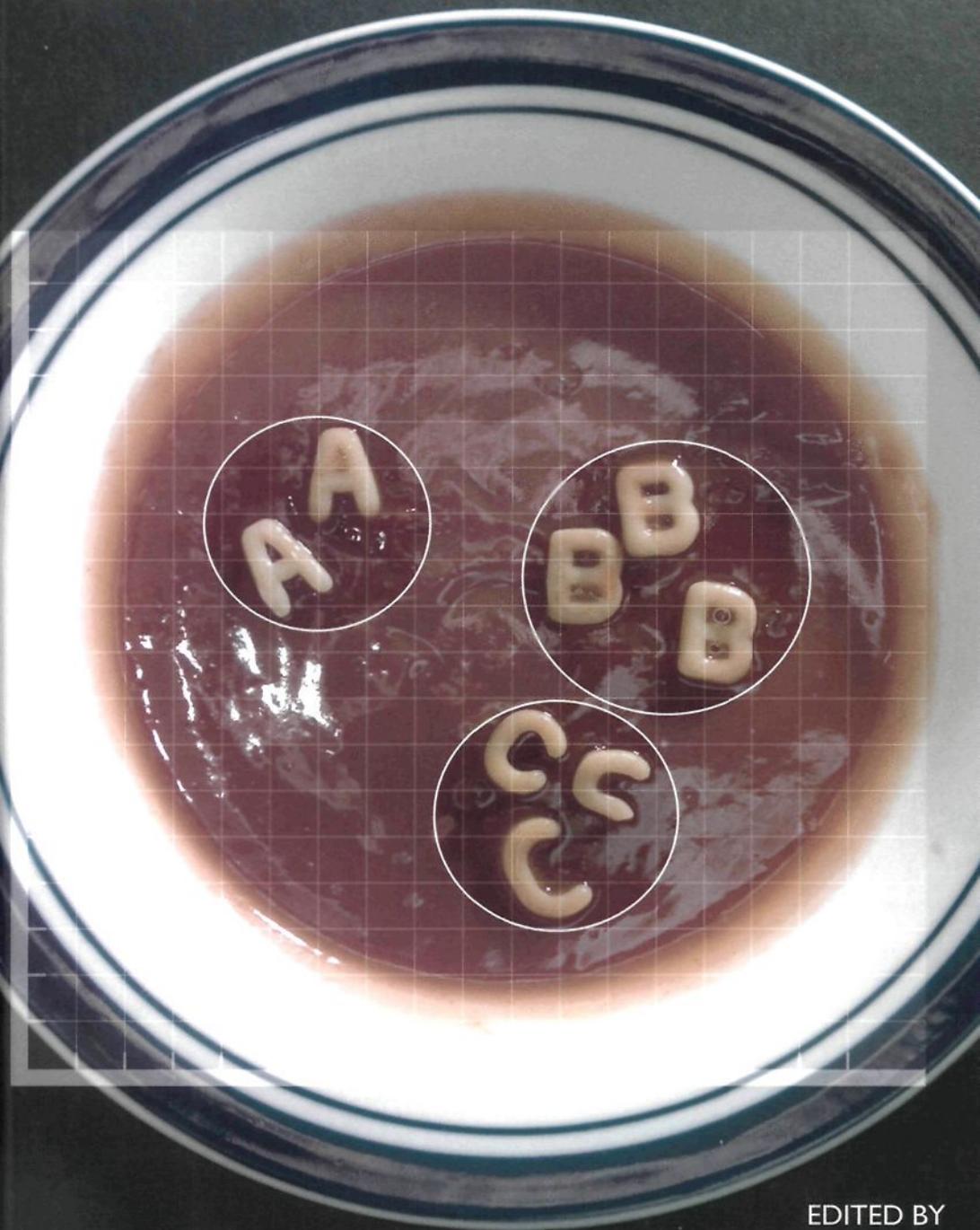


# LECTURE NOTES IN DATA MINING



EDITED BY  
MICHAEL W. BERRY · MURRAY BROWNE

LECTURE NOTES IN  
**DATA MINING**

This page is intentionally left blank

# LECTURE NOTES IN DATA MINING

EDITED BY

MICHAEL W. BERRY  
MURRAY BROWNE

*UNIVERSITY OF TENNESSEE, USA*

 World Scientific

NEW JERSEY • LONDON • SINGAPORE • BEIJING • SHANGHAI • HONG KONG • TAIPEI • CHENNAI

*Published by*

World Scientific Publishing Co. Pte. Ltd.

5 Toh Tuck Link, Singapore 596224

*USA office:* 27 Warren Street, Suite 401-402, Hackensack, NJ 07601

*UK office:* 57 Shelton Street, Covent Garden, London WC2H 9HE

**British Library Cataloguing-in-Publication Data**

A catalogue record for this book is available from the British Library.

**LECTURE NOTