps for organizing text documents. Lin formed a 25-dimensional vector space model of a 140 document collection using 25 key index terms extracted from the text. The document vectors were used to train a 140 node feature map, generating the result shown here (the fact that the number of nodes matches the number of documents is coincidental). Lin was also among the first to assign text labels to various regions of the resulting map to improve the interpretability and utility of the resulting product (Fig. 2).



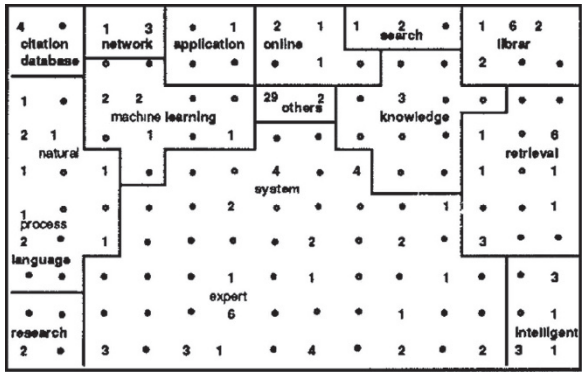
Text Visualization. Figure 1.

BEAD

The BEAD system (Chalmers & Chitson, 1992) was a text visualization prototype developed during the early 1990s at Rank Xerox EuroPARC. BEAD employed a vector space model constructed using document keywords and a hybrid MDS algorithm based on an optimized form of simulated annealing. Although it did not include a region labeling component, BEAD did support highlighting of visualization features in response to query operations, a now standard text visualization system feature. The BEAD project also pioneered a number of now common interaction techniques, and was among the first to explore 3D representations of document collections (Fig. 3).

IN-SPIRE

IN-SPIRE (formerly SPIRE, Spatial Paradigm for Information Retrieval and Exploration) (Wise et al., 1995), was originally developed in 1995 at Pacific Northwest National Laboratory (PNNL). Over the years, IN-SPIRE has evolved from using MDS, to Anchored Least Stress, to a hybrid clustering/PCA projection scheme. The SPIRE/IN-SPIRE system introduced several new concepts, including the use of a 3D "landscape" abstraction (called a *ThemeView*) for depicting the general characteristics of large text collections. A recently developed parallelized version of the software is capable of



Text Visualization. Figure 2.

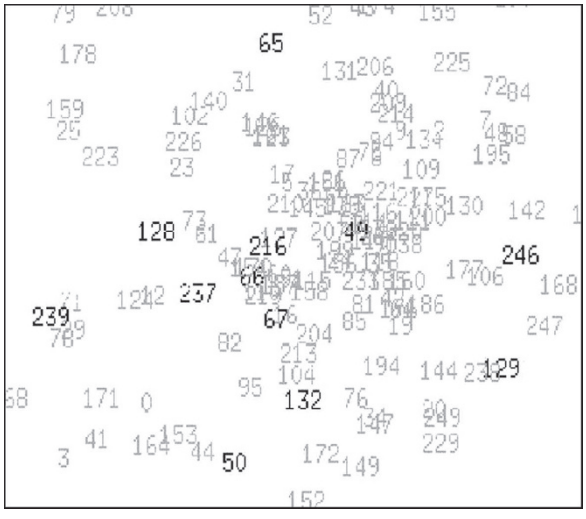
generating visualizations of document collections containing millions of items (Fig. 4).

WEBSOM

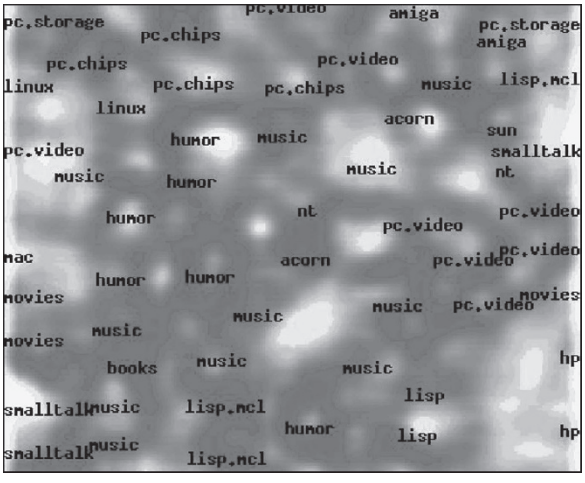
WEBSOM (Kaski et al., 1998) was another early application of Kohonnen self-organizing maps to text data. Early versions of WEBSOM used an independent SOM to generate reduced dimensionality text vectors which were then mapped with a second SOM for visualization purposes. More recent SOM-based text visualization experiments have employed vectors constructed via random projections of weighted word histograms (Kohonen et al., 2000). SOMs have been used to generate semantic maps containing millions of documents (Fig. 5).

Starlight

Starlight (Risch et al., 1999) is a general-purpose information visualization system developed at PNNL that includes a text visualization component. Starlight's text visualization system uses the Boeing *Text Representation Using Subspace Transformation* (TRUST) text engine for vector space modeling and text summarization. Text vectors generated by TRUST are clustered, and the cluster centroids are down-projected to 2D and 3D using a nonlinear manifold learning algorithm. Individual document vectors associated with each cluster are likewise projected within a local coordinate system established at the projected coordinates of their associated cluster centroid, and TRUST is used to generate topical labels for each cluster. Starlight is unique in that

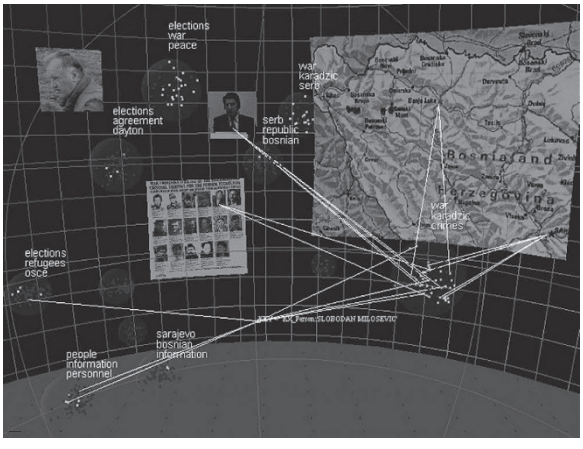


Text Visualization. Figure 3.



Text Visualization. Figure 5.

Text Visualization. Figure 4.



Text Visualization. Figure 6.

it couples text visualization with a range of other information visualization techniques (such as link displays) to depict multiple aspects of information simultaneously (Fig. 6).

Cross References

►Dimensionality Reduction
►Document Classification/Clustering
►Feature Selection/Construction
►Information Extraction/Visualization
►Self-Organizing Maps
►Text Preprocessing

Recommended Reading

Agrafiotis, D. K. (2003). Stochastic proximity embedding. *Journal of Computational Chemistry, 24*(10), 1215–1221.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American, 284*(5), 34–43.

Booker, A., Condliff, M., Greaves, M., Holt, F. B., Kao, A., Pierce, D. J., et al. (1999). Visualizing text data sets. *Computing in Science & Engineering, 1*(4), 26–35.

Chalmers, M., & Chitson, P. (1992). Bead: Explorations in information visualization. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 330–337). New York: ACM.

Chatfield, C., & Collins, A. (1980). *Introduction to multivariate analysis*. London: Chapman & Hall.

Cox, M. F., & Cox, M. A. A. (2001). *Multidimensional scaling*. London: Chapman & Hall.

Crouch, D. (1986). The visual display of information in an information retrieval environment. In *Proceedings of the ACM SIGIR conference on research and development in information retrieval* (pp. 58–67). New York: ACM.

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of American Society for Information Science, 41*(6), 391–407.

de Silva, V., & Tenenbaum, J. B. (2003). Global versus local methods in nonlinear dimensionality reduction. In S. Becker, S. Thrun, & K. Obermayer (Eds.), *Proceedings of the NIPS* (Vol. 15, pp. 721–728).

Doyle, L. (1961). Semantic roadmaps for literature searchers. *Journal of the Association for Computing Machinery, 8*(4), 367–391.

Fabrikant, S. I. (2001). Visualizing region and scale in information spaces. In *Proceedings of the 20th international cartographic conference, ICC 2001* (pp. 2522–2529). Beijing, China.

Havre, S., Hetzler, E., Whitney, P., & Nowell, L. (2002). ThemeRiver: Visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics, 8*(1), 9–20.

Huang, S., Ward, M., & Rundensteiner, E. (2003). *Exploration of dimensionality reduction for text visualization* (Tech. Rep. TR-03-14). Worcester, MA: Worcester Polytechnic Institute, Department of Computer Science.

Kao, A., Poteet, S., Ferng, W., Wu, J., & Quach, L. (2008). Latent semantic indexing and beyond, to appear. In M. Song & Y. F. Wu (Eds.), *Handbook of research on text and web mining technologies*. Hershey, PA: Idea Group Inc.

Kaski, S., Honkela, T., Lagus, K., & Kohonen, T. (1998). WEBSOM-self-organizing maps of document collections. *Neurocomputing, 21*, 101–117.

Kohonen, T. (1997). *Self-organizing maps. Series in information sciences* (2nd ed., Vol. 30). Heidelberg: Springer.

Kohonen, T., Kaski, S., Lagus, K., Salojärvi, J., Honkela, J., Paatero, V., et al. (2000). Self organization of a massive document collection. *IEEE Transactions on Neural Networks, 11*(3), 574–585.

Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika, 29*(1), 1–27.

Lin, X., Soergel, D., & Marchionini, D. A. (1991). Self-organizing semantic map for information retrieval. In *Proceedings of the fourteenth annual international ACM/SIGIR conference on research and development in information retrieval* (pp. 262–269). Chicago.

Mooney, R. J., & Bunescu, R. (2006). Mining knowledge from text using information extraction. In K. Kao & S. Poteet (Eds.), *SigKDD explorations* (pp. 3–10).

Paulovich, F. V., Nonato, L. G., & Minghim, R. (2006). Visual mapping of text collections through a fast high precision projection technique. In *Proceedings of the tenth international conference on information visualisation (IV'06)* (pp. 282–290).

Risch, J. S., Rex, D. B., Dowson, S. T., Walters, T. B., May, R. A., & Moon, B. D. (1999). The STARLIGHT information visualization system. In S. Card, J. Mackinlay, & B. Shneiderman (Eds.), *Readings in information visualization: Using vision to think* (pp. 551–560). San Francisco: Morgan Kaufmann.

Salton, G. (1989). *Automatic text processing*. Reading, MA: Addison-Wesley.

Sammon, J. W. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computer, 18*(5), 401–409.

Small, D. (1996). Navigating large bodies of text. *IBM Systems Journal, 35*(3&4), 514–525.

Wise, J. A., Thomas, J. J., Pennock, K., Lantrip, D., Pottier, M., Schur, A., et al. (1995). Visualizing the non-visual: Spatial analysis and interaction with information from text documents. In *Proceedings of the IEEE information visualization symposium '95* (pp. 51–58). Atlanta, GA.

York, J., Bohn, S., Pennock, K., & Lantrip, D. (1995). Clustering and dimensionality reduction in SPIRE. In *Proceedings, symposium on advanced information processing and analysis, AIPA95*. Tysons Corner, VA.

TF–IDF

TF–IDF (*term frequency–inverse document frequency*) is a term weighting scheme commonly used to represent textual documents as vectors (for purposes of classification, clustering, visualization, retrieval, etc.). Let $T = \{t_1,\ldots, t_n\}$ be the set of all terms occurring in the document corpus under consideration. Then a document

d_i is represented by a n-dimensional real-valued vector $\mathbf{x}_i = (x_{i1},\ldots,x_{in})$ with one component for each possible term from T.

The weight x_{ij} corresponding to term t_j in document d_i is usually a product of three parts: one which depends on the presence or frequency of t_j in d_i, one which depends on t_j's presence in the corpus as a whole, and a normalization part which depends on d_j. The most common TF–IDF weighting is defined by $x_{ij} = \mathrm{TF}_i \cdot \mathrm{IDF}_j \cdot (\sum_j (\mathrm{TF}_{ij}\mathrm{IDF}_j)^2)^{-1/2}$, where TF_{ij} is the *term frequency* (i.e., number of occurrences) of t_j in d_i, and IDF_j is the IDF of t_j, defined as $\log(N/\mathrm{DF}_j)$, where N is the number of documents in the corpus and DF_j is the document frequency of t_j (i.e., the number of documents in which t_j occurs). The normalization part ensures that the vector has a Euclidean length of 1.

Several variations on this weighting scheme are also known. Possible alternatives for TF_{ij} include $\min\{1, \mathrm{TF}_{ij}\}$ (to obtain binary vectors) and $(1 + \mathrm{TF}_{ij}/\max_j \mathrm{TF}_{ij})/2$ (to normalize TF within the document). Possible alternatives for IDF_j include 1 (to obtain plain TF vectors instead of TF–IDF vectors) and $\log(\sum_i \sum_k \mathrm{TF}_{ik}/\sum_i \mathrm{TF}_{ij})$. The normalization part can be omitted altogether or modified to use some other norm than the Euclidean one.

Threshold Phenomena in Learning

►Phase Transitions in Machine Learning

Time Sequence

►Time Series

Time Series

Eamonn Keogh
University of California
Riverside, CA, USA

Synonyms

Temporal data; Time sequence; Trajectory data

Definition

A *Time Series* is a sequence $T = (t_1, t_2,\ldots,t_n)$ which is an ordered set of n real-valued numbers. The ordering is typically temporal; however, other kinds of data such as color distributions (Hafner, Sawhney, Equitz, Flickner, & Niblack, 1995), shapes (Ueno, Xi, Keogh, & Lee, 2006), and spectrographs also have a well-defined ordering and can be fruitfully considered "time series" for the purposes of machine learning algorithms.

Motivation and Background

The special structure of time series produces unique challenges for machine learning researchers.

It is often the case that each individual time series object has a very high dimensionality. Whereas classic algorithms often assume a relatively low dimensionality (for example, a few dozen measurements such as "height, weight, blood sugar," etc.), time series learning algorithms must be able to deal with dimensionalities in hundreds or thousands. The problems created by high-dimensional data are more than mere computation time considerations; the very meaning of normally intuitive terms, such as "similar to" and "cluster forming," become unclear in high-dimensional space. The reason for this is that as dimensionality increases, all objects become essentially equidistant to each other and thus classification and clustering lose their meaning. This surprising result is known as the ►*curse of dimensionality* and has been the subject of extensive research. The key insight that allows meaningful time series machine learning is that although the actual dimensionality may be high, the *intrinsic* dimensionality is typically much lower. For this reason, virtually all time series data mining algorithms avoid operating on the original "raw" data; instead, they consider some higher level representation or abstraction of the data. Such algorithms are known as ►*dimensionality reduction* algorithms. There are many general dimensionality reduction algorithms, such as singular value decomposition and random projections, in addition to many reduction algorithms specifically designed for time series, including piecewise liner approximations, Fourier transforms, wavelets, and symbol approximations (Ding, Trajcevski, Scheuermann, Wang, & Keogh, 2008).

In addition to the high dimensionality of individual time series objects, many time series datasets have very

high numerosity, resulting in a large volume of data. One implication of high numerosity combined with the high dimensionality of this is that the entire dataset may not fit in main memory. This requires an efficient disk-aware learning algorithm or a careful *sampling* approach.

A final consideration due to the special nature of time series is the fact that individual datapoints are typically highly correlated with their neighbors (a phenomenon known as *autocorrelation*). Indeed, it is this correlation that makes most time series excellent candidates for dimensionality reduction. However, for learning algorithms that assume the independence of features (i.e., ►Naïve Bayes), this lack of independence must be countered or mitigated in some way.

While virtually every machine learning method has been used to classify time series, the current state-of-the-art method is the nearest neighbor algorithm (Ueno et al., 2006) with a suitable distance measure (Ding et al., 2008). This simple method outperforms neutral networks and Bayesian classifiers.

The major database (SIGMOD, VLDB, PODS) and data mining (SIGKDD, ICDM, SDM) conferences typically feature several time series machine learning/data mining papers each year. In addition, because of the ubiquity of time series, several other communities have active subgroups that conduct research on time series; for example, the SIGGRAPH conference typically has papers on learning or indexing or motion capture time series, and most medical conferences have tracks devoted to medical time series, such as electrocardiograms and electroencephalograms.

The UCR Time Series Archive has several dozen time series datasets which are widely used to test classification and clustering algorithms, and the UCI Data Mining archive has several additional datasets.

Recommended Reading

Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., & Keogh, E. A. (2008). Querying and mining of time series data: Experimental comparison of representations and distance measures. *In Proceeding of the VLDB*. VLDB Endowment.

Hafner, J., Sawhney, H., Equitz, W., Flickner, M., & Niblack, W. (1995). Efficient color histogram indexing for quadratic form distance functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 17*(7), 729–736.

Ueno, K., Xi, X., Keogh, E., & Lee, D. (2006). Anytime classification using the nearest neighbor algorithm with applications to stream mining. In Proceedings of IEEE international conference on data mining (ICDM).

Topic Mapping

►Text Visualization

Topology of a Neural Network

Risto Miikkulainen
The University of Texas at Austin
Austin, TX, USA

Synonyms

Connectivity; Neural network architecture; Structure

Definition

Topology of a neural network refers to the way the ►Neurons are connected, and it is an important factor in network functioning and learning. A common topology in unsupervised learning is a direct mapping of inputs to a collection of units that represents categories (e.g., ►Self-organizing maps). The most common topology in supervised learning is the fully connected, three-layer, feedforward network (see ►Backpropagation, Radial Basis Function Networks). All input values to the network are connected to all neurons in the hidden layer (hidden because they are not visible in the input or the output), the outputs of the hidden neurons are connected to all neurons in the output layer, and the activations of the output neurons constitute the output of the whole network. Such networks are popular partly because theoretically they are known to be universal function approximators (with e.g., a sigmoid or gaussian nonlinearity in the hidden layer neurons), although in practice networks with more layers may be easier to train (see ►Cascade Correlation, Deep Belief Networks). Layered networks can be extended to processing sequential input and/or output by saving a copy of the hidden layer activations and using it as additional input to the hidden layer in the next time step (see ►Simple Recurrent Networks).

Fully recurrent topologies, where each neuron is connected to all other neurons (and possibly to itself) can also be used to model time-varying behavior, although such networks may be unstable and difficult to train (e.g., with backpropagation; but see also ►Boltzmann Machines). Modular topologies, where different parts of the networks perform distinctly different tasks, can improve stability and can also be used to model high-level behavior (e.g., ►Reservoir Computing, ►Adaptive Resonance Theory). Whatever the topology, in most cases, learning involves modifying the ►Weights on the network connections. However, arbitrary network topologies are possible as well and can be constructed as part of the learning (e.g. with backpropagation or ►Neuroevolution) to enhance feature selection, recurrent memory, abstraction, or generalization.

Trace-Based Programming

Pierre Flener[1,2], Ute Schmid[3]
[1]Sabancı University, Orhanlı, Tuzla, İstanbul, Turkey
[2]Uppsala University
Uppsala, Sweden
[3]University of Bamberg
Bamberg, Germany

Synonyms

Programming from traces; Trace-based programming

Definition

Trace-based programming addresses the inference of a program from a small set of example computation traces. The induced program is typically a recursive program. A computation *trace* is a nonrecursive expression that describes the transformation of some specific input into the desired output with help of a predefined set of primitive functions. While the construction of traces is highly dependent on background knowledge or even on knowledge about the program searched for, the inductive ►generalization is based on syntactical methods of detecting regularities and dependencies between traces, as proposed in classical approaches to ►inductive programming (see Example 5 of that encyclopedia entry) or ►explanation-based learning (EBL). As an alternative to providing traces by hand-simulation, AI planning techniques or ►programming by demonstration (PBD) can be used.

Cross References

►Inductive Programming
►Programming by Demonstration

Recommended Reading

Biermann, A. W. (1972). On the inference of Turing machines from sample computations. *Artificial Intelligence, 3*(3), 181–198.

Schmid, U., & Wysotzki, F. (1998). Induction of recursive program schemes. In *Proceedings of the tenth european conference on machine learning (ECML 1998): Lecture notes in artificial intelligence* (Vol. 1398, pp. 214–225). Berlin: Springer.

Schrödl, S., & Edelkamp, S. (1999). Inferring flow of control in program synthesis by example. In *Proceedings of the 23rd annual german conference on artificial intelligence (KI 1999): Lecture notes in artificial intelligence* (Vol. 1701, pp. 171–182). Berlin: Springer.

Shavlik, J. W. (1990). Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning, 5*, 39–70.

Wysotzki, F. (1983). Representation and induction of infinite concepts and recursive action sequences. In *Proceedings of the eighth international joint conference on artificial intelligence (IJCAI 1983)* (pp. 409–414). San Mateo, CA: Morgan Kaufmann.

Training Curve

►Learning Curves in Machine Learning

Training Data

Synonyms

Training examples; Training instances; Training set

Definition

Training data are data to which a ►learner is applied.

Training Examples

►Training Data

Training Instances

►Training Data

Training Set

Synonyms
Training data

Definition
A training set is a ►data set containing data that are used for learning by a ►learning system. A training set may be divided further into a ►growing set and a ►pruning set.

Cross References
►Data Set
►Training Data

Training Time

A learning algorithm is typically applied at two distinct times. Training time refers to the time when an algorithm is learning a model from ►training data. ►Test time refers to the time when an algorithm is applying a learned model to make predictions. ►Lazy learning usually blurs the distinction between these two times, deferring most learning until test time.

Trait

►Attribute

Trajectory Data

►Time Series

Transductive Learning

►Semi-Supervised Learning
►Semi-Supervised Text Processing

Transfer of Knowledge Across Domains

►Inductive Transfer

Transition Probabilities

In a ►Markov decision process, the *transition probabilities* represent the probability of being in *state* s' at time $t+1$, given you take *action* a from state s at time t for all s, a and t.

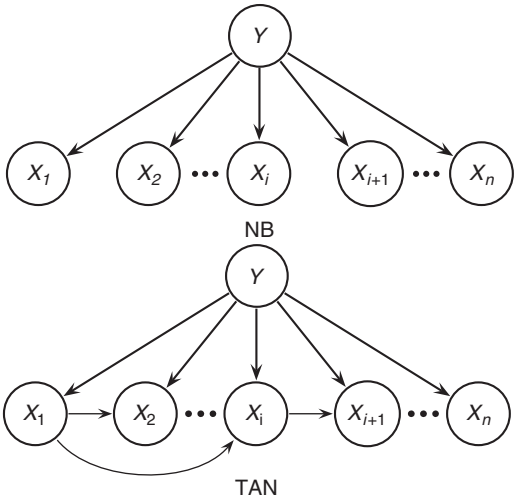
Tree Augmented Naive Bayes

Fei Zheng, Geoffrey I. Webb
Monash University

Synonyms
TAN

Definition
Tree augmented naive Bayes is a ►semi-naive Bayesian Learning method. It relaxes the ►naive Bayes attribute independence assumption by employing a tree structure, in which each attribute only depends on the class and one other attribute. A maximum weighted spanning tree that maximizes the likelihood of the training data is used to perform classification.

Classification with TAN
Interdependencies between attributes can be addressed directly by allowing an attribute to depend on other non-class attributes. However, techniques for learning unrestricted ►Bayesian networks often fail to deliver lower zero-one loss than naive Bayes (Friedman, Geiger, & Goldszmidt, 1997). One possible reason for this is that full Bayesian networks are oriented toward optimizing the likelihood of the training data rather than the conditional likelihood of the class attribute given a full set of other attributes. Another possible reason is that full Bayesian networks have high variance due



Tree Augmented Naive Bayes. Figure 1. Bayesian network examples of the forms of model created by NB and TAN

to the large number of parameters estimated. An intermediate alternative technique is to use a less restrict structure than naive Bayes. Tree augmented naive Bayes (TAN) (Friedman et al., 1997) employs a tree structure, allowing each attribute to depend on the class and at most one other attribute. Figure 1 shows Bayesian network representations of the types of model that NB and TAN respectively create.

Chow and Liu (1968) proposed a method that efficiently constructs a maximum weighted spanning tree which maximizes the likelihood that the training data was generated from the tree. The weight of an edge in the tree is the mutual information of the two attributes connected by the edge. TAN extends this method by using conditional mutual information as weights. Since the selection of root does not affect the log-likelihood of the tree, TAN randomly selects a root attribute and directs all edges away from it. The parent of each attribute X_i is indicated as $\pi(X_i)$ and the parent of the class is $\varnothing$. It assumes that attributes are independent given the class and their parents and classifies the test instance $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ by selecting

$$\underset{y}{\operatorname{argmax}} \hat{P}(y) \prod_{1 \le i \le n} \hat{P}(x_i \mid y, \pi(x_i)), \qquad (1)$$

where $\pi(x_i)$ is a value of $\pi(X_i)$ and y is a class label.

Due to the relaxed attribute independence assumption, TAN considerably reduces the ▶bias of naive Bayes at the cost of an increase in ▶variance. Empirical results (Friedman et al., 1997) show that it substantially reduces zero-one loss of naive Bayes on many data sets and that of all data sets examined it achieves lower zero-one loss than naive Bayes more often than not.

Cross References

▶Averaged One-Dependence Estimators
▶Bayesian Network
▶Naive Bayes
▶Semi-Naive Bayesian Learning

Recommended Reading

Chow, C. K., & Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, 14*, 462–467.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning, 29*(2), 131–163.

Tree Mining

Siegfried Nijssen
Katholieke Universiteit Leuven,
Belgium

Definition

Tree mining is an instance of constraint-based pattern mining and studies the discovery of tree patterns in data that is represented as a tree structure or as a set of trees structures. Minimum frequency is the most studied constraint.

Motivation and Background

Tree mining is motivated by the availability of many types of data that can be represented as tree structures. There is a large variety in tree types, for instance, ordered trees, unordered trees, rooted trees, unrooted (free) trees, labeled trees, unlabeled trees, and binary trees; each of these has its own application areas. An example are trees in tree banks, which store sentences annotated with parse trees. In such data, it is

not only of interest to find commonly occurring sets of words (for which frequent itemset miners could be used), but also to find commonly occurring parses of these words. Tree miners aim at finding patterns in this structured information. The patterns can be interesting in their own right, or can be used as features in classification algorithms.

Structure of Problem

All tree miners share a similar problem setting. Their input consists of a set of trees and a set of constraints, usually a minimum frequency constraint, and their output consists of all subtrees that fulfill the constraints.

Tree miners differ in the constraints that they are able to deal with, and the types of trees that they operate on. The following types of trees can be distinguished:

Free trees, which are graphs without cycles, and no order on the nodes or edges;

Unordered trees, which are free trees in which one node is chosen to be the root of the tree;

Ordered trees, which are rooted trees in which the nodes are totally ordered.

For each of these types of tree, we can choose to have labels on the nodes, or on the edges, or on both.

The differences between these types of trees are illustrated in Fig. 1. Every graph in this figure can be interpreted as a free tree F_i, an unordered tree U_i, or an ordered tree T_i. When interpreted as ordered trees, none of the trees are equivalent. When we interpret them as unordered trees, U_1 and U_2 are equivalent representations of the same unordered tree that has B as its root and C and D as its children. Finally, as free trees, not only F_1 and F_2 are equivalent, but also F_5 and F_7.

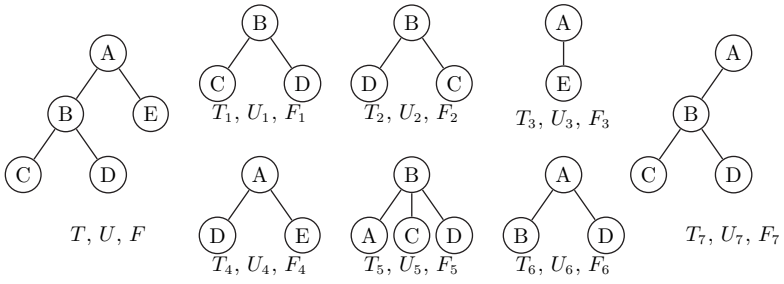
Intuitively, a free tree requires less specification than an ordered tree. The number of possible free trees is smaller than the number of possible ordered trees. On the other hand, to test if two trees are equivalent we need a more elaborate computation for free trees than for ordered trees.

Assume that we have data represented as (a set of) trees, then the data mining problem is to find patterns, represented as trees, that fulfill constraints based on this data. To express these constraints, we need a coverage relation that expresses when one tree can be considered to occur in another tree. Different coverage relations can be expressed for free trees, ordered trees, and unordered trees. We will introduce these relations through operations that can be used to transform trees. As an example, consider the operation that removes a leaf from a tree. We can repeatedly apply this operation to turn a large tree into a smaller one. Given two trees A and B, we say that A occurs in B as

Induced subtree, if A can be obtained from B by repeatedly removing leaves from B. When dealing with rooted trees, the root is here also considered to be a leaf if it has one child;

Root-induced subtree, if A can be obtained from B by repeatedly removing leaves from B. When dealing with rooted trees, the root is not allowed to be removed;

Embedded subtree, if A can be obtained from B by repeatedly either (1) removing a leaf or (2) removing an internal node, reconnecting the children of



Tree Mining. Figure 1. The leftmost tree is part of the data, the other trees could be patterns in this tree, depending on the subtree relation that is used

the removed node with the parent of the removed node;

Bottom-up subtree, if there is a node v in B such that if we remove all nodes from B that are not a descendant of v, we obtain A;

Prefix, if A can be obtained from B by repeatedly removing the last node from the ordered tree B;

Leaf set, if A can be obtained from B by selecting a set of leaves from B, and all their ancestors in B.

For free trees, only the induced subtree relation is well-defined. A prefix is only well-defined for ordered trees, the other relations apply both to ordered and unordered trees. In the case of unordered trees, we assume that each operation maintains the order of the original tree B. The relations are also illustrated in Fig. 2.

Intuitively, we can speak of *occurrences* (also called *embeddings* by some authors) of a small tree in a larger tree. Each such occurrence (or embedding) can be thought of as a function φ that maps every node in the small tree to a node in the large tree.

Using an occurrence relation, we can define frequency measures. Assume given a forest $\mathcal{F}$ of trees, all ordered, unordered, or free. Then the frequency of a tree A can be defined

Transaction-based, where we count the number of trees $B \in \mathcal{F}$ such that A is a subtree of B;

Node-based, where we count the number of nodes v in $\mathcal{F}$ such that A is a subtree of the bottom-up subtree below v.

Node-based frequency is only applicable in rooted trees, in combination with the root-induced, bottom-up, prefix, or leaf set subtree relations.

Given a definition of frequency, constraints on trees of interest can be expressed:

Minimum frequency, to specify that only trees with a certain minimum number of occurrences are of interest;

Closedness, to specify that a tree is only of interest if its frequency is different from all its supertrees;

Maximality, to specify that a tree is only of interest if none of its supertrees is frequent.

Observe that in all of these constraints, the subtree relation is again important. The subtree relation is not only used to compare patterns with data, but also patterns among themselves.

The tree mining problem can now be stated as follows. Given a forest of trees $\mathcal{F}$ (ordered, unordered, or free) and a set of constraints, based on a subtree relation, the task is to find all trees that satisfy the given constraints.

Tree	Induced	Embedded	Root-Induced	Bottom-up	Prefix	Leaf-set
T_1	yes	yes	no	yes	no	no
T_2	no	no	no	no	no	no
T_3	yes	yes	yes	no	no	yes
T_4	no	yes	no	no	no	no
T_5	no	no	no	no	no	no
T_6	no	no	no	no	no	no
T_7	yes	yes	yes	no	yes	yes

Tree	Induced	Embedded	Root-Induced	Bottom-up	Leaf-set
U_1	yes	yes	no	yes	no
U_2	yes	yes	no	yes	no
U_3	yes	yes	yes	no	yes
U_4	no	yes	no	no	no
U_5	no	no	no	no	no
U_6	no	no	no	no	no
U_7	yes	yes	yes	no	yes

Tree	Induced
F_1	yes
F_2	yes
F_3	yes
F_4	no
F_5	yes
F_6	no
F_7	yes

Tree Mining. Figure 2. Relations between the trees of Fig. 1

Theory/Solution

The tree mining problem is an instance of the more general problem of constraint-based pattern mining under constraints. For more information about the general setting, see the sections on constraint-based mining, itemset mining, and graph mining.

All algorithms iterate a process of generating candidate patterns, and testing if these candidates satisfy the constraints. Essential is to avoid that every possible tree is considered as a candidate. To this purpose, the algorithms exploit that many frequency measures are anti-monotonic. This property states that for two given

trees A and B, where A is a subtree of B, if A is infrequent, then also B is infrequent, and therefore, we do not need to consider it as a candidate.

This observation can make it possible to find all trees that satisfy the constraints, if these requirements are fulfilled:

- We have an algorithm to enumerate candidate subtrees, which satisfies these properties:
  - It should be able to enumerate all trees in the search space;
  - It should avoid that no two equivalent subtrees are listed;
  - It should only list a tree after at least one of its subtrees has been listed, to exploit the anti-monotonicity of the frequency constraint;
- We have an algorithm to efficiently compute in how many database trees a pattern tree occurs.

The algorithmic solutions to these problems depend on the type of tree and the subtree relation.

Encoding and Enumerating Trees

We will first consider how tree miners internally represent trees. Two types of encodings have been proposed, both of which are string-based. We will illustrate these encodings for node-labeled trees, and start with *ordered* trees.

The first encoding is based on a *preorder* listing of nodes: (1) for a rooted ordered tree T with a single vertex r, the *preorder string* of T is $S_{T,r} = l_r - 1$, where l_r is the label for the single vertex r, and (2) for a rooted ordered tree T with more than one vertex, assuming the root of T is r (with label l_r) and the children of r are $r_1, \ldots, r_K$ from left to right, then the preorder string for T is $S_{T,r} = l_r S_{T,r_1} \cdots S_{T,r_K} - 1$, where $S_{T,r_1}, \ldots, S_{T,r_K}$ are the preorder strings for the bottom-up subtrees below nodes $r_1, \ldots, r_K$ in T.

The second encoding is based on listing the *depths* of the nodes together with their labels in prefix-order. The depth of a node v is the length of the path from the root to the node v. The code for a tree is $S_{T,r} = d_r l_r S_{T,r_1} \cdots S_{T,r_K}$, where d_r is the depth of the node r in tree T.

Both encodings are illustrated in Fig. 3.

Tree	Depth-sequence	Preorder string
T_6	1*A*2*B*2*D*	*AB*-1*D*-1
T_7	1*A*2*B*3*C*3*D*	*ABC*-1*D*-1-1-1
T	1*A*2*B*3*C*3*D*2*E*	*ABC*-1*D*-1-1*E*-1
T_4	1*A*2*D*2*E*	*AD*-1*E*-1-1
T_3	1*A*2*E*	*AE*-1-1
T_5	1*B*2*A*2*C*2*D*	*BA*-1*C*-1*D*-1-1
T_1	1*B*2*C*2*D*	*BC*-1*D*-1-1
T_2	1*B*2*D*2*C*	*BD*-1*C*-1-1

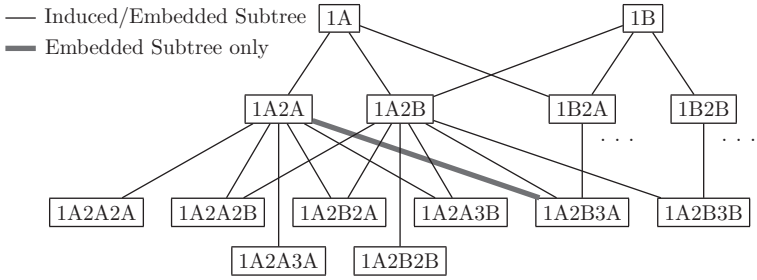
Tree Mining. Figure 3. Depth sequences for all the trees of Fig. 1, sorted in lexicographical order. Tree T_2 is the canonical form of unordered tree U_2, as its depth sequence is the highest among equivalent representations

A search space of trees can be visualized as in Fig. 4. In this figure, every node corresponds to the depth encoding of a tree, while the edges visualize the partial order defined by the subtree relation. It can be seen that the number of induced subtree relations between trees is smaller than the number of embedded subtree relations.

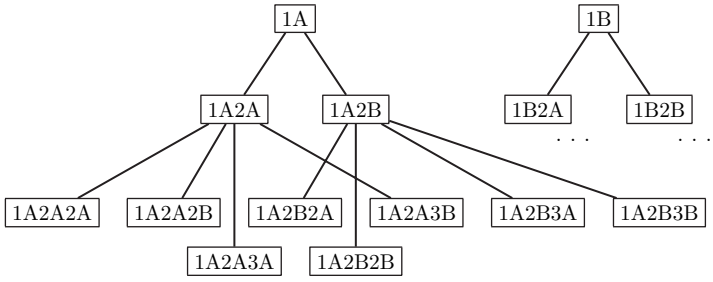
The task of the enumeration algorithm is to traverse this search space starting from trees that contain only one node. Most algorithms perform the search by building an enumeration tree over the search space. In this enumeration tree every pattern should have a single parent. The children of a pattern in the enumeration tree are called its *extensions* or its *refinements*. An example of an enumeration tree for the induced subtree relation is given in Fig. 5.

In the enumeration tree that is given here, the parent of a tree is its prefix in the depth encoding. An alternative definition is that the parent of a tree can be obtained by removing the last node in a prefix order traversal of the ordered tree. Every refinement in the enumeration has one additional node that is connected to the *rightmost path* of the parent.

The enumeration problem is more complicated for *unordered trees*. In this case, the trees represented by the strings 1*A*2*A*2*B* and 1*A*2*B*2*A* are equivalent, and we



Tree Mining. Figure 4. A search space of ordered trees, where edges denote subtree relationships



Tree Mining. Figure 5. Part of an enumeration tree for the search space of Fig. 4

only wish to enumerate one of these strings. This can be achieved by defining a total order on all strings that represent trees, and to define that only the highest (or lowest) string of a set of equivalent strings should be considered.

For depth encodings, the ordering is usually lexicographical, and the highest string is chosen to be the *canonical* encoding. In our example, 1*A*2*B*2*A* would be canonical. This code has the desirable property that every prefix of a canonical code is also a canonical code. Furthermore it can be determined in polynomial time which extensions of a canonical code lead to a canonical code, such that it is not necessary to consider any code that is not canonical.

Alternative codes have also been proposed, which are not based on a preorder, depth-first traversal of a tree, but on a level-wise listing of the nodes in a tree.

Finally, for *free trees* we have the additional problem that we de not have a root for the tree. Fortunately, it is known that every free tree either has a uniquely determined *center* or a uniquely determined *bicenter*. This (bi)center can be found by determining the longest path between two nodes in a free tree: the node(s) in the middle of this path are the center of the tree. It can be shown that if multiple paths have the same maximal length, they will have the same (bi)center. By appointing one center to be the root, we obtain a rooted tree, for which we can compute a code.

To avoid that two codes are listed that represent equivalent free trees, several solutions have been proposed. One is based on the idea of first enumerating paths (thus fixing the center of a tree), and for each of these paths enumerating all trees that can be grown around them. Another solution is based on enumerating all rooted, unordered trees under the constraint that at least two different children of the root have a bottom-up subtree of equal, maximal depth. In the first approach, a preorder depth encoding was used; in the second approach a level-wise encoding was used.

Counting Trees

To evaluate the frequency of a tree the subtree relation between a candidate pattern tree and all trees in the database has to be computed. For each of our subtree relations, polynomial algorithms are known to decide the relation, which are summarized in Table 1.

Tree Mining. Table 1 Worst case complexities of the best known algorithms that determine whether a tree relation holds between two trees; *m* is the number of nodes in the pattern tree, *l* is the number of leafs in the pattern tree, *n* the number of nodes in the database tree

	Ordered	Unordered
Embedding	$O(nl)$	NP-complete
Induced	$O(nm)$	$O(nm^{1\frac{1}{2}}/\log m)$
Root-induced	$O(n)$	$O(nm^{1\frac{1}{2}}/\log m)$
Leaf-set	$O(n)$	$O(nm^{1\frac{1}{2}}/\log m)$
Bottom-up	$O(n)$	$O(n)$
Prefix	$O(m)$	

Even though a subtree testing algorithm and an algorithm for enumerating trees are sufficient to compute all frequent subtrees correctly, in practice fine-tuning is needed to obtain an efficient method. There are two reasons for this:

- In some databases, the number of candidates can by far exceed the number of trees that are actually frequent. One way to reduce the number of candidates is to only generate a particular candidate after we have encountered at least one occurrence of it in the data (this is called *pattern growth*); another way is to require that a candidate is only generated if at least two of its subtrees satisfy the constraints (this is called *pattern joining*).
- The trees in the search space are very similar to each other: a parent only differs from its children by the absence of a single node. If memory allows, it is desirable to *reuse* the subtree matching information, instead of starting the matching from scratch.

A large number of data structures have been proposed to exploit these observations. We will illustrate these ideas using the FreqT algorithm, which mines induced, ordered subtrees, and uses a depth encoding for the trees.

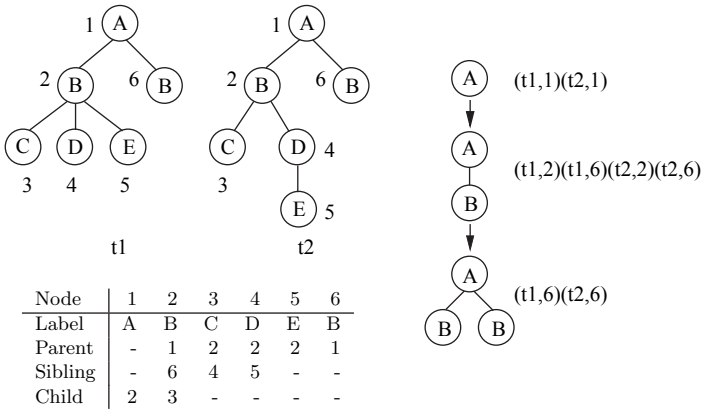
In FreqT, for a given pattern tree A, a list of (database tree, database node) pointers is stored. Every element (B, v) in this list corresponds to an occurrence of tree A in tree B in which the last node (in terms of the preorder) of A is mapped to node v in database tree B. For a database and three example trees this is illustrated in Fig. 6.

Every tree in the database is stored as follows. Every node is given an index, and for every node, we store the index of its parent, its righthand sibling, and its first child.

Let us consider how we can compute the occurrences of the subtree 1*A*2*B*2*B* from the occurrences of the tree 1*A*2*B*. The first occurrence of 1*A*2*B* is $(t1, 2)$, which means that the *B* labeled node can be mapped to node 2 in $t1$. Using the arrays that store the database tree, we can then conclude that node 6, which is the right-hand sibling of node 2, corresponds to an occurence of the subtree 1*A*2*B*2*B*. Therefore, we add $(t1, 6)$ to the occurrence list of 1*A*2*B*2*B*. Similarly, by scanning the data we find out that the first child of node 2 corresponds to an occurrence of the subree 1*A*2*B*3*C*, and we add $(t1, 3)$ to the occurrence list of 1*A*2*B*3*C*.

Overall, using the parent, sibling and child pointers we can scan every node in the data that could correspond to a valid expansion of the subtree 1*A*2*B*, and update the corresponding lists. After we have done this for every occurrence of the subtree, we know the occurrence lists of all possible extensions.

From an occurrence list we can determine the frequency of a tree. For instance, the transaction-based frequency can be computed by counting the number of different database trees occurring in the list.



Node	1	2	3	4	5	6
Label	A	B	C	D	E	B
Parent	-	1	2	2	2	1
Sibling	-	6	4	5	-	-
Child	2	3	-	-	-	-

Tree Mining. Figure 6. A tree database (left) and three ordered trees with their occurrence lists according to the FreqT algorithm (right). The datastructure that stores *t*1 in FreqT is given in the table (right)

As we claimed, this example illustrates two features that are commonly seen in tree miners: first, the occurrence list of one tree is used to compute the occurrence list of another tree, thus reusing information; second, the candidates are collected from the data by scanning the nodes that connect to the occurrence of a tree in the data. Furthermore, this example illustrates that a careful design of the datastructure that stores the data can ease the frequency evaluation considerably.

FreqT does not perform pattern joining. The most well-known example of an algorithm that performs tree joining is the embedded TreeMiner (Zaki, 2002). Both the FreqT and the TreeMiner perform the search depth-first, but also tree miners that use the traditional level-wise approach of the Apriori algorithm have been proposed. The FreqT and the TreeMiner have been extended to unordered trees.

Other Constraints

As the number of frequent subtrees can be very large, approaches have been studied to reduce the number of trees returned by the algorithm, of which closed and maximal trees are the most popular. To find closed or maximal trees, two issues need to be addressed:

- How do we make sure that we only output a tree if it is closed or maximal, that is, how do we determine that none of its supertrees has the same support, or is frequent?
- Can we conclude that some parts of the search space will never contain a closed or maximal tree, thus making the search more efficient?

Two approaches can be used to address the first issue:

- All closed patterns can be stored, and every new pattern can be compared with the stored set of patterns;
- When we evaluate the frequency of a pattern in the data, we also (re)evaluate the frequency of all its possible extensions, and only output the pattern if its support is different.

The second approach requires less memory, but in some cases requires more computations.

To prune the search space, a common approach is to check all occurrences of a tree in the data. If every occurrence of a tree can be extended into an occurrence of another tree, the small tree should not be considered, and the search should continue with the tree that contains all common edges and nodes. Contrary to graph mining, it can be shown that this kind of pruning can safely be done in most cases.

Applications

Examples of databases to which tree mining algorithms have been applied are

Parse tree analysis: Since the early 1990s large *Treebank* datasets have been collected consisting of

sentences and their grammatical structure. An example is the Penn TreeBank (Marcus, Santorini, & Marcinkiewicz, 1993). These databases contain rooted, ordered trees. To discover differences in domain languages it is useful to compare commonly occurring grammatical constructions in two different sets of parsed texts, for which tree miners can be used (Sekine, 1998).

Computer network analysis: IP *multicast* is a protocol for sending data to multiple receivers. In an IP multicast session a webserver sends a packet once; routers copy a packet if two different routes are required to reach multiple receivers. During a multicast session rooted trees are obtained in which the root is the sender and the leaves are the receivers. Commonly occurring patterns in the routing data can be discovered by analyzing these unordered rooted trees (Chalmers & Almeroth, 2003).

Webserver access log analysis: When users browse a website, this behavior is reflected in the access log files of the webserver. Servers collect information such as the webpage that was visited, the time of the visit, and the webpage that was clicked to reach the webpage. The access logs can be transformed into a set of ordered trees, each of which corresponds to a visitor. Nodes in these trees correspond to webpages; edges are inserted if a user browses from one webpage to another. Nodes are ordered in viewing order. A tool was developed to perform this transformation in a sensible way (Punin, Krishnamoorthy, & Zaki, 2002).

Phylogenetic trees: One of the largest tree databases currently under construction is the TreeBASE database, which is comprised of a large number of phylogenetic trees (Morell, 1996). The trees in the TreeBASE database are submitted by researchers and are collected from publications. Originating from multiple sources, they can disagree on parts of the phylogenetic tree. To find common agreements between the trees, tree miners have been used (Zhang & Wang, 2005). The phylogenetic trees are typically unordered; labels among siblings are unique.

Hypergraph mining: Hypergraphs are graphs in which one edge can have more than two endpoints. Those hypergraphs in which no two nodes share the same label can be transformed into unordered trees, as follows. First, an artificial root is inserted. Second, for each edge of the hypergraph a child node is added to the root, labeled with the label of the hyperedge. Finally, the labels of nodes within hyperedges are added as leaves to the tree. An example of hypergraph data is bibliographic data: if each example corresponds to a paper, nodes in the hypergraph correspond to authors cited by the paper, and hyperedges connect coauthors of cited papers.

Multi-relational data mining: Many multi-relational databases are tree shaped, or a tree-shaped view can be created. For instance, a transaction database in which every transaction is associated with customers and their information, can be represented as a tree (Berka, 1999).

XML data mining: Several authors have stressed that tree mining algorithms are most suitable for mining XML data. XML is a tree–shaped data format, and tree miners can be helpful when trying to (re)construct Document Type Definitions (DTDs) for such documents.

Cross References

►Constraint-based Mining
►Graph Mining

Further Reading

The FreqT algorithm was introduced in (Asai, Abe, Kawasoe, Arimura, Satamoto, & Arikawa, 2002; Wang & Liu, 1998; Zaki, 2002). The most popular tree miner is the embedded tree miner by Zaki (2002). A more detailed overview of tree miners can be found in Chi, Nijssen, Muntz, and Kok (2005). Most implementations of tree miners are available on request from their authors.

Recommended Reading

Asai, T., Abe, K., Kawasoe, S., Arimura, H., Satamoto, H., & Arikawa, S. (2002). Efficient substructure discovery from large semi-structured data. In *Proceedings of the second SIAM international conference on data mining* (pp. 158–174). SIAM.

Berka, P. (1999). *Workshop notes on discovery challenge PKDD-99* (Tech. Rep.). Prague, Czech Republic: University of Economics.

Chalmers, R., & Almeroth, K. (2003). On the topology of multicast trees. In *IEEE/ACM transactions on networking* (Vol. 11, pp. 153–165). IEEE Press/ACM Press.

Chi, Y., Nijssen, S., Muntz, R. R., & Kok, J. N. (2005). Frequent subtree mining—An overview. In *Fundamenta Informaticae* (Vol. 66, pp. 161–198). IOS Press.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. In *Computational linguistics* (Vol. 19, pp. 313–330). MIT Press.

Morell, V. (1996). TreeBASE: The roots of phylogeny. In *Science* (Vol. 273, p. 569).

Punin, J., Krishnamoorthy, M., & Zaki, M. J. (2002). LOGML—log markup language for web usage mining. In *WEBKDD 2001—mining web log data across all customers touch points. Third international workshop. Lecture notes in artificial intelligence* (Vol. 2356, pp. 88–112). Springer.

Sekine, S. (1998). *Corpus-based parsing and sublanguages studies*. Ph.D. dissertation. New York University, New York.

Wang, K., & Liu, H. (1998). Discovering typical structures of documents: A road map approach. In *Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 146–154). ACM Press.

Zaki, M. J. (2002). Efficiently mining frequent trees in a forest. In *Proceedings of the 8th international conference knowledge discovery and data mining* (*KDD*) (pp. 71–80). ACM Press.

Zhang, S., & Wang, J. (2005). Frequent agreement subtree mining. http://aria.njit.edu/mediadb/fast/.

Tree-Based Regression

►Regression Trees

True Negative

True negatives are the negative examples that are correctly classified by a classification model. See ►confusion matrix for a complete range of related terms.

True Negative Rate

►Specificity

True Positive

True positives are the positive examples that are correctly classified by a classification model. See ►confusion matrix for a complete range of related terms.

True Positive Rate

►Sensitivity

Type

►Class

Typical Complexity of Learning

►Phase Transitions in Machine Learning

U

Underlying Objective

The term *objective* used in Evolutionary Multi-Objective Optimization refers to an indicator of quality returning an element from an ordered set of scalar values, such as a real number. For any test-based coevolutionary problem, a set of underlying objectives exists such that knowledge of the objective values of an individual is sufficient to determine the outcomes of all possible tests. The existence of a set of underlying objectives is guaranteed, as the set of possible tests itself satisfies this property.

Unit

►Neuron

Universal Learning Theory

Marcus Hutter
Australian National University, Canberra, Australia

Definition, Motivation, and Background

Universal (machine) learning is concerned with the development and study of algorithms that are able to learn from data in a very large range of environments with as few assumptions as possible. The class of environments typically considered includes all computable stochastic processes. The investigated learning tasks range from ►inductive inference, sequence prediction, sequential decisions, to (re)active problems like ►reinforcement learning (Hutter, 2005), but also include ►clustering, ►regression, and others (Li & Vitányi, 2008). Despite various no-free-lunch theorems (Wolpert & Macready, 1997), universal learning is *possible* by assuming that the data possess *some* effective structure, but without specifying any further, *which* structure. Learning algorithms that are universal (at least to some degree) are also *necessary* for developing autonomous general intelligent systems, required, for example, for exploring other planets, as opposed to decision *support* systems which keep a human in the loop. There is also an *intrinsic* interest in striving for generality: Finding new learning algorithms for every particular (new) problem is possible but cumbersome and prone to disagreement or contradiction. A sound, formal, general and ideally complete theory of learning can unify existing approaches, guide the development of practical learning algorithms, and last but not least lead to novel and deep insights.

Deterministic Environments

Let $t, n \in \mathbb{N}$ be natural numbers, $\mathcal{X}^*$ be the set of finite strings and $\mathcal{X}^\infty$ be the set of infinite sequences over some alphabet $\mathcal{X}$ of size $|\mathcal{X}|$. For a string $x \in \mathcal{X}^*$ of length $\ell(x) = n$ we write $x_1 x_2 \ldots x_n$ with $x_t \in \mathcal{X}$, and further abbreviate $x_{t:n} := x_t x_{t+1} \ldots x_{n-1} x_n$ and $x_{<n} := x_1 \ldots x_{n-1}$, and $\epsilon = x_{<1}$ for the empty string. Consider a countable class of hypotheses $\mathcal{M} = \{H_1, H_2, \ldots\}$. Each hypothesis $H \in \mathcal{M}$ (also called model) shall describe an infinite sequence $x^H_{1:\infty}$, for example, like in IQ test questions "2, 4, 6, 8, …." In online learning, for $t = 1, 2, 3, \ldots$, we predict x_t based on past observations $\dot{x}_{<t}$, then nature reveals $\dot{x}_t$, and so on, where the dot above x indicates the true observation. We assume that the true hypothesis is in $\mathcal{M}$, that is, $\dot{x}_{1:\infty} = x^{H_m}_{1:\infty}$ for some $m \in \mathbb{N}$. Goal is to ("quickly") identify the unknown H_m from the observations.

Learning by enumeration works as follows: Let $\mathcal{M}_t = \left\{H \in \mathcal{M} : x^H_{<t} = \dot{x}_{<t}\right\}$ be the set of hypotheses consistent with our observations $\dot{x}_{<t}$ so far. The

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

hypothesis in $\mathcal{M}_t$ with smallest index, say m'_t, is selected and used for predicting x_t. Then x_t is observed and all $H \in \mathcal{M}_t$ inconsistent with x_t are eliminated, that is, they are not included in $\mathcal{M}_{t+1}$. Every prediction error results in the elimination of at least $H_{m'_t}$, so after at most $m-1$ errors, the true hypothesis H_m gets selected forever, since it never makes an error ($H_m \in \mathcal{M}_t \; \forall t$). This identification may take arbitrarily long (in t), but the number of errors on the way is bounded by $m-1$, and the latter is often more important. As an example for which the bound is attained, consider H_i with $x^{H_i}_{1:\infty} := 1^{f(i)}0^\infty \; \forall i$ for any strictly increasing function f, for example, $f(i) = i$. But we now show that we can do much better than this, at least for finite $\mathcal{X}$.

Majority learning:

Consider (temporarily in this paragraph only) a binary alphabet $\mathcal{X} = \{0,1\}$ and a *finite* deterministic hypothesis class $\mathcal{M} = \{H_1, H_2, \ldots, H_N\}$. H_m and $\mathcal{M}_t$ are as before, but now we take a majority vote among the hypotheses in $\mathcal{M}_t$ as our prediction of x_t. If the prediction turns out to be wrong, then at least half (the majority) of the hypotheses get eliminated from $\mathcal{M}_t$. Hence after at most $\log N$ errors, there is only a single hypothesis, namely H_m, left over. So this majority predictor makes at most $\log N$ errors. As an example where this bound is essentially attained, consider $m = N = 2^n - 1$ and let $x^{H_i}_{1:\infty}$ be the digits after the comma of the binary expansion of $(i-1)/2^n$ for $i = 1, \ldots, N$.

Weighted majority for countable classes:

Majority learning can be adapted to denumerable classes $\mathcal{M}$ and general finite alphabet $\mathcal{X}$ as follows: Each hypothesis H_i is assigned a weight $w_i > 0$ with $\sum_i w_i \leq 1$. Let $W := \sum_{i:H_i \in \mathcal{M}_t} w_i$ be the total weight of the hypotheses in $\mathcal{M}_t$. Let $\mathcal{M}^a_t := \{H_i \in \mathcal{M}_t : x^{H_i}_t = a\}$ be the consistent hypotheses predicting $x_t = a$, and W_a their weight, and take the weighted majority prediction $x_t = \arg\max_a W_a$. Similarly as above, a prediction error decreases W by a factor of $1 - 1/|\mathcal{X}|$, since $\max_a W_a \geq W/|\mathcal{X}|$. Since $w_m \leq W \leq 1$, this algorithm can at most make $\log_{1-1/|\mathcal{X}|} w_m = O\left(\log w_m^{-1}\right)$ prediction errors. If we choose, for instance, $w_i = (i+1)^{-2}$, the number of errors is $O(\log m)$, which is an exponential improvement over the Gold-style learning by enumeration above.

Algorithmic Probability

Algorithmic probability has been founded by Solomonoff (1964). The so-called universal probability or a-priori probability is the key quantity for universal learning. Its philosophical and technical roots are ►Ockham's razor (choose the simplest model consistent with the data), Epicurus' principle of multiple explanations (keep all explanations consistent with the data), (Universal) Turing machines (to compute, quantify and assign codes to all quantities of interest), and Kolmogorov complexity (to define what simplicity/complexity means). This section considers deterministic computable sequences, and the next section the general setup of computable probability distributions.

(Universal) monotone Turing machines: Since we consider infinite computable sequences, we need devices that convert input data streams to output data streams. For this we define the following variants of a classical deterministic Turing machine: A monotone Turing machine T is defined as a Turing machine with one unidirectional input tape, one unidirectional output tape, and some bidirectional work tapes. The input tape is binary (no blank) and read only; the output tape is over finite alphabet $\mathcal{X}$ (no blank) and write only; unidirectional tapes are those where the head can only move from left to right; work tapes are initially filled with zeros and the output tape with some fixed element from $\mathcal{X}$. We *say* that *monotone Turing machine* T outputs/computes a string starting with x on input p, and write $T(p) = x*$ if p is to the left of the input head when the last bit of x is output (T reads all of p but no more). T may continue operation and need not halt. For a given x, the set of such p forms a prefix code. Such codes are called *minimal* programs. Similarly, we write $T(p) = \omega$ if p outputs the infinite sequence ω. A *prefix code* $\mathcal{P}$ is a set of binary strings such that no element is proper prefix of another. It satisfies *Kraft's inequality* $\sum_{p \in \mathcal{P}} 2^{-\ell(p)} \leq 1$.

The table of rules of a Turing machine T can be prefix encoded in a canonical way as a binary string, denoted by $\langle T \rangle$. Hence, the set of Turing machines $\{T_1, T_2, \ldots\}$ can be effectively enumerated. There are so-called universal Turing machines that can "simulate" all other Turing machines. We define a particular one which simulates monotone Turing machine $T(q)$ if fed

with input $\langle T\rangle q$, that is, $U(\langle T\rangle q) = T(q)\ \forall T, q$. Note that for p not of the form $\langle T\rangle q$, $U(p)$ does not output anything. We call this particular U the *reference universal Turing machine*.

Universal weighted majority learning: $T_1(\epsilon), T_2(\epsilon), \ldots$ constitutes an effective enumeration of all finite and infinite computable sequences, hence also monotone $U(p)$ for $p \in \{0,1\}^*$. As argued below, the class of computable infinite sequences is conceptually very interesting. The halting problem implies that there is no recursive enumeration of all partial-recursive functions with infinite domain; hence we cannot remove the finite sequences algorithmically. It is very fortunate that we don't have to. Hypothesis H_p is identified with the sequence $U(p)$, which may be finite, infinite, or possibly even empty. The class of considered hypotheses is $\mathcal{M} := \{H_p : p \in \{0,1\}^*\}$.

The weighted majority algorithm also needs weights w_p for each H_p. Ockham's razor combined with Epicurus' principle demand to assign a high (low) prior weight to a simple (complex) hypothesis. If complexity is identified with program length, then w_p should be a decreasing function of $\ell(p)$. It turns out that $w_p = 2^{-\ell(p)}$ is the "right" choice, since minimal p forms a prefix code and therefore $\sum_p w_p \leq 1$ as required.

Using H_p for prediction can now fail in two ways. H_p may make a wrong prediction or no prediction at all for x_t. The true hypothesis H_m is still assumed to produce an infinite sequence. The weighted majority algorithm in this setting makes at most $O\left(\log w_p^{-1}\right) = O(\ell(p))$ errors. It is also plausible that learning $\ell(p)$ bits requires $O(\ell(p))$ "trials."

Universal mixture prediction: Solomonoff (1978) defined the following universal a-priori probability

$$M(x) := \sum_{p:U(p)=x*} 2^{-\ell(p)}. \qquad (1)$$

That is, $M(x) = W$ is the total weight of the computable deterministic hypotheses consistent with x for the universal weight choice $w_p = 2^{-\ell(p)}$. The universal weighted majority algorithm predicted $\arg\max_a M(\dot{x}_{<t}a)$. Instead, one could also make a probability prediction $M(a|\dot{x}_{<t}) := M(\dot{x}_{<t}a)/M(\dot{x}_{<t})$, which is the relative weight of hypotheses in $\mathcal{M}_t$ predicting a. The higher the probability $M(\dot{x}_t|\dot{x}_{<t})$ that is assigned to the true next observation $\dot{x}_t$, the better. Consider the absolute prediction error $|1 - M(\dot{x}_t|\dot{x}_{<t})|$ and the logarithmic error $-\log M(\dot{x}_t|\dot{x}_{<t})$. The cumulative logarithmic error is bounded by $\sum_{t=1}^{n} -\log M(\dot{x}_t|\dot{x}_{<t}) = -\log M(\dot{x}_{1:n}) \leq \ell(p)$ for any program p that prints $\dot{x}*$. For instance, p could be chosen as the shortest one printing $\dot{x}_{1:\infty}$, which has length $Km(\dot{x}_{1:\infty}) := \min\{\ell(p) : U(p) = \dot{x}_{1:\infty}\}$. Using $1 - z \leq -\log z$ and letting $n \to \infty$ we get

$$\sum_{t=1}^{\infty} |1 - M(\dot{x}_t|\dot{x}_{<t})| \leq \sum_{t=1}^{\infty} -\log M(\dot{x}_t|\dot{x}_{<t}) \leq Km(\dot{x}_{1:\infty}).$$

Hence again, the cumulative absolute and logarithmic errors are bounded by the number of bits required to describe the true environment.

Universal Bayes

The exposition so far has dealt with deterministic environments only. Data sequences produced by real-world processes are rarely as clean as IQ test sequences. They are often noisy. This section deals with stochastic sequences sampled from computable probability distributions. The developed theory can be regarded as an instantiation of Bayesian learning. Bayes' theorem allows to update beliefs in face of new information but is mute about how to choose the prior and the model class to begin with. Subjective choices based on prior knowledge are informal, and traditional "objective" choices like Jeffrey's prior are not universal. Machine learning, the computer science branch of statistics, develops (fully) automatic inference and decision algorithms for very large problems. Naturally, machine learning has (re)discovered and exploited different principles (Ockham's and Epicurus') for choosing priors, appropriate for this situation. This leads to an alternative representation of universal probability as a mixture over all lower semi-computable semimeasures with Kolmogorov complexity-based prior as described below.

Bayes

Sequences $\omega = \omega_{1:\infty} \in \mathcal{X}^\infty$ are now assumed to be sampled from the "true" probability measure μ, that is, $\mu(x_{1:n}) := \mathrm{P}[\omega_{1:n} = x_{1:n}|\mu]$ is the μ-probability that ω starts with $x_{1:n}$. Expectations w.r.t. μ are denoted by $\mathbf{E}$. In particular for a function $f : \mathcal{X}^n \to \mathbb{R}$, we have $\mathbf{E}[f] = \mathbf{E}[f(\omega_{1:n})] = \sum_{x_{1:n}} \mu(x_{1:n}) f(x_{1:n})$.

Note that in Bayesian learning, measures, environments, and models are the same objects; let $\mathcal{M} = \{\nu_1, \nu_2, \ldots\} \equiv \{H_{\nu_1}, H_{\nu_2}, \ldots\}$ denotes a countable class of these measures≡hypotheses. Assume that μ is unknown but known to be a member of $\mathcal{M}$, and $w_\nu := \mathrm{P}[H_\nu]$ is the given prior belief in H_ν. Then the Bayesian mixture

$$\begin{aligned}\xi(x_{1:n}) &:= \mathrm{P}[\omega_{1:n} = x_{1:n}] \\ &= \sum_{\nu\in\mathcal{M}} \mathrm{P}[\omega_{1:n} = x_{1:n}|H_\nu]\mathrm{P}[H_\nu] \\ &\equiv \sum_{\nu\in\mathcal{M}} \nu(x_{1:n})w_\nu\end{aligned}$$

must be our a-priori belief in $x_{1:n}$, and $\mathrm{P}[H_\nu|\omega_{1:n} = x_{1:n}] = w_\nu\nu(x_{1:n})/\xi(x_{1:n})$ be our posterior belief in ν by Bayes' rule.

Universal Choice of $\mathcal{M}$

Next, we need to find a universal class of environments $\mathcal{M}_U$. Roughly speaking, Bayes' works if $\mathcal{M}$ contains the true environment μ. The larger $\mathcal{M}$, the less restrictive is this assumption. The class of all computable distributions, although only countable, is pretty large from a practical point of view, since it includes, for instance, all of today's valid physics theories. (Finding a non-computable physical system would indeed overturn the generally accepted Church-Turing thesis.) It is the largest class, relevant from a computational point of view. Solomonoff (1964, Eq. (13)) defined and studied the mixture over this class.

One problem is that this class is not (effectively = recursively) enumerable, since the class of computable functions is not enumerable due to the halting problem, nor is it decidable whether a function is a measure. Hence ξ is completely incomputable. Leonid Levin (Zvonkin & Levin, 1970) had the idea to "slightly" extend the class and include also lower semi-computable semimeasures.

A function $\nu : \mathcal{X}^* \to [0,1]$ is called a semimeasure iff $\nu(x) \geq \sum_{a\in\mathcal{X}} \nu(xa)\ \forall x \in \mathcal{X}^*$. It is a proper probability measure iff equality holds and $\nu(\epsilon) = 1$. $\nu(x)$ still denotes the ν-probability that a sequence starts with string x. A function is called lower semi-computable, if it can be approximated from below. Similarly to the fact that the class of partial recursive functions is recursively enumerable, one can show that the class $\mathcal{M}_U = \{\nu_1, \nu_2, \ldots\}$ of lower semi-computable semimeasures is recursively enumerable. In some sense $\mathcal{M}_U$ is the largest class of environments for which ξ is in some sense computable, but even larger classes are possible (Schmidhuber, 2002).

Kolmogorov Complexity

Before we can turn to the prior w_ν, we need to quantify complexity/simplicity. Intuitively, a string is simple if it can be described in a few words, like "the string of one million ones," and is complex if there is no such short description, like for a random object whose shortest description is specifying it bit by bit. We are interested in effective descriptions, and hence restrict decoders to be Turing machines. One can define the *prefix Kolmogorov complexity* of string x as the length ℓ of the shortest *halting* program p for which U outputs x:

$$K(x) := \min_p\{\ell(p) : U(p) = x \text{ halts}\}.$$

Simple strings like 000…0 can be generated by short programs, and hence have low Kolmogorov complexity, but irregular (e.g., random) strings are their own shortest description, and hence have high Kolmogorov complexity. For non-string objects o (like numbers and functions) one defines $K(o) := K(\langle o\rangle)$, where $\langle o\rangle \in \mathcal{X}^*$ is some standard code for o. In particular, $K(\nu_i) = K(i)$.

To be brief, K is an excellent universal complexity measure, suitable for quantifying Ockham's razor.

The Universal Prior

We can now quantify a prior biased toward simple models. First, we quantify the complexity of an environment ν or hypothesis H_ν by its Kolmogorov complexity $K(\nu)$. The universal prior should be a decreasing function in the model's complexity, and of course sum to (less than) one. Since $\sum_x 2^{-K(x)} \leq 1$ by the prefix property and Kraft's inequality, this suggests the choice

$$w_\nu = w_\nu^U := 2^{-K(\nu)}. \tag{2}$$

Since $\log i \leq K(\nu_i) \leq \log i + 2\log\log i$ for "most" i, most ν_i have prior approximately reciprocal to their index i as also advocated by Jeffreys and Rissanen.

Representations

Combining the universal class $\mathcal{M}_U$ with the universal prior 2, we arrive at the universal mixture

$$\xi_U(x) := \sum_{\nu\in\mathcal{M}_U} 2^{-K(\nu)}\nu(x) \tag{3}$$

which has remarkable properties. First, it is itself a lower semi-computable semimeasure, that is $\xi_U \in \mathcal{M}_U$, which is very convenient. Note that for most classes, $\xi \notin \mathcal{M}$.

Second, ξ_U coincides with M within an irrelevant multiplicative constant, and $M \in \mathcal{M}_U$. This means that the mixture over deterministic computable sequences is as rich as the mixture over the much larger class of semi-computable semimeasures. The intuitive reason is that the probabilistic semimeasures are in the convex hull of the deterministic ones, and therefore need not be taken extra into account in the mixture.

There is another, possibly the simplest, representation: One can show that $M(x)$ is equal to the probability that U outputs a string starting with x when provided with uniform random noise on the program tape. Note that a uniform distribution is also used in many no-free-lunch theorems to prove the impossibility of universal learners, but in our case the uniform distribution is piped through a universal Turing machine, which defeats these negative implications as we will see in the next section.

Applications

In the stochastic case, identification of the true hypothesis is problematic. The posterior $P[H|x]$ may not concentrate around the true hypothesis H_μ if there are other hypotheses H_ν that are not asymptotically distinguishable from H_μ. But even if model identification (*induction* in the narrow sense) fails, *predictions*, *decisions*, and *actions* can be good, and indeed, for universal learning this is generally the case.

Universal Sequence Prediction

Given a sequence $x_1 x_2 \ldots x_{t-1}$, we want to predict its likely continuation x_t. We assume that the strings which have to be continued are drawn from a computable "true" probability distribution μ. The maximal prior information a prediction algorithm can possess is the exact knowledge of μ, but often the true distribution is unknown. Instead, prediction is based on a guess ρ of μ. Let $\rho(a|x) := \rho(xa)/\rho(x)$ be the "predictive" ρ-probability that the next symbol is $a \in \mathcal{X}$, given sequence $x \in \mathcal{X}^*$. Since $\mu \in \mathcal{M}_U$ it is natural to use ξ_U or M for prediction.

Solomonoff's (Hutter, 2005; Solomonoff, 1978) celebrated result indeed shows that M converges to μ. For general alphabet it reads

$$\sum_{t=1}^{\infty} \mathbf{E}\left[\sum_{a\in\mathcal{X}} \left(M(a|\omega_{<t}) - \mu(a|\omega_{<t})\right)^2\right] \leq K(\mu)\ln 2 + O(1). \tag{4}$$

Analogous bounds hold for ξ_U and for other than the Euclidean distance, for example, the Hellinger and the absolute distance and the relative entropy.

For a sequence $a_1, a_2, \ldots$ of random variables, $\sum_{t=1}^{\infty} \mathbf{E}\left[a_t^2\right] \leq c < \infty$ implies $a_t \to 0$ for $t \to \infty$ with μ-probability 1 (w.p.1). Convergence is rapid in the sense that the probability that a_t^2 exceeds $\varepsilon > 0$ at more than $c/\varepsilon\delta$ times is bounded by δ. This might loosely be called the number of errors. Hence Solomonoff's bounds implies

$$M(x_t|\omega_{<t}) - \mu(x_t|\omega_{<t}) \longrightarrow 0 \text{ for any } x_t \text{ rapid w.p.1 for } t \to \infty.$$

The number of times M deviates from μ by more than $\varepsilon > 0$ is bounded by $O(K(\mu))$, that is, proportional to the complexity of the environment, which is again reasonable. A counting argument shows that $O(K(\mu))$ errors for most μ are unavoidable. No other choice for w_ν would lead to significantly better bounds. Again, in general it is not possible to determine *when* these "errors" occur. Multi-step lookahead convergence $M(x_{t:n_t}|\omega_{<t}) - \mu(x_{t:n_t}|\omega_{<t}) \to 0$ even for unbounded lookahead $n_t - t \geq 0$, relevant for delayed sequence prediction and in reactive environments, can also be shown.

In summary, M is an excellent sequence predictor under the only assumption that the observed sequence is drawn from some (unknown) computable probability distribution. No ergodicity, stationarity, or identifiability or other assumption is required.

Universal Sequential Decisions

Predictions usually form the basis for decisions and actions, which result in some profit or loss. Let $\ell_{x_t y_t} \in [0,1]$ be the received loss for decision $y_t \in \mathcal{Y}$ when $x_t \in \mathcal{X}$ turns out to be the true tth symbol of the sequence. The ρ-optimal strategy

$$y_t^{\Lambda_\rho}(\omega_{<t}) := \arg\min_{y_t} \sum_{x_t} \rho(x_t|\omega_{<t})\ell_{x_t y_t} \tag{5}$$

minimizes the ρ-expected loss. For instance, if we can decide among $\mathcal{Y} = \{sunglasses, umbrella\}$ and it turns out to be $\mathcal{X} = \{sun, rain\}$, and our personal loss matrix is $\ell = \left(\begin{smallmatrix} 0.0 & 0.1 \\ 1.0 & 0.3 \end{smallmatrix}\right)$, then Λ_ρ takes $y_t^{\Lambda_\rho} = sunglasses$ if $\rho(rain|\omega_{<t}) < 1/8$ and an *umbrella* otherwise. For $\mathcal{X} = \mathcal{Y}$ and 0–1 loss $\ell_{xy} = 0$ for $x = y$ and 1 else, Λ_ρ predicts the most likely symbol $y_t^{\Lambda_\rho} = \arg\max_a \rho(a|\omega_{<t})$ as in Sect. 2.

The cumulative μ(=true)-expected loss of Λ_ρ for the first n symbols is

$$\mathrm{Loss}_n^{\Lambda_\rho} := \sum_{t=1}^{n} \mathbf{E}\left[\ell_{\omega_t y_t^{\Lambda_\rho}(\omega_{<t})}\right] \equiv \sum_{t=1}^{n}\sum_{x_{1:t}} \mu(x_{1:t})\ell_{x_t y_t^{\Lambda_\rho}(x_{<t})}.$$

If μ is known, Λ_μ obviously results in the best decisions in the sense of achieving minimal expected loss among all strategies. For the predictor Λ_M based on M (and similarly ξ_U), one can show

$$\sqrt{\mathrm{Loss}_n^{\Lambda_M}} - \sqrt{\mathrm{Loss}_n^{\Lambda_\mu}} \le \sqrt{2K(\mu)\ln 2 + O(1)} \quad (6)$$

This implies that $\mathrm{Loss}_n^{\Lambda_M}/\mathrm{Loss}_n^{\Lambda_\mu} \to 1$ for $\mathrm{Loss}_n^{\Lambda_\mu} \to \infty$, or if $\mathrm{Loss}_\infty^{\Lambda_\mu}$ is finite, then also $\mathrm{Loss}_\infty^{\Lambda_M} < \infty$. This shows that M (via Λ_M) also performs excellent from a decision-theoretic perspective, that is, suffers loss only slightly larger than the optimal Λ_μ strategy.

One can also show that Λ_M is pareto-optimal (admissible) in the sense that every other predictor with smaller loss than Λ_M in some environment $\nu \in \mathcal{M}_U$ must be worse in another environment.

Universal Classification and Regression

The goal of classification and regression is to infer the functional relationship $f : \mathcal{Y} \to \mathcal{X}$ from data $\{(y_1, x_1), \ldots, (y_n, x_n)\}$. In a predictive online setting one wants to "directly" infer x_t from y_t given $(y_{<t}, x_{<t})$ for $t = 1, 2, 3, \ldots$. The universal induction framework has to be extended by regarding $y_{1:\infty}$ as independent side-information presented in the form of an oracle or extra tape information or extra parameter. The construction has to ensure that $x_{1:n}$ depends only on $y_{1:n}$ but is (functionally or statistically) independent of $y_{n+1:\infty}$.

First, we augment a monotone Turing machine with an extra input tape containing $y_{1:\infty}$. The Turing machine is called chronological if it does not read beyond $y_{1:n}$ before $x_{1:n}$ has been written. Second, semimeasures $\rho = \mu, \nu, M, \xi_U$ are extended to $\rho(x_{1:n}|y_{1:\infty})$, that is, one semimeasure $\rho(\cdot|y_{1:\infty})$ for each $y_{1:\infty}$ (no distribution over y is assumed). Any such semimeasure must be chronological in the sense that $\rho(x_{1:n}|y_{1:\infty})$ is independent of y_t for $t > n$, hence we can write $\rho(x_{1:n}|y_{1:n})$. In classification and regression, ρ is typically (conditionally) i.i.d., that is, $\rho(x_{1:n}|y_{1:n}) = \prod_{t=1}^{n}\rho(x_t|y_t)$, which is chronological, but note that the Bayesian mixture ξ is *not* i.i.d. One can show that the class of lower semi-computable chronological semimeasures $\mathcal{M}_U^{|} = \{\nu_1(\cdot|\cdot), \nu_2(\cdot|\cdot), \ldots\}$ is effectively enumerable.

The generalized universal a-priori semimeasure also has two equivalent definitions:

$$M(x_{1:n}|y_{1:n}) := \sum_{p:U(p,y_{1:n})=x_{1:n}} 2^{-\ell(p)} = \sum_{\nu\in\mathcal{M}} 2^{-K(\nu)}\nu(x_{1:n}|y_{1:n}) \quad (7)$$

which is again in $\mathcal{M}_U^{|}$. In case of $|\mathcal{Y}| = 1$, this reduces to (1) and (3). The bounds (4) and (6) and others continue to hold, now for all individual ys, that is, M predicts asymptotically x_t from y_t and $(y_{<t}, x_{<t})$ for any y, provided x is sampled from a computable probability measure $\mu(\cdot|y_{1:\infty})$. Convergence is rapid if μ is not too complex.

Universal Reinforcement Learning

The generalized universal a-priori semimeasure (7) can be used to construct a universal reinforcement learning agent, called AIXI. In reinforcement learning, an *agent* interacts with an *environment* in cycles $t = 1, 2, \ldots, n$. In cycle t, the agent chooses an *action* y_t (e.g., a limb movement) based on past *perceptions* $x_{<t}$ and past actions $y_{<t}$. Thereafter, the agent perceives $x_t \equiv o_t r_t$, which consists of a (regular) *observation* o_t (e.g., a camera image) and a real-valued *reward* r_t. The reward may be scarce, for example, just +1 (−1) for winning (losing) a chess game, and 0 at all other times. Then the next cycle $t + 1$ starts. The goal of the agent is to maximize its expected reward over its lifetime n. Probabilistic planning deals with the situation in which the environmental probability distribution $\mu(x_{1:n}|y_{1:n})$ is known. Reinforcement learning deals with the case of unknown μ. In universal reinforcement learning, the unknown μ is replaced

by M similarly to the prediction, decision, and classification cases above. The universally optimal action in cycle t is (Hutter, 2005)

$$y_t := \arg\max_{y_t} \sum_{x_t} \ldots \max_{y_n} \sum_{x_n} [r_t + \cdots + r_n] M(x_{1:n}|y_{1:n}). \tag{8}$$

The expectations (Σ) and maximizations (max) over future x and y are interleaved in chronological order to form an expectimax tree similarly to minimax decision trees in extensive zero-sum games like chess. Optimality and universality results similar to the prediction case exist.

Approximations and Practical Applications

Since K and M are only semi-computable, they have to be approximated in practice. For instance, $-\log M(x) = K(x) + O(\log l(x))$, and $K(x)$ can be and has been approximated by off-the-shelf compressors like Lempel-Ziv and successfully applied to a plethora of clustering problems (Cilibrasi & Vitányi, 2005). The approximations upper-bound $K(x)$ and, for example, for Lempel-Ziv converge to $K(x)$ if x is sampled from a context tree source. The ▶Minimum Description Length principle (Grünwald, 2007) also attempts to approximate $K(x)$ for stochastic x. The Context Tree Weighting algorithm considers a relatively large subclass of $\mathcal{M}_U$ that can be summed over efficiently. This can be and has been combined with Monte-Carlo sampling to efficiently approximate AIXI 8 (Veness, Ng, Hutter, & Silver, 2010). The time-bounded versions of K and M, namely Levin complexity Kt and the speed prior S have also been applied to various learning tasks (Gaglio, 2007).

Other Applications

Continuously parameterized model classes are very common in statistics. Bayesian's usually assume a-prior *density* over some parameter $\theta \in \mathbb{R}^d$, which works fine for many problems, but has its problems. Even for continuous classes $\mathcal{M}$, one can assign a (proper) universal prior (not density) $w_\theta^U := 2^{-K(\theta)} > 0$ for computable θ (and ν_θ), and 0 for uncomputable ones. This effectively reduces $\mathcal{M}$ to a discrete class $\{\nu_\theta \in \mathcal{M} : w_\theta^U > 0\} \subseteq \mathcal{M}_U$ which is typically dense in $\mathcal{M}$. There are various fundamental philosophical and statistical problems and paradoxes around (Bayesian) induction, which nicely disappear in the universal framework. For instance, universal induction has no zero and no improper p(oste)rior problem, that is, can confirm universally quantified hypotheses, is reparametrization and representation invariant, and avoids the old-evidence and updating problem, in contrast to most classical continuous prior densities. It even performs well in incomputable environments, actually better than the latter (Hutter, 2007).

Discussion and Future Directions

Universal learning is designed to work for a wide range of problems without any a-priori knowledge. In practice, we often have extra information about the problem at hand, which could and should be used to guide the forecasting. One can incorporate it by explicating all our prior knowledge z, and place it on an extra input tape of our universal Turing machine U, or prefix our observation sequence x by z and use $M(zx)$ for prediction.

Another concern is the dependence of K and M on U. The good news is that a change of U changes $K(x)$ only within an additive and $M(x)$ within a multiplicative constant independent of x. This makes the theory practically immune to any "reasonable" choice of U for large data sets x, but predictions for short sequences (shorter than typical compiler lengths) can be arbitrary. One solution is to take into account our (whole) scientific prior knowledge z (Hutter, 2006), and predicting the now long string zx leads to good (less sensitive to "reasonable" U) predictions. This is a kind of grand transfer learning scheme. It is unclear whether a more elegant theoretical solution is possible.

Finally, the incomputability of K and M prevents a *direct* implementation of Solomonoff induction. Most fundamental theories have to be approximated for practical use, sometimes systematically like polynomial time approximation algorithms or numerical integration, and sometimes heuristically like in many AI-search problems or in non-convex optimization problems. Universal machine learning is similar, except that its core quantities are only semi-computable. This makes them often hard, but as described in the previous section, not impossible, to approximate.

In any case, universal induction can serve as a "gold standard" which practitioners can aim at. Solomonoff's

theory considers the class of all computable (stochastic) models, and a universal prior inspired by Ockham and Epicurus, quantified by Kolmogorov complexity. This leads to a universal theory of induction, prediction, decisions, and, by including Bellman, to universal actions in reactive environments. Future progress on the issues above (incorporating prior knowledge, getting rid of the compiler constants, and finding better approximations) will lead to new insights and will continually increase the number of applications.

Cross References

- ►Bayes Rule
- ►Bayesian Methods
- ►Bayesian Reinforcement Learning
- ►Classification
- ►Data Set
- ►Discriminative Learning
- ►Hypothesis Space
- ►Inductive Inference
- ►Loss
- ►Minimum Description Length
- ►On-line Learning
- ►Prior Probability
- ►Reinforcement Learning
- ►Time Series

Recommended Reading

Cilibrasi, R., & Vitányi, P. M. B. (2005). Clustering by compression. *IEEE Transactions on Information Theory, 51*(4), 1523–1545.

Gaglio, M. (2007). Universal search. *Scholarpedia, 2*(11), 2575.

Grünwald, P. D. (2007). *The minimum description length principle*. Cambridge: The MIT Press.

Hutter, M. (2005). *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Berlin: Springer.

Hutter, M. (2006). Human knowledge compression prize. open ended, http://prize.hutter1.net/.

Hutter, M. (2007). On universal prediction and Bayesian confirmation. *Theoretical Computer Science, 384*(1), 33–48.

Li, M., & Vitányi, P. M. B. (2008). *An introduction to Kolmogorov complexity and its applications* (3rd ed.). Berlin: Springer.

Schmidhuber, J. (2002). Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. *International Journal of Foundations of Computer Science, 13*(4), 587–612.

Solomonoff, R. J. (1964). A formal theory of inductive inference: Parts 1 and 2. *Information and Control, 7*, 1–22 and 224–254.

Solomonoff, R. J. (1978). Complexity-based induction systems: Comparisons and convergence theorems. *IEEE Transactions on Information Theory, IT-24*, 422–432.

Veness, J., Ng, K. S., Hutter, M., & Silver, D. (2010). Reinforcement learning via AIXI approximation. In *Proceedings of 24th AAAI conference on artificial intelligence*, Atlanta. AAAI Press. 605–611.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.

Zvonkin, A. K., & Levin, L. A. (1970). The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys, 25*(6), 83–124.

Unknown Attribute Values

►Missing Attribute Values

Unknown Values

►Missing Attribute Values

Unlabeled Data

Unlabeled data are ►data for which there are no target values. Unlabeled data are used in ►unsupervised learning. They stand in contrast to *labeled data* that have target values and are used in ►supervised learning.

Unsolicited Commercial Email Filtering

►Text Mining for Spam Filtering

Unstable Learner

An *unstable learner* produces large differences in generalization patterns when small changes are made to its initial conditions. The obvious initial condition is the set of training data used – for an unstable learner, sampling

a slightly different training set produces a large difference in testing behavior. Some models can be unstable in additional ways, for example ►neural networks are unstable with respect to the initial weights. In general this is an undesirable property – high sensitivity to training conditions is also known as high ►variance, which results in higher overall mean squared error. The flexibility enabled by being sensitive to data can thus be a blessing or a curse. Unstable learners can however be used to an advantage in ►ensemble learning methods, where large variance is "averaged out" across multiple learners.

Examples of unstable learners are: neural networks (assuming gradient descent learning), and ►decision trees. Examples of stable learners are ►support vector machines, ►K-nearest neighbor classifiers, and ►decision stumps. It should of course be recognized that there is a continuum between "stable" and "unstable," and the opinion of whether something is "sensitive" to initial conditions is somewhat of a subjective one. See also ►bias-variance decomposition for a more formal interpretation of this concept.

Unsupervised Learning

Unsupervised learning refers to any machine learning process that seeks to learn structure in the absence of either an identified output (cf. ►supervised learning) or feedback (cf. ►reinforcement learning). Three typical examples of unsupervised learning are ►clustering, ►association rules, and ►self-organizing maps.

Unsupervised Learning on Document Datasets

►Document Clustering

Utility Problem

►Explanation-Based Learning

V

Value Function Approximation

Michail G. Lagoudakis
Technical University of Crete

Synonyms

Approximate Dynamic Programming, Neuro-dynamic Programming, Cost-to-go Function Approximation

Definition

The goal in sequential decision making under uncertainty is to find good or optimal policies for selecting actions in stochastic environments in order to achieve a long term goal; such problems are typically modeled as ►Markov Decision Processes (MDPs). A key concept in MDPs is the *value function*, a real-valued function that summarizes the long-term goodness of a decision into a single number and allows the formulation of optimal decision making as an optimization problem. Exact representation of value functions in large real-world problems is infeasible, therefore a large body of research has been devoted to *value function approximation* methods, which sacrifice some representation accuracy for the sake of scalability. These approaches have delivered effective approaches to deriving good policies in hard decision problems and laid the foundation for efficient reinforcement learning algorithms, which learn good policies in unknown stochastic environments through interaction.

Motivation and Background

Markov Decision Processes

A *Markov Decision Process* (MDP) is a six-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{D})$, where $\mathcal{S}$ is the state space of the process, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}$ is a Markovian transition model ($\mathcal{P}(s'|s,a)$ denotes the probability of a transition to state s' when taking action a in state s), $\mathcal{R}$ is a reward function ($\mathcal{R}(s,a)$ is the reward for taking action a in state s), $\gamma \in (0,1]$ is the discount factor for future rewards (a reward received after t steps is weighted by γ^t), and $\mathcal{D}$ is the initial state distribution (Puterman, 1994). MDPs are discrete-time processes. The process begins at time $t = 0$ in some state $s_0 \in \mathcal{S}$ drawn from $\mathcal{D}$. At each time step t, the decision maker observes the current state of the process $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$. The next state of the process s_{t+1} is drawn stochastically according to the transition model $\mathcal{P}(s_{t+1}|s_t, a_t)$ and the reward r_t at that time step is determined by the reward function $\mathcal{R}(s_t, a_t)$. The horizon h is the temporal extent of each run of the process and is typically infinite. A complete run of the process over its horizon is called an *episode* and consists of a long sequence of states, actions, and rewards:

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \ldots s_{h-1} \xrightarrow[r_{h-1}]{a_{h-1}} s_h.$$

The quantity of interest is the *expected total discounted reward* from any state s:

$$E\left(r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots + \gamma^h r_h \mid s_0 = s\right) = E\left(\sum_{t=0}^{h} \gamma^t r_t \,\middle|\, s_0 = s\right),$$

where the expectation is taken with respect to all possible trajectories of the process in the state space under the decisions made and the transition model, assuming that the process is initialized in state s. The goal of the decision maker is to make decisions so that the expected total discounted reward, when s is drawn from $\mathcal{D}$, is optimized. (The optimization objective could be maximization or minimization depending on the problem. Here, we adopt a reward maximization viewpoint, but there are analogous definitions for cost minimization. There are also other popular optimality measures, such as maximization/minimization of the average reward/cost per step.)

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Policies

A *policy* dictates how the decision maker chooses actions in each state. A *stationary, deterministic policy* π is a mapping $\pi : \mathcal{S} \mapsto \mathcal{A}$ from states to actions; $\pi(s)$ denotes the action the agent takes in state s. In this case, there is a single action choice for each state, and this choice does not change over time. In contrast, a *stationary, stochastic policy* π is a mapping $\pi : \mathcal{S} \mapsto \Omega(\mathcal{A})$, where $\Omega(\mathcal{A})$ is the set of all probability distributions over $\mathcal{A}$. Stochastic policies are also called *soft*, for they do not commit to a single action per state; $\pi(a|s)$ stands for the probability of choosing action a in state s under policy π. Each policy π (deterministic or stochastic) is characterized by the expected total discounted reward it accumulates during an episode. An *optimal policy* π^* for an MDP is a policy that maximizes the expected total discounted reward from any state $s \in \mathcal{S}$. It is well-known that for every MDP there exists at least one, not necessarily unique, optimal policy, which is stationary and deterministic.

Value Functions

The quality of any policy π can be quantified formally through a value function, which measures the expected return of the policy under different process initializations. For any MDP and any policy π, the *state value function* V assigns a numeric value to each state. The value $V^\pi(s)$ of a state s under a policy π is the expected return, when the process starts in state s and the decision maker follows policy π (all decisions at all steps are made according to π):

$$V^\pi(s) = E_{a_t \sim \pi \,;\, s_t \sim \mathcal{P} \,;\, r_t \sim \mathcal{R}} \left(\sum_{t=0}^{\infty} \gamma^t r_t \;\middle|\; s_0 = s \right).$$

Similarly, the *state-action value function* Q assigns a numeric value to each pair (s, a) of states and actions. The value $Q^\pi(s, a)$ of taking action a in state s under a policy π is the expected return when the process starts in state s, and the decision maker takes action a for the first step and follows policy π thereafter:

$$Q^\pi(s, a) = E_{a_t \sim \pi \,;\, s_t \sim \mathcal{P} \,;\, r_t \sim \mathcal{R}} \left(\sum_{t=0}^{\infty} \gamma^t r_t \;\middle|\; s_0 = s,\ a_0 = a \right).$$

The state and the state-action value functions for a deterministic policy π are related as follows:

$$V^\pi(s) = Q^\pi\left(s, \pi(s)\right).$$

For a stochastic policy π this relationship needs to take into account the probability distribution over actions:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s, a).$$

The state-action value function of a policy π (either deterministic or stochastic) can also be expressed in terms of the state value function:

$$Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) V^\pi(s').$$

The *optimal value functions* $V^* = V^{\pi^*}$ and $Q^* = Q^{\pi^*}$ are the state and the state-action value functions of any optimal policy π^*. Even if there are several distinct optimal policies, they all share the same unique optimal value functions.

Bellman Equations

Given the full MDP model, the state or the state-action value function of any given policy can be computed by solving a linear system formed using the linear Bellman equations. In general, the linear *Bellman equation* relates the value of the function at any point to the values of the function at several – in fact, all – other points. This is achieved by separating the first step of an episode from the rest and using the definition of the value function recursively in the next step. In particular, for any deterministic policy π, the linear Bellman equation for the state value function is

$$V^\pi(s) = \mathcal{R}(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, \pi(s)) V^\pi(s'),$$

whereas for a stochastic policy π, it becomes

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) V^\pi(s') \right).$$

The exact V^π values for all states can be found by solving the $(|\mathcal{S}| \times |\mathcal{S}|)$ linear system that results from writing down the linear Bellman equation for all states.

Similarly, the linear Bellman equation for the state-action value function given any deterministic policy π is

$$Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) Q^\pi\left(s', \pi(s')\right),$$

whereas for a stochastic policy π, it becomes

$$Q^{\pi}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^{\pi}(s',a').$$

The exact Q^{π} values for all state-action pairs can be found by solving the $(|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}||\mathcal{A}|)$ linear system that results from writing down the linear Bellman equation for all state-action pairs.

The unique optimal state or state-action value function can be computed even for an unknown optimal policy π^* using the non-linear *Bellman optimality equation*, which relates values of the function at different points while exploiting the fact that there exists a deterministic optimal policy that achieves the maximum value at each point. In particular, the non-linear Bellman optimality equation for the state value function is

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^*(s') \right\},$$

whereas for the state-action value function is

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathcal{A}} \{ Q^*(s',a') \}.$$

The functions V^* and Q^* can be approximated arbitrarily closely by the iterative application of the operator formed by the right-hand side of the equations above (Bellman optimality operator). This iteration is a contraction with rate γ, so starting with any arbitrary initialization it eventually converges to V^* or Q^*.

Significance of Value Functions

Value functions play a critical role in sequential decision making because they address two core problems: policy evaluation and policy improvement. Policy evaluation refers to the problem of quantifying the quality of any given policy π in a given MDP. Apparently, computing the value function V^{π} or Q^{π} using the Bellman equations provides the solution to this problem. Policy improvement, on the other hand, refers to the problem of deriving an improved policy π' given any base policy π, so that π' is at least as good as π and possibly better. The knowledge of V^{π} or Q^{π} allows for the derivation of an improved deterministic policy π' through a simple one-step look-ahead maximization procedure:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \left\{ \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^{\pi}(s') \right\}$$
$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \{ Q^{\pi}(s,a) \}.$$

Note that this maximization does not need the MDP model when using the state-action value function. Once policy evaluation and policy improvement have been addressed, the derivation of an optimal policy for any MDP is straightforward. One can alternate between policy evaluation and policy improvement producing a sequence of improving policies until convergence to an optimal policy; this algorithm is known as policy iteration. Alternatively, one can iteratively compute an optimal value function V^* or Q^* and extract an optimal policy through a trivial step of policy improvement on top of V^* or Q^*; this algorithm is known as value iteration. In either case, value functions provide the means to the end.

The problem of deriving an optimal policy using the full MDP model is known as planning. Nevertheless, in many real-world sequential decision domains the model is unknown. The problem of optimal decision making in an unknown stochastic environment is known as reinforcement learning, because the decision maker relies on the feedback received through interaction with the environment to reinforce or discourage past decisions. More specifically, the learner interacts with an unknown MDP and typically observes the state of the process and the immediate reward at every step, however $\mathcal{P}$ and $\mathcal{R}$ are not accessible. At each step of interaction, the learner observes the current state s, chooses an action a, and observes the resulting next state s' and the reward received r, thus learning is based on (s,a,r,s') samples. The core problems in reinforcement learning are known as prediction and control. Prediction refers to the problem of learning the value function of a given policy π in an unknown MDP through interaction. Well-known algorithms for the prediction problem are Monte-Carlo Estimation and Temporal Difference (TD) learning. Control, on the other hand, refers to the problem of gradually learning a good or even optimal policy in an unknown MDP through interaction. Well-known algorithms for the control problem are SARSA and Q-learning. Again,

value functions play a critical role in reinforcement learning; they are absolutely necessary for the prediction problem and the majority of control approaches are value-function-based.

Structure of Learning System

Value Function Approximation

Most algorithms for planning or learning in MDPs rely on computing or learning a value function. However, if the state space of the process is fairly large, the exact (tabular) representation of a value function becomes problematic. Not only does memory space become insufficient very quickly, but also computing or learning accurately all the distinct entries of the function requires a tremendous amount of computation and data. This is known as the ►*curse of dimensionality*: the exponential growth of the state or action space as a function of the dimensionality of the state or action. The urgent need for solutions to large real-world sequential decision problems has drawn attention to approximate methods. These methods use function approximation techniques for approximating value functions, therefore they sacrifice some representational accuracy in order to make the representation manageable in practice. Sacrificing accuracy in the representation of the value function is acceptable, since the ultimate goal is to find a good policy and not necessarily an accurate value function. As a result, value function approximation methods cannot guarantee optimal solutions, but only good solutions. This is not to say that they are doomed to always finding suboptimal solutions; if an optimal solution lies within the space spanned by the value function approximation scheme, it is possible that an optimal solution will be discovered.

Let $\widehat{V}^{\pi}(s;w)$ be an approximation to the state value function $V^{\pi}(s)$ represented by a parametric approximation architecture with free parameters w. The key idea of value function approximation is that the parameters w can be adjusted appropriately so that the approximate values are "close enough" to the original values,

$$\widehat{V}^{\pi}(s;w) \approx V^{\pi}(s),$$

and, therefore, $\widehat{V}^{\pi}$ can be used in place of the exact value function V^{π}. Similarly, let $\widehat{Q}^{\pi}(s,a;w)$ be an approximation to the state-action value function $Q^{\pi}(s,a)$. Again, the goal is to adjust the parameters w so that

$$\widehat{Q}^{\pi}(s,a;w) \approx Q^{\pi}(s,a),$$

and, therefore, $\widehat{Q}^{\pi}$ can be used in place of the exact value function Q^{π}. Approximations $\widehat{V}^{*}$ and $\widehat{Q}^{*}$ of the optimal value functions V^{*} and Q^{*} are defined similarly. The characterization "close enough" ($\approx$) accepts a variety of interpretations in this context and it does not necessarily refer to the minimization of some norm. Value function approximation should be regarded as a *functional* approximation rather than as a pure *numerical* approximation, where "functional" refers to the ability of the approximation to play closely the functional role of the original value function within a decision making algorithm.

The benefits of value function approximation are obvious. The storage requirements are much smaller compared to the tabular case, since only the parameters w need to be stored along with a compact description of the functional form of the architecture. In general, for most approximation architectures, the storage needs are independent of the size of the state space and/or the size of the action space. Furthermore, for most approximation architectures there is no restriction on the state space to be a finite set; it could be an infinite, or even a continuous, space. This flexibility nevertheless reveals the need for good generalization abilities on behalf of the architecture, since the approximate value function will have to provide good values over the entire state/state-action space, using data only from a limited subset of the space.

The main difficulty associated with value function approximation, beyond the loss in accuracy, is the choice of the *projection method*, which is the method of finding appropriate parameters that maximize the accuracy of the approximation according to certain criteria and with respect to the target function. Typically, for ordinary function approximation, this is accomplished using a training set of examples of the form $\{s, V^{\pi}(s)\}$, $\{s, V^{*}(s)\}$, $\{(s,a), Q^{\pi}(s,a)\}$, or $\{(s,a), Q^{*}(s,a)\}$ that provide the true value of the target function at certain sample points s or (s,a) (supervised learning). Unfortunately, in the context of sequential decision making, the target value function is completely unknown. Had it been possible to compute it easily, value function

approximation would have been unnecessary. In fact, it is not possible to analytically compute the true value of the target value function even at certain isolated sample points due to interdependencies between the values at all points. The implication of this difficulty is that evaluation and projection to the approximation architecture must be blended together. This is usually achieved by trying to find values for the free parameters so that the approximate function retains some properties of the original exact value function. Another implication of using approximation for value functions is that all convergence properties of exact planning or learning algorithms are compromised. Therefore, significant attention must be paid to the choice of the approximation architecture and the evaluation and projection method to minimize the chances for divergence or oscillations.

Approximation Architectures

There is a variety of architectures available for value function approximation: ►perceptrons, ►neural networks, splines, polynomials, ►radial basis functions, ►support vector machines, ►decision trees, CMACs, wavelets, etc. These architectures have diverse representational power and generalization abilities and the most appropriate choice will heavily depend on the properties of the decision making problem at hand. The projection methods associated with these approximation architectures are typically designed for a supervised learning setting. For successful use in the context of decision making, combined evaluation and projection methods are necessary.

A broad categorization of approximation architectures distinguishes between nonlinear and linear architectures. The characterization "nonlinear" or "linear" refers to the way the free parameters enter into the architecture and not to the approximation ability of the architecture. Nonlinear architectures are usually more expressive than the linear ones, due to the complex interactions among their free parameters, however tuning their parameters is a much more elaborate task compared to tuning the parameters of their linear counterparts. Linear architectures are perhaps the most popular choice for value function approximation; interestingly, most theoretical results on convergence properties in the context of value function approximation are restricted to linear architectures.

A *linear approximation architecture* approximates a function $V^\pi(s)$ or $Q^\pi(s,a)$ as a linear weighted combination of k *basis functions* (also called *features*):

$$\widehat{V}^\pi(s;w) = \sum_{j=1}^{k} \phi_j(s) w_j = \phi(s)^\top w$$

$$\widehat{Q}^\pi(s,a;w) = \sum_{j=1}^{k} \phi_j(s,a) w_j = \phi(s,a)^\top w.$$

The free *parameters* of the architecture are the coefficients w_j of the combination (also called *weights*). The basis functions ϕ_j are fixed, but arbitrary and, in general, nonlinear, functions of s or (s,a). It is required that the basis functions ϕ_j are linearly independent to ensure that there are no redundant parameters and that the matrices involved in the computations are full rank. In general, $k \ll |\mathcal{S}|$ and $k \ll |\mathcal{S}||\mathcal{A}|$ and the basis functions ϕ_j have small compact descriptions. As a result, the storage requirements of a linear approximation architecture are much smaller than those of the tabular representation of a value function. There is a large variety of linear approximation architectures and in fact many common schemes for value function approximation can be cast as linear architectures.

- *Look-up table*: This is the exact tabular representation (There is no approximation under this scheme; it is included only to illustrate that exact representation belongs in the family of linear architectures.) suitable for problems with discrete state variables. Each basis function is an indicator function whose value is 1 only for a specific discrete input point (state or state-action) and 0 otherwise. Each parameter stores one value/entry of the table.
- *Discretization*: This is a common technique for turning a continuous space into discrete using a uniform- or variable-resolution grid. One indicator basis function is assigned to each cell of the discretization and the corresponding parameter holds the value of that cell.
- *Tile coding (CMAC)*: This scheme utilizes several overlapping discretizations (tilings) for better accuracy. It generates indicator basis functions for each cell of each tiling and concatenates the basis functions for all tilings. Each parameter corresponds to

one cell in one tiling, but the value at each input point is computed additively from the values of all containing cells from all tilings.

- *State aggregation*: This is a family of schemes where "similar" (by some metric) states are grouped together and are treated as one state. The similarity metric is usually formed through dimensionality reduction techniques for identifying the most significant dimensions in a high-dimensional input space, unlike conventional proximity measures in the same space. There is one indicator basis function for each group and a single value for all states in the group.
- *Polynomials*: This is a generic approximation scheme suitable for problems with several (continuous) state variables. Each basis function is a polynomial term composed of state variables up to a certain degree.
- *Radial basis functions (RBFs)*: This is another generic approximation scheme suitable for continuous state variables. Each basis function is a Gaussian with fixed mean and variance; the Gaussians are topologically arranged so that they cover the input space with some overlap.
- *Kernel methods*: Kernels are symmetric functions between two points and they are used to represent compactly dot products of feature vectors in high- or even infinite-dimensional spaces. The compactness of kernels allows for approximation schemes that essentially enjoy the flexibility provided by a huge or infinite number of basis functions. The basis functions, in this case, are implicitly defined through the choice of the kernel.
- *Partitioning*: This technique is used for constructing complex approximators by partitioning the state space in several subsets and using a different approximator in each one of them. If linear architectures are used in all partitions, then a set of basis functions for the global architecture can be constructed by concatenating the basis functions of the smaller linear architectures multiplying each subset with an indicator function for the corresponding partition.

Linear architectures offer several advantages: they are easy to implement and use, and their behavior is fairly transparent, both from an analysis standpoint and from a debugging and feature engineering standpoint. It is usually relatively easy to get some insight into the reasons for which a particular choice of features succeeds or fails. This is facilitated by the fact that the magnitude of each parameter is related to the importance of the corresponding feature in the approximation (assuming normalized features).

A *nonlinear approximation architecture* approximates a function $V^\pi(s)$ or $Q^\pi(s,a)$ using arbitrary parametric functions of s and (s,a), possibly in conjunction with some *features* ϕ computed over s and (s,a). The best-known representative of this category are the multi-layer feed-forward neural networks, which use one or more layers of linear combinations followed by a nonlinear sigmoidal transformations (thresholding). In their simplest form (one layer), neural networks approximate value functions as

$$\widehat{V}^\pi(s;w,\theta) = \sum_{i=1}^{m} \theta_i \sigma\left(\sum_{j=1}^{k} \phi_j(s) w_{ji}\right)$$

$$= \sum_{i=1}^{m} \theta_i \sigma\left(\phi(s)^\top w_i\right)$$

$$\widehat{Q}^\pi(s,a;w,\theta) = \sum_{i=1}^{m} \theta_i \sigma\left(\sum_{j=1}^{k} \phi_j(s,a) w_{ji}\right)$$

$$= \sum_{i=1}^{m} \theta_i \sigma\left(\phi(s,a)^\top w_i\right).$$

Common choices for the differentiable, bounded, and monotonically increasing function σ are the hyperbolic tangent function $\sigma(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ and the logistic function $\sigma(x) = 1/(1+e^{-x})$. The free *parameters* of the architecture (also called *weights*) are the coefficients w_{ji} of the linear combinations of the inputs and the coefficients θ_i of the linear combination of the sigmoidal transformations for the output. Notice that the parameters w_{ji} enter non-linearly into the approximation.

A key question in all approximation architectures is *how* features are generated and selected. The *feature generation and selection* problem is an open question that spans most of machine learning research and admits no easy and general answer. Prior domain-specific knowledge and experience can be very helpful in choosing appropriate features. Several recent studies also describe

methods for automatically generating features targeted for value function approximation (Menache et al., 2005; Mahadevan and Maggioni, 2007; Parr et al., 2007).

Learning

Learning (or *training*, or *parameter estimation*) in value function approximation refers to parameter tuninlg methods that take as input a policy π, an approximation architecture for V^π/Q^π, and the full MDP model or samples of interaction with the process and output a set of parameters w^π such that $\widehat{V}^\pi/\widehat{Q}^\pi$ is a good approximation to V^π/Q^π. Learning also covers methods for the harder problem of taking an approximation architecture for V^*/Q^* and the model or samples and outputting a set of parameters w^* such that $\widehat{V}^*/\widehat{Q}^*$ is a good approximation to V^*/Q^*. The former problem is somewhat easier because the policy π, unlike an optimal policy π^*, is known and therefore in the presence of a simulator of the process the value function can be estimated at any isolated point using Monte-Carlo estimation techniques based on repeated policy rollouts from that point. Each rollout is an execution of an episode starting from a state s (or state-action (s,a)) using policy π to obtain an unbiased estimate of the return of the policy from s (or (s,a)). In this case, value function approximation can be cast as a classic supervised learning problem; the true value of V^π/Q^π is estimated at a subset of points to form a training set, which can be subsequently used to train the approximation architecture using supervised learning techniques. However, in the absence of a simulator or when seeking approximations to V^*/Q^*, evaluation and projection into the architecture have to be carried out simultaneously.

The true value function has two key properties: it satisfies the Bellman equations and it is the fixed point of the Bellman operator. Learning in value function approximation strives to find values for the free parameters so that the approximate function retains at least one of these properties to the extent this is possible. Learning methods that focus on satisfying the Bellman equations attempt to find an approximate function that minimizes the Bellman residual, the difference between the left- and the right-hand sides of the system of Bellman equations. On the other hand, learning methods that focus on the fixed point property attempt to find an approximate function that exhibits a fixed point behavior under the combined application of the Bellman operator and projection onto the space spanned by the basis functions. Experimental evidence suggests that it is really hard to satisfy both properties under approximation and therefore these two approaches differ significantly in their solutions. The majority of existing learning methods focus on fixed point approximation, which experimentally has been shown to exhibit more stable behavior and delivers better policies. There are also proposals for combining the benefits of both approaches into a hybrid method (Johns et al., 2009).

The most widely-used learning approach is based on gradient descent and is applicable to any approximation architecture that is differentiable with respect to its parameters. Any stochastic approximation learning method for tabular representations of value functions can be extended to approximate representations. For example, given any sample (s,a,r,s') of interaction with the process, the Temporal Difference (TD) learning update rule

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha\left(r + \gamma V^\pi(s') - V^\pi(s)\right)$$

for tabular representations, where $\alpha \in (0,1]$ is the learning rate, becomes

$$w^\pi \leftarrow w^\pi + \alpha\left(r + \gamma \widehat{V}^\pi(s'; w^\pi) - \widehat{V}^\pi(s; w^\pi)\right)\nabla_{w^\pi}\widehat{V}^\pi(s; w^\pi)$$

under an approximation scheme $\widehat{V}^\pi$. Similarly, the SARSA update rule

$$Q^\pi(s,a) \leftarrow Q^\pi(s,a) + \alpha\left(r + \gamma Q^\pi(s', \pi(s')) - Q^\pi(s,a)\right)$$

for tabular representations, becomes

$$w^\pi \leftarrow w^\pi + \alpha\left(r + \gamma\widehat{Q}^\pi(s', \pi(s'); w^\pi) - \widehat{Q}^\pi(s,a; w^\pi)\right) \nabla_{w^\pi}\widehat{Q}^\pi(s,a; w^\pi)$$

under an approximation scheme $\widehat{Q}^\pi$. Finally, the Q-learning update rule

$$Q^*(s,a) \leftarrow Q^*(s,a) + \alpha\left(r + \gamma \max_{a' \in \mathcal{A}}\{Q^*(s',a')\} - Q^*(s,a)\right)$$

for tabular representations, under an approximation scheme $\widehat{Q}^*$ becomes

$$w^* \leftarrow w^* + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} \left\{ \widehat{Q}^*(s', a'; w^*) \right\} - \widehat{Q}^*(s, a; w^*) \right) \nabla_{w^*} \widehat{Q}^*(s, a; w^*).$$

These rules are applicable to any approximation architecture, linear or non-linear. However, when using linear architectures they can be greatly simplified, because the gradient with respect to a parameter w_j is simply the corresponding basis function ϕ_j, for $j = 1, 2, \ldots, k$.

$$\text{TD: } w_j^\pi \leftarrow w_j^\pi + \alpha \left(r + \gamma \phi(s')^\top w^\pi - \phi(s)^\top w^\pi \right) \phi_j(s)$$

$$\text{SARSA: } w_j^\pi \leftarrow w_j^\pi + \alpha \left(r + \gamma \phi(s', \pi(s'))^\top w^\pi - \phi(s, a)^\top w^\pi \right) \phi_j(s, a)$$

$$Q\text{-learning: } w_j^* \leftarrow w_j^* + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} \left\{ \phi(s', a')^\top w^* \right\} - \phi(s, a)^\top w^* \right) \phi_j(s, a)$$

More sophisticated learning approaches rely on retaining the desired value function property in a batch manner by processing several samples collectively. A variety of least-squares techniques have been proposed for linear architectures giving rise to several least-squares reinforcement learning methods, such as Least-Squares Temporal Difference (LSTD) learning (Bradtke and Barto, 1996), Least-Squares Policy Evaluation (LSPE) (Nedić and Bertsekas, 2003), Hybrid Least-Squares Approximation (Johns et al., 2009), and Least-Squares Policy Iteration (LSPI) (Lagoudakis and Parr, 2003). The parameters of a linear architecture can also be estimated using Linear Programming (de Farias and Van Roy, 2003). Specialized learning algorithms have been proposed when using a kernel-based approximation architecture, based either on Gaussian Process TD (GPTD) (Engel et al., 2003), Gaussian Process SARSA (GPSARSA) (Engel et al., 2005), kernelized LSTD (KLSTD) and LSPI (KLSPI) (Xu et al., 2005), Support Vector Regression (Bethkeh et al., 2008), or Gaussian Process regression (Rasmussen and Kuss, 2004; Bethke and How, 2009). A unified view of kernelized value function approximation is offered by Taylor and Parr (2009). On the other hand, boot-strapping – the updating of a value estimate based on other value estimates – is the main learning approach behind batch methods for non-linear architectures, such as Fitted Q-Iteration (FQI) (Ernst et al., 2005).

Examples

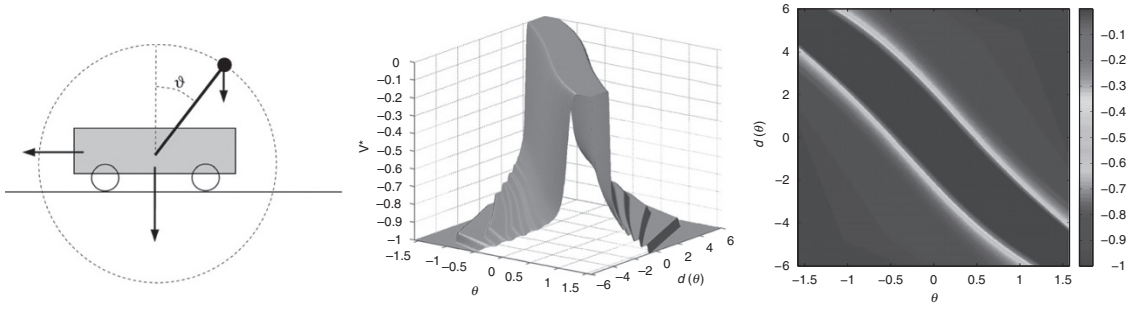
Very close approximations of the state value function of optimal policies in two well-known problems are presented to illustrate the difficulty of value function approximation. To obtain these close approximations, a fine discretization of the two-dimensional state space into a uniform grid of 250×250 was used for representation. The state-action value function Q was initially computed using approximate policy iteration (a sparse-matrix version of LSPI) with a set of indicator basis functions over the state grid and all actions and 500 (s, a, r, s') samples for each one of the 187,500 discrete cells (s, a), until convergence to a near-optimal policy; the state value function V was extracted from the Q values.

Inverted Pendulum

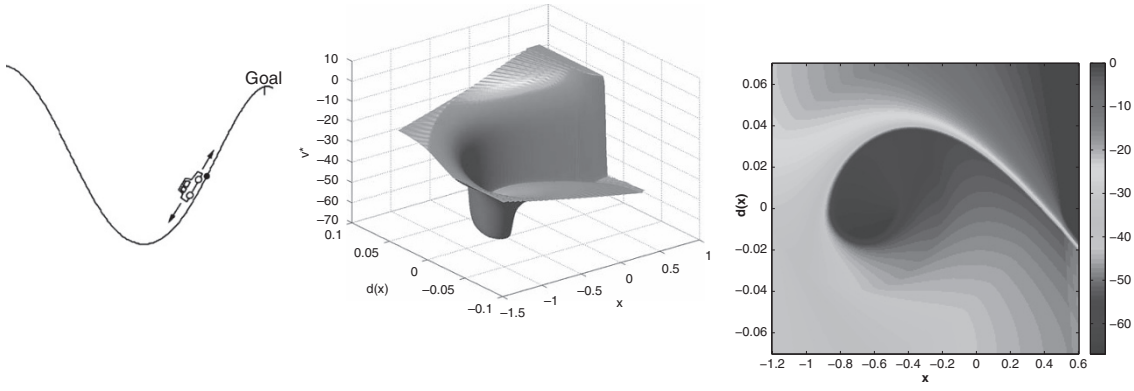
The *inverted pendulum* problem is to balance a pendulum of unknown length and mass at the upright position by applying forces to the cart it is attached to (Fig. 1, left). Three actions are allowed: left force LF (−50 N), right force RF (+50 N), or no force NF (0 N). All three actions are noisy; Gaussian noise with $\mu = 0$ and $\sigma^2 = 10$ is added to the chosen action. The state space of the problem is continuous and consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. The transitions are governed by the nonlinear dynamics of the system and depend on the current state and the current (noisy) control u:

$$\ddot{\theta} = \frac{g \sin(\theta) - \alpha m l (\dot{\theta})^2 \sin(2\theta)/2 - \alpha \cos(\theta) u}{4l/3 - \alpha m l \cos^2(\theta)},$$

where g is the gravity constant ($g = 9.8\,\mathrm{m/s^2}$), m is the mass of the pendulum (default: $m = 2.0$ kg), M is the mass of the cart (default: $M = 8.0$ kg), l is the length of the pendulum (default: $l = 0.5$ m), and $\alpha = 1/(m + M)$. The simulation step is 0.1 s, thus the control input is given at a rate of 10 Hz, at the beginning of each time step, and is kept constant during any time step. A reward of 0 is given as long as the angle of the



Value Function Approximation. Figure 1. Inverted pendulum: state value function of an optimal policy (3D and 2D) (Courtesy of Ioannis Rexakis)



Value Function Approximation. Figure 2. Mountain car: state value function of an optimal policy (3D and 2D) (Courtesy of Ioannis Rexakis)

pendulum does not exceed $\pi/2$ in absolute value (the pendulum is above the horizontal line). An angle greater than $\pi/2$ in absolute value signals the end of the episode and a reward (penalty) of –1. The discount factor of the process is 0.95.

Figure 1 shows a close approximation to the state value function V^* of an optimal policy for the inverted pendulum domain over the two-dimensional state space $(\theta, \dot{\theta})$. The value function indicates that states which potentially offer high return are clustered within a zone where θ and $\dot{\theta}$ have different signs and therefore the gravity force can be counteracted. Notice the non-linearity of the function and the difficult approximation problem it presents.

Mountain Car

The *mountain car* problem is to drive an underpowered car from the bottom of a valley between two mountains to the top of the mountain on the right (Fig. 2, left). The car is not powerful enough to climb any of the hills directly from the bottom of the valley even at full throttle; it must build some energy by climbing first to the left (moving away from the goal) and then to the right. Three actions are allowed: forward throttle FT (+1), reverse throttle RT (–1), or no throttle NT (0). All three actions are noisy; Gaussian noise with $\mu = 0$ and $\sigma^2 = 0.2$ is added to the chosen action. The state space of the problem is continuous and consists of the position x and the velocity $\dot{x}$ of the car along the horizontal axis. The transitions are governed by the nonlinear dynamics of the system and depend on the current state $(x(t), \dot{x}(t))$ and the current (noisy) control $u(t)$:

$$x(t+1) = \text{BOUND}_x[x(t) + \dot{x}(t+1)]$$

$$\dot{x}(t+1) = \text{BOUND}_{\dot{x}}[\dot{x}(t) + 0.001u(t) - 0.0025\cos(3x(t))],$$

where BOUND_x is a function that keeps x within $[-1.2, 0.5]$, while $\text{BOUND}_{\dot{x}}$ keeps $\dot{x}$ within $[-0.07, 0.07]$. If the car hits the bounds of the position x, the velocity $\dot{x}$

is set to zero. A penalty of −1 is given at each step as long as the position of the car is below the right bound (0.5). As soon as the car position hits the right bound of the position, it has reached the goal; the episode ends successfully and a reward of 0 is given. The discount factor of the process is 0.99.

Figure 2 shows a close approximation to the state value function V^* of an optimal policy for the mountain car domain over the two-dimensional state space $(x, \dot{x})$. The value function indicates that in order to gain high return the car has to follow a spiral in the state space that goes through states with progressively higher value. In practice, this means that the car has to move back and forth between the two mountains until sufficient energy is built to escape from the valley.

Again, notice the high non-linearity of the function and the hard approximation problem it presents.

Definitions

The table summarizes the differences in names and symbols between the common notation (adopted here) and the alternative notation used in the literature.

Common notation		Alternative notation	
Name	Symbol	Symbol	Name
State space	$\mathcal{S}$	S	States
State	s, s'	i, j	State
Action space	$\mathcal{A}$	U	Controls
Action	a	u	Control
Transition model	$\mathcal{P}(s'\|s,a)$	$p_{ij}(u)$	Transition probabilities
Reward function	$\mathcal{R}$	g	Cost function
Discount factor	γ	α	Discount factor
Policy	π	μ	Policy
State value function	V	J	Cost-to-go function
State-action value function	Q	Q	Q-factors
Parameters/ weights	w	r	Parameters
Learning rate	α	γ	Step size

Cross References

- ▶Curse of Dimensionality
- ▶Dynamic Programming
- ▶Feature Selection
- ▶Gaussian Process Reinforcement Learning
- ▶Least-Squares Reinforcement Learning Methods
- ▶Q-Learning; Radial Basis Functions
- ▶Reinforcement Learning
- ▶Temporal Difference Learning
- ▶Value Iteration

Recommended Reading

Brett, B., & How, J. P. (2009). Approximate dynamic programming using Bellman residual elimination and Gaussian process regression. *Proceedings of the American Control Conference*, St. Louis, MO, USA, pp. 745–750.

Brett, B., How, J. P., & Ozdaglar, A. (2008). Approximate dynamic programming using support vector regression. *Proceedings of the IEEE Conference on Decision and Control*, Cancun, Mexico, pp. 745–750.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont: Athena Scientific.

Bradtke, S. J., & Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning, 22*(1–3), 33–57.

Buşoniu, L., Babuška, R., De Schutter, B., & Ernst, D. (2010). *Reinforcement learning and dynamic programming using functions approximators*. CRC Press, Boca Raton, FL, USA.

de Farias, D. P., & Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research, 51*(6), 850–865.

Engel, Y., Mannor, S., & Meir, R. (2003). Bayes meets Bellman: the Gaussian process approach to temporal difference learning. *Proceedings of the International Conference on Machine Learning (ICML)*, Washington, DC, pp. 154–161.

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. *Proceedings of the International Conference on Machine Learning (ICML)*, Bonn, Germany, pp. 201–208.

Ernst, D., Geurts, P., & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research, 6*, 503–556.

Johns, J., Petrik, M., & Mahadevan, S. (2009). Hybrid least-squares algorithms for approximate policy evaluation. *Machine Learning, 76*(2–3), 243–256.

Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research, 4*, 1107–1149.

Mahadevan, S., & Maggioni, M. (2007). Proto-value functions: a Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research, 8*, 2169–2231.

Menache, I., Mannor, S., & Shimkin, N. (2005). Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research, 134*(1), 215–238.

Nedić, A., & Bertsekas, D. P. (2003). Least-squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems: Theory and Applications, 13*(1–2), 79–110.

Parr, R., Painter-Wakefield, C., Li, L., & Littman, M. (2007). Analyzing feature generation for value-function approximation. *Proceedings of the International Conference on Machine Learning (ICML)*, Corvallis, pp. 449–456.

Puterman, M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming*. New York: Wiley.

Rasmussen, C. E., & Kuss, M. (2004). Gaussian processes in reinforcement learning. *Advances in Neural Information Processing Systems (NIPS)*, pp. 751–759.

Sutton, R., & Barto, A. (1998). Reinforcement learning: an introduction. Cambridge: MIT Press.

Taylor, G., & Parr, R. (2009). Kernelized value function approximation for reinforcement learning. *Proceedings of the International Conference on Machine Learning (ICML)*, Toronto, Canada, pp. 1017–1024.

Xu, X., Hu, D., & Lu, X. (2007). Kernel-based least-squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks, 18*(4), 973–992.

Variable Selection

►Feature Selection

Variable Subset Selection

►Feature Selection

Variance

►Bias Variance Decomposition

Variance Hint

►Inductive Bias

VC Dimension

Thomas Zeugmann
Hokkaido University, Sapparo, Japan

Motivation and Background

We define an important combinatorial parameter that measures the combinatorial complexity of a family of subsets taken from a given universe (learning domain) X. This parameter was originally defined by Vapnik and Chervonenkis (1971) and is thus commonly referred to as Vapnik–Chervonenkis dimension, abbreviated as VC *dimension*. Subsequently, Dudley (1978, 1979) generalized Vapnik and Chervonenkis (1971) results. The reader is also referred to Vapnik's (2000) book in which he greatly extends the original ideas. This results in a theory which is called ►structural risk minimization.

The importance of the VC dimension for ►PAC Learning was discovered by Blumer, Ehrenfeucht, Haussler, & Warmuth (1989), who introduced the notion to computational learning theory.

As Anthony and Biggs (1992, p. 71) have put it, "The development of this notion is probably the most significant contribution that mathematics has made to Computational Learning Theory."

Recall that we use $|S|$ and $\wp(S)$ to denote the cardinality and the power set of any set S, respectively. We first define the VC dimension and provide a short explanation of its importance for ►PAC learning. Then we present some examples.

Definition

Let $X \neq \varnothing$ be any learning domain, let $\mathcal{C} \subseteq \wp(X)$ be any nonempty concept class, and let $S \subseteq X$ be any finite set. We set

$$\Pi_{\mathcal{C}}(S) = \{S \cap c \mid c \in \mathcal{C}\}.$$

1. S is said to be *shattered* by $\mathcal{C}$ iff $\Pi_{\mathcal{C}}(S) = \wp(S)$.
2. The VC *dimension* of $\mathcal{C}$ is the cardinality of the largest finite set $S \subseteq X$ that is shattered by $\mathcal{C}$.

If arbitrary large finite sets S are shattered by $\mathcal{C}$, then the VC dimension of $\mathcal{C}$ is defined to be infinite.
Notation: By $\mathrm{VC}(\mathcal{C})$ we denote the VC *dimension* of $\mathcal{C}$.

Remarks

As far as ►PAC Learning is concerned, for a sample set S, the notion $\Pi_{\mathcal{C}}(S)$ has the following meaning. Essentially, $\Pi_{\mathcal{C}}(S)$ collects the set of *all subsets* of the sample set S which are made positive by some concept $c \in \mathcal{C}$. Consequently, $S \cap c$ represents the elements of S that are labeled as to be positive by the concept c. Hence, $\Pi_{\mathcal{C}}(S)$ is the collection of all such subsets taken over all $c \in \mathcal{C}$. If *every* subset of S can be labeled as to be positive by some

concept $c \in \mathcal{C}$ and c does not make any other element of S positive, then S is shattered.

If $\mathrm{VC}(\mathcal{C}) = d$ then there *exists* a finite set $S \subseteq X$ such that $|S| = d$, and S is shattered by $\mathcal{C}$. Moreover, *every* set $S \subseteq X$ with $|S| > d$ is *not* shattered by $\mathcal{C}$.

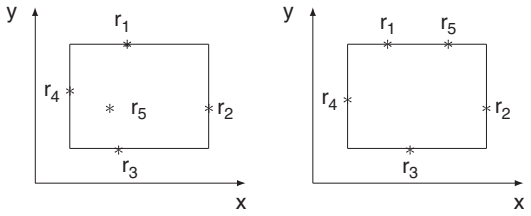
It is intuitively clear that an infinite VC dimension might enormously complicate learning. On the other hand, it is by no means obvious that a finite VC dimension may always guarantee the learnability of the corresponding concept class. However, this is a central theorem of the ►PAC Learning theory. Moreover, the value of the VC dimension is a measure of the sample complexity. This holds for PAC Learning and beyond. Further models where this is true comprise the ►Online Learning models (cf. Haussler, Littlestone, & Warmuth (1994), Maass and Turán (1990); Littlestone (1988), models of Query Based Learning (cf. Maass and Turán, 1990), and others.

Examples

First, let $\mathcal{C}$ be any finite concept class. Then, since it requires 2^d distinct concepts to shatter a set of cardinality d, no set of cardinality larger than $\log|\mathcal{C}|$ can be shattered. Thus, $\log|\mathcal{C}|$ is always an upper bound for the VC dimension of finite concept classes. Here log denotes the logarithm to the base 2.

However, if the VC dimension can be determined, it usually gives a better bound than $\log|\mathcal{C}|$. To see this, let $\mathcal{L}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2 \ldots, x_n, \bar{x}_n\}$, $n \geq 1$ be a set of literals and let $X = \{0,1\}^n$ be the n-dimensional Boolean learning domain. Furthermore, let $\mathcal{C}_n \subseteq \wp(X)$ be the class of all concepts describable by a monomial, including the empty monomial (representing $\{0,1\}^n$) and the conjunction of all literals (representing $\varnothing$). Then $|\mathcal{C}_n| = 3^n + 1$ and thus $\mathrm{VC}(\mathcal{C}) \leq n(\log 3) + 1$. But $\mathrm{VC}(\mathcal{C}_n) = n$ for all $n \geq 2$ and $\mathrm{VC}(\mathcal{C}_1) = 2$ as shown by Natschläger and Schmitt (1996). Note that the same is true for the class of all concepts describable by *monotone monomials*, i.e., monomials containing only non-negated literals.

Next, we consider the concept class $\mathcal{C}$ of all axis-parallel rectangles. So let $X = \mathbb{E}^2$ be the two-dimensional Euclidean space and $\mathcal{C} \subseteq \wp(\mathbb{E}^2)$ be the set of all axis-parallel rectangles, i.e., products of intervals on the x-axis with intervals on the y-axis. Then, it is not hard to see that $\mathrm{VC}(\mathcal{C}) = 4$.



VC Dimension. Figure 1. No set of cardinality 5 can be shattered by axis-parallel rectangles

Clearly, we can shatter the empty set and sets of cardinality 1, 2, and 3. Now, let $S = \{r_1, r_2, r_3, r_4\}$ be such that r_1, r_2, r_3, r_4 are the middle points of the sides of some square. Then it is not hard to see that there are 16 concepts c_i, $1 \leq i \leq 16$, in $\mathcal{C}$ such that $\wp(S) = \{S \cap c_i \mid 1 \leq i \leq 16\}$. Hence, $\mathrm{VC}(\mathcal{C}) \geq 4$.

Next, let $S = \{r_1, r_2, r_3, r_4, r_5\}$ be any set of five pairwise different points. Let c be the smallest closed axis-parallel rectangle containing the points of S. Since c has only four sides, there must be some point $r \in S$, say r_5, such that r_5 lies either in the interior of c or r_5 lies on some side of c along with another point of S (cf. Fig. 1). Suppose S is shattered by $\mathcal{C}$. Then, there has to be a concept $c \in \mathcal{C}$ such that $\{r_1, r_2, r_3, r_4\} = S \cap c$. However, by construction we obtain that $\{r_1, r_2, r_3, r_4\} = S \cap c$ implies $r_5 \in S \cap c$, a contradiction. Thus, *no* set of cardinality 5 is shattered. Hence, $\mathrm{VC}(\mathcal{C}) = 4$.

The latter result can be easily generalized. Let $X = \mathbb{E}^n$, and let $\mathcal{C}$ be the set of all axis-parallel parallelepipeds in $\mathbb{E}^n$. Then $\mathrm{VC}(\mathcal{C}) = 2n$.

A further generalization is as follows. Let X be the real line (one-dimensional Eucleadean space), i.e., $X = \mathbb{E}$, and let $\mathcal{C}$ be the set of all unions of at most s (closed or open) intervals for some fixed constant $s \geq 1$. Let $S = \{x_i \mid 1 \leq i \leq 2s,\ x_i < x_{i+1} \text{ for all } 1 \leq i < 2s\}$. Then one easily verifies that S is shattered by $\mathcal{C}$. Hence, $\mathrm{VC}(\mathcal{C}) \geq 2s$. On the other hand, if S is any set of $2s + 1$ pairwise different points with $x_i < x_{i+1}$ for all $1 \leq i \leq 2s$, then no concept in $\mathcal{C}$ contains $x_1, x_3, \ldots, x_{2s+1}$ without also containing a point in $x_2, x_4, \ldots, x_{2s}$. Thus, no such S is shattered. Consequently, $\mathrm{VC}(\mathcal{C}) = 2s$.

Furthermore, we can generalize the observations made above by deriving some rules that turn out to be very useful to estimate the VC dimension of more complicated concept classes, provided they can be constructed from simpler classes.

First, let $\mathcal{C}_1$ and $\mathcal{C}_2$ be concept classes such that $\mathcal{C}_1 \subseteq \mathcal{C}_2$. Then we clearly have

$$\mathrm{VC}(\mathcal{C}_1) \leq \mathrm{VC}(\mathcal{C}_2).$$

Second, let X be any learning domain, let $\mathcal{C} \subseteq \wp(X)$ and define the complement of $\mathcal{C}$ to be $\overline{\mathcal{C}} = \{X \backslash c \mid c \in \mathcal{C}\}$. Then we have

$$\mathrm{VC}(\overline{\mathcal{C}}) = \mathrm{VC}(\mathcal{C}).$$

Third, consider two concept classes $\mathcal{C}_1$ and $\mathcal{C}_2$ defined over the same learning domain X. Let $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ be the union of $\mathcal{C}_1$ and $\mathcal{C}_2$. Then,

$$\mathrm{VC}(\mathcal{C}) \leq \mathrm{VC}(\mathcal{C}_1) + \mathrm{VC}(\mathcal{C}_2) + 1.$$

Fourth, let $\mathcal{C}$ be any concept class such that $\mathrm{VC}(\mathcal{C}) = d$. Consider the $\mathcal{C}_s$ union (or intersection) of at most s concepts from $\mathcal{C}$, where $s \geq 1$ is any fixed constant, i.e., $\mathcal{C}_s = \{c \mid c = \bigcup_{1 \leq i \leq s} c_i,\ c_i \in \mathcal{C}\}$ (or $\mathcal{C}_s = \{c \mid c = \bigcap_{1 \leq i \leq s} c_i,\ c_i \in \mathcal{C}\}$). Then one can show that

$$\mathrm{VC}(\mathcal{C}_s) \leq 2ds \cdot \log(3s).$$

Numerous further examples can be found in, e.g., Vapnik and Chervonenkis (1974), Haussler and Welz (1987), Anthony and Bartlett (1999), Wenocur and Dudley (1981), Karpinski and Werther (1994), Karpinski and Macintyre (1995), Sakurai (1995), and Mitchell, Scheffer, Sharma, & Stephan (1999).

Applications

Let us return to the notion $\Pi_{\mathcal{C}}(S)$ and generalize it a bit as follows. For any natural number $m \in \mathbb{N}$ and any nonempty concept class $\mathcal{C} \subseteq \wp(S)$, we set:

$$\Pi_{\mathcal{C}}(m) = \max\{|\Pi_{\mathcal{C}}(S)| \mid S \subseteq X,\ |S| = m\}.$$

We can use the new notion to give an equivalent definition of the VC dimension of a concept class $\mathcal{C}$, i.e.,

$$\mathrm{VC}(\mathcal{C}) = \max\{d \mid d \in \mathbb{N},\ \Pi_{\mathcal{C}}(d) = 2^d\}.$$

Looking at $\Pi_{\mathcal{C}}(m)$ from the perspective of learning, we see the following. The argument m refers to the sample size. $\Pi_{\mathcal{C}}(m)$ is describing the maximum number of ways a sample of size m can be labeled by concepts taken from $\mathcal{C}$. Hence, the number $\Pi_{\mathcal{C}}(m)$ behaves as a measure of concept class complexity. What can be said about $\Pi_{\mathcal{C}}(m)$? Suppose, $d = VC(\mathcal{C})$; then $m \leq d$ implies $\Pi_{\mathcal{C}}(m) = 2^m$. On the other hand, $m > d$ directly implies $\Pi_{\mathcal{C}}(m) < 2^m$. Therefore, we are interested in learning how fast $\Pi_{\mathcal{C}}(m)$ really grows provided $m > d$. The key ingredient to obtain the desired information is usually referred to as Sauer's Lemma Sauer (1972). Under the assumptions made above, it states that

$$\Pi_{\mathcal{C}}(m) \leq \sum_{i=0}^{d} \binom{m}{i}, \quad \textit{where} \quad \binom{m}{i} = 0 \quad \textit{if } i > m.$$

Like many important results, Sauer's Lemma Sauer (1972) has several proofs and generalizations have been studied, too. We refer the reader to Anthony and Biggs (1992), Kearns and Vazirani (1994), and Gurvits (1997) for a more detailed exposition.

Let us first look at the case $m \leq d$ already considered. For this case, Sauer's Lemma is telling us that

$$\Pi_{\mathcal{C}}(m) \leq \sum_{i=0}^{d} \binom{m}{i} = 2^m,$$

and thus, we get an exponential bound. The interesting aspect is that in the remaining cases the bound is *polynomial*. For simplifying notation, we set

$$\Phi(d, m) = \sum_{i=0}^{d} \binom{m}{i}.$$

Using combinatorial arguments and Stirling approximation, one can show that

1. $\Phi(0, m) = \binom{m}{0} = 1$ for all $m \in \mathbb{N}$.
2. $\Phi(d, 1) = \binom{1}{0} + \binom{1}{1} = 2$ for all $d \in \mathbb{N}$, $d \geq 1$.
3. $\Phi(d, m) = \Phi(d, m-1) + \Phi(d-1, m-1)$ for all $d, m \in \mathbb{N}$, $d \geq 1$, $m \geq 2$.
4. $\Phi(d, m) \leq m^d + 1$ for all $d \geq 0$, $m \geq 0$.
5. $\Phi(d, m) \leq m^d$ for all $d \geq 2$, $m \geq 2$.
6. $\Phi(d, m) \leq \left(\frac{em}{d}\right)^d$ for all $m \geq d \geq 1$.

That is, (4) through (6) provide a bound polynomial in m for $\Pi_{\mathcal{C}}(m)$ whenever $\mathrm{VC}(\mathcal{C})$ is finite. This insight is fundamental for ▶PAC Learning and other learning models.

Finally, we refer the reader to Schaefer (1999), who has determined the complexity of computing the VC dimension and to Goldberg and Jerrum (1995), who succeeded in bounding the VC dimension of concept classes parameterized by real numbers.

Cross References

►Epsilon Nets
►PAC Learning
►Statistical Machine Learning
►Structural Risk Minimization

Recommended Reading

Anthony, M., & Bartlett, P. L. (1999). *Neural network learning: Theoretical foundations*. Cambridge: Cambridge University Press.

Anthony, M., & Biggs, N. (1992). *Computational learning theory. Cambridge tracts in theoretical computer science* (No. 30). Cambridge: Cambridge University Press.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik–Chervonenkis dimension. *Journal of the ACM, 36*(4), 929–965.

Dudley, R. M. (1978). Central limit theorems for empirical measures. *Annals of Probability, 6*(6), 899–929.

Dudley, R. M. (1979). Corrections to "Central limit theorems for empirical measures". *Annals of Probability, 7*(5), 909–911.

Goldberg, P. W., & Jerrum, M. R. (1995). Bounding the Vapnik–Chervonenkis dimension of concept classes parameterized by real numbers. *Machine Learning, 18*(2–3), 131–148.

Gurvits, L. (1997). Linear algebraic proofs of VC-dimension based inequalities. In S. Ben-David (Ed.), *Computational learning theory, third European conference, EuroCOLT '97, Jerusalem, Israel, March 1997, Proceedings, Lecture notes in artificial intelligence* (Vol. 1208, pp. 238–250). Springer.

Haussler, D., & Welz, E. (1987). Epsilon nets and simplex range queries. *Discrete & Computational Geometry, 2*, 127–151.

Haussler, D., & Littlestone, N., & Warmuth, M. K. (1994). Predicting f0; 1g functions on randomly drawn points. *Information and Computation, 115*(2), 248–292.

Karpinski, M., & Macintyre, A. (1995). Polynomial bounds for VC dimension of sigmoidal neural networks. In *Proceedings of twenty-seventh annual ACM symposium on theory of computing* (pp. 200–208). New York: ACM Press.

Karpinski, M., & Werther, T. (1994). VC dimension and sampling complexity of learning sparse polynomials and rational functions. In S. J. Hanson, G. A. Drastal, and R. L. Rivest (Eds.), *Computational learning theory and natural learning systems, Vol. I: Constraints and prospects* (Chap. 11, pp. 331–354). Cambridge, MA: MIT Press.

Kearns, M. J., & Vazirani, U. V. (1994). *An introduction to computational learning theory*. Cambridge, MA: MIT Press.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning, 2*(4), 285–318.

Maass, W., & Turan, G. (1990). On the complexity of learning from counterexamples and membership queries. In *Proceedings of the thirty-first annual symposium on Foundations of Computer Science (FOCS 1990)*, St. Louis, Missouri, October 22–24, 1990 (pp. 203–210). Los Alamitos, CA: IEEE Computer Society Press.

Mitchell, A., Scheffer, T., Sharma, A., & Stephan, F. (1999). The VC-dimension of subclasses of pattern languages. In O. Watanabe & T. Yokomori (Eds.), *Algorithmic learning theory, tenth international conference, ALT'99, Tokyo, Japan, December 1999, Proceedings, Lecture notes in artificial intelligence* (Vol. 1720, pp. 93–105). Springer.

Natschläger, T., & Schmitt, M. (1996). Exact VC-dimension of Boolean monomials. *Information Processing Letters, 59*(1), 19–20.

Sakurai, A. (1995). On the VC-dimension of depth four threshold circuits and the complexity of Boolean-valued functions. *Theoretical Computer Science, 137*(1), 109–127. Special issue for ALT '93

Sauer, N. (1972). On the density of families of sets. *Journal of Combinatorial Theory (A), 13*(1), 145–147.

Schaefer, M. (1999). Deciding the Vapnik–Červonenkis dimension is Σ_3^p-complete. *Journal of Computer System Sciences, 58*(1), 177–182.

Vapnik, V. N. (2000). *The nature of statistical learning theory*, (2nd ed.). Berlin: Springer.

Vapnik, V. N., & Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probabability and Its Applications, 16*(2), 264–280.

Vapnik, V. N., & Chervonenkis, A. Y. (1974). *Theory of pattern recognition*. Moskwa: Nauka (in Russian).

Wenocur, R. S., & Dudley, R. M. (1981). Some special Vapnik–Chervonenkis classes. *Discrete Mathematics, 33*, 313–318.

Vector Optimization

►Multi-Objective Optimization

Version Space

Claude Sammut
The University of New South Wales,
Sydney, Australia

Definition

Mitchell (1977, 1982) defines the *version space* for a learning algorithm as the subset of hypotheses consistent with the training examples. That is, the ►hypothesis language is capable of describing a large, possibly infinite, number of concepts. When searching for the target concept, we are only interested in the subset of sentences in the hypothesis language that are consistent

with the training examples, where consistent means that the examples are correctly classified (assuming deterministic concepts and no ►noise in the data). While the version space may be infinite, it can often be represented in a compact manner by maintaining only its *bounds*, the ►most specific (►Most Specific Hypothesis) and ►most general hypotheses. Any hypothesis that is more general than a hypothesis in the most specific bound and more specific than a hypothesis in the most general bound is in the version space.

Cross References

►Learning as Search
►Noise

Recommended Reading

Mitchell, T. M. (1977). Version Spaces: A candidate elimination approach to rule-learning (pp. 305–310). In *Proceedings of the fifth international joint conference on artificial intelligence, Cambridge.*

Mitchell, T. M. (1982). Generalization as Search. *Artificial Intelligence*, *18*(2), 203–226.

Viterbi Algorithm

A dynamic programming algorithm for finding the most likely sequence of hidden states resulting in an observed sequence of output events. The most likely sequence is called the Viterbi path. The Viterbi algorithm was popularized due to its usability in Hidden Markov models (HMM).

The Viterbi algorithm was initially proposed by Andrew Viterbi as an error-correction scheme for noisy digital communication links. It is now also commonly used in speech recognition, natural language processing, and bioinformatics.

Recommended Reading

Viterbi, A.J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory 13*(2), 260–269.

W

Web Advertising

►Text Mining for Advertising

Weight

Risto Miikkulainen
The University of Texas at Austin
Austin, TX, USA

Synonyms

Connection strength; Synaptic efficacy

Definition

In a ►neural network, connections between neurons typically have weights that indicate how strong the connection is. The neuron computes by forming a weighted sum of its input, i.e., the activation of each input neuron is multiplied by the corresponding connection weight. Adapting such weights is the most important way of learning in neural networks. Connection weights are loosely modeled after the synaptic efficacies in biological neurons, where they determine how large a positive or negative change in the membrane potential each input spike generates (see ►Biological Learning). In most models, all connection parameters are abstracted into a weight: attenuation or interaction of the potentials and connection delays are usually not taken into account. The weights are usually real-valued numbers $(-\infty .. \infty)$, although in some algorithms, intended for VLSI implementation, the range and precision of these values can be restricted (or weights eliminated altogether). Weights in some methods can be restricted to positive values if the inputs are known to be positive and the method is based on comparing the similarity to the weights (as in e.g., ►Self-Organizing Maps, ►Adaptive Resonance Theory, and ►Radial Basis Function networks). Most learning methods are based on adjusting the weight values. The weights are often initialized to small random values, although if enough is known about the input space and the task, more systematic initialization can improve performance significantly. The weights are then adjusted based on local information that is available on either side of the connection. Usually, only small modifications are made in each learning step to avoid disrupting what the network already knows, and learning converges over time to a setting of values that solves the task.

Within-Sample Evaluation

►In-Sample Evaluation

Word Sense Disambiguation

Rada Mihalcea
University of North Texas
Denton, TX, USA

Synonyms

Learning word senses; Solving semantic ambiguity

Definition

Ambiguity is inherent to human language. In particular, word sense ambiguity is prevalent in all natural languages, with a large number of the words in any given language carrying more than one meaning. For instance, the English noun *plant* can mean *green plant* or *factory*; similarly the French word *feuille* can mean *leaf* or *paper*. The correct sense of an ambiguous word can be selected based on the context where it occurs, and correspondingly the problem of word

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

sense disambiguation is defined as the task of automatically assigning the most appropriate meaning to a polysemous word within a given context.

Motivation and Background

Word sense disambiguation is considered one of the most difficult problems in natural language processing, due to the high semantic ambiguity that is typically associated with language. It was first noted as a problem in the context of machine translation, when Warren Weaver, in his famous 1949 memorandum, pointed out word ambiguity as one of the problems that needed to be solved in order to enable automatic translations between the languages of the world (Weaver, 1995). More than 50 years later, word sense ambiguity is still regarded as an important and difficult research problem, and it has been demonstrated to have a potentially significant impact on several natural language processing applications.

Applications

In addition to machine translation, the role of word sense disambiguation has also been explored in connection to other applications, such as monolingual information retrieval, cross-language information retrieval, question answering, knowledge acquisition, information extraction, text classification, and others. In particular, a significant amount of work has been carried out in areas related to information retrieval, where the resolution of word ambiguity has been shown to have an impact on both the precision of the system (by allowing for matches only between identical word meanings in the query and in the documents), as well as the recall of the system (by performing query expansion using synonyms of selected word meanings).

Brief History

Over the years, the field of word sense disambiguation has undergone steady improvements in both quality and scope, moving from the rule-based systems using hand crafted knowledge that were popular in the 1970s and 1980s, to the more advanced corpus-based methods used in the 1990s, and to the current hybrid systems that rely on a mix of knowledge-based and corpus-based resources, minimizing the need of sense annotated data and taking advantage of the Web. The shift from small-scale rule-based systems to large-scale data-driven methods has also implied an increase in coverage, with early systems typically addressing a handful of ambiguous words for which hand-coded rules were available, while many of the current systems have the ability to address all or almost all content words in unrestricted text.

Methods

Current word sense disambiguation systems are divided into three main categories:

Knowledge-based: These systems rely mainly on information drawn from lexical resources, such as dictionaries or thesauruses. The Lesk algorithm (Lesk, 1986) is one of the most well-known knowledge-based word sense disambiguation methods. It decides the meaning of a word based on a measure of similarity among the definitions provided by a dictionary. For instance, for the phrase *pine cone*, the algorithm will select the meaning of *kind of evergreen tree* for *pine*, and *fruit of evergreen tree* for *cone*, as these are the definitions with the highest lexical overlap among all the possible definitions provided by a dictionary.

Unsupervised corpus-based: These approaches typically consist of algorithms for clustering word sense occurrences in a corpus, without making explicit reference to a sense inventory. The clustering can be performed in a monolingual environment, in which case different word occurrences are represented by features derived from their immediate context (Schutze, 1998). Alternatively, a clustering of word senses can also be performed using cross-lingual evidence drawn from the translations observed in a parallel corpus (Ng, Wang, & Chan, 2003). This line of work is often referred to as word sense discrimination, as the word meanings are not disambiguated against a sense inventory, but are discriminated against each other.

Supervised corpus-based: These methods are the focus of the current chapter, and they consist primarily of machine learning algorithms applied on large sense-annotated corpora. Supervised algorithms have been typically applied to one word at a time, although experiments have also been carried out for their application to all words in unrestricted text. While sense-annotated corpora have usually been constructed by hand, recent work has also explored various approaches for the automatic generation of such data, which has

been used successfully in conjunction with machine learning algorithms.

Structure of the Learning System

Among the various knowledge-based and data-driven word sense disambiguation methods that have been proposed to date, supervised systems have been constantly observed as leading to the highest performance. In these systems, the sense disambiguation problem is formulated as a supervised learning task, where each sense-tagged occurrence of a particular word is transformed into a feature vector, which is then used in an automatic learning process.

Given a target word and a set of examples where this word occurs, each occurrence being annotated with the correct sense, a supervised system will attempt to learn how to automatically annotate occurrences of the given word in new, previously unseen, contexts. This process is accomplished in two steps. First, representative features are extracted from the context of the ambiguous word; this step is applied to the annotated examples (training) as well as the unlabeled examples (test). Second, a machine learning algorithm is applied on the feature vectors, and consequently the most likely sense is assigned to the test occurrences of the target word.

Features

Research in supervised word sense disambiguation has considered two main types of featutlllrestomodel occurrences of ambiguous words:

Contextual features, which are extracted tfrom the immediate vicinity of the ambiguous word. These features usually consist of the words before and after the target word (a window size of 3–10 words is typical), their parts of speech, words in a syntactic dependency with the target word (e.g., the subject of the verb, the noun modified by an adjective), position in the sentence, and the like. For instance, the adjective *green* could be one of the contextual features extracted from the context *the green plant* for the ambiguous word *plant*.

Topical features, which are represented by the words most frequently co-occurring with a given meaning of the target word. These words are usually determined by counting the number of times each word occurs in the context of a word meaning, divided by the total number of occurrences in the context of the word regardless of its meaning. For instance, the *factory* meaning of *plant* could have topical features such as *industrial* and *work*, whereas the *green plant* meaning of *plant* might have features such as *animal* and *water*.

As an example of feature vector construction, consider the following two contexts provided for the ambiguous word *plant*:

The/det growth/noun of/prep a/det seedling/noun into/prep a/det flowering/adj **plant**/noun helps/verb children/noun investigate/verb the/det conditions/noun that/prep plants/noun need/verb for/prep growth/noun.

The/det operations/noun staff/noun in/prep an/det industrial/adj **plant**/noun is/verb typically/adv measured/verb in/prep asset/noun utilization/noun.

The following two feature vectors are constructed:

Machine Learning

Provided a set of feature vectors representing different occurrences of an ambiguous target word, the goal of the machine learning system is to learn how to predict the most likely sense for a new occurrence. The word sense disambiguation literature describes experiments with a large number of machine learning algorithms, including decision lists (Yarowsky, 2000), instance-based learning (Ng & Lee, 1996), Naïve Bayes and decision trees (Pedersen, 1998), support vector machines (Lee & Ng, 2002), and others. A comparison of several machine learning algorithms for word sense disambiguation is provided in Lesk (1986) and Mooney (1996).

W-1	W + 1	P-1	P+1	Growth	Flowering	Industrial	Staff	Sense
Flowering	Helps	Adj	Verb	Y	Y	N	N	Green plant
Industrial	Is	Adj	Verb	N	N	Y	Y	Factory

Generation of Sense-Tagged Corpora

One of the main drawbacks associated with the supervised methods for word sense disambiguation is the cost incurred in the process of building sense-tagged corpora. Despite their high performance, the applicability of these supervised systems is limited to those few words for which sense-tagged data is available, and their accuracy is strongly connected to the amount of labeled data available at hand.

Sense annotations have been typically carried out by humans, which resulted in several publicly available data sets, such as those made available during the Senseval evaluations (http://www.senseval.org). However, despite the effort that went into the construction of these data sets, their applicability is limited to a handful of approximately 100 ambiguous words.

To address the sense-tagged data bottleneck problem, different methods for automatic sense-tagged data annotation have been proposed in the past, with various degrees of success. One such method relies on monosemous relatives extracted from dictionaries, which can be used to identify ambiguity-free occurrences in large corpora (Leacock, Chodorow, & Miller, 1998; Mihalcea, 1999). Another method relies on automatically bootstrapped disambiguation patterns, which can be used to generate a large number of sense-tagged examples (Mihalcea, 2002; Yarowsky, 1995). The use of volunteer contributors to create sense-annotated corpora has also been explored in the Open Mind Word Expert system (Chklovski and Mihalcea, 2002). Finally, in recent work, Wikipedia was identified as a rich source of word sense annotations, which can be used to build supervised word sense disambiguation systems (Mihalcea, 2007).

Cross References

►Semi-Supervised Text Processing

Recommended Reading

Agirre, E., & Edmonds, P. (2006). *Word sense disambiguation: Algorithms and applications*. Berlin: Springer. http://www.wsdbook.org

Chklovski, T., & Mihalcea, R. (2002). Building a sense tagged corpus with open mind word expert. In *Proceedings of ACL 2002 workshop on WSD*. Philadelphia, PA.

Leacock, C., Chodorow, M., & Miller, G. A. (1998). Using corpus statistics and wordnet relations for sense identification. *Computational Linguistics*, *24*(1), 147–165.

Lee, Y. K., & Ng, H. T. (2002). An empirical evaluation of knowledge sources and learning algorithms for word sense disambiguation. In *Proceedings of EMNLP 2002*. Philadelphia, PA.

Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *SIGDOC 1986*. Toronto, ON, Canada.

Mihalcea, R. (1999). An automatic method for generating sense tagged corpora. In *Proceedings of AAAI 1999*. Orlando, FL.

Mihalcea, R. (2002). Bootstrapping large sense tagged corpora. In *Proceedings of LREC 2002*. Las Palmas, Spain.

Mihalcea, R. (2007). Using wikipedia for automatic word sense disambiguation. In *Proceedings of NAACL 2007*. Rochester, NY.

Mihalcea, R., & Pedersen, T. (2005). Advances in word sense disamhbiguation. Tutorial presented at IBERAMIA 2004, ACL 2005, AAAI 2005. http://www.d.umn.edu/~tpederse/WSDTutorial.html

Mooney, R. (1996). Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning. In *Proceedings of EMNLP*. Philadelphia, PA.

Ng, H. T. & Lee, H. B. (1996). Integrating multiple knowledge sources to disambiguate word sense: An examplar-based approach. In *Proceedings of ACL*. Santa Cruz, CA.

Ng, H. T., Wang, B., & Chan, Y. S. (2003). Exploiting parallel texts for word sense disambiguation: an empirical study. In *Proceedings of ACL*. Sapporo, Japan.

Pedersen, T. (1998). *Learning probabilistic models of word sense disambiguation*. Ph.D. Dissertation. Southern Methodist University.

Schutze, H. (1998). Automatic word sense discrimination. *Computational Linguistics*, *24*(1), 97–123.

Weaver, W. (1995). Translation. In W. N. Locke, & A. D. Booth (Eds.), *Machine translation of languages: Fourteen essays*. Cambridge, MA: MIT Press.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of ACL*. Cambridge, MA.

Yarowsky, D. (2000). Hierarchical decision lists for word sense disambiguation. *Computers and the Humanities*, *34*(1–2), 179–186.

Word Sense Discrimination

Word sense discrimination is sometimes used as a synonym for ►word sense disambiguation. Note, however, that these two terms refer to somewhat different problems, as word sense discrimination implies a distinction between different word meanings in a corpus (without reference to a sense inventory), whereas word sense disambiguation refers to a sense assignment using a given sense inventory.

Z

Zero-One Loss

Zero-one loss is a common ▶loss function used with ▶classification learning. It assigns 0 to loss for a correct classification and 1 for an incorrect classification.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,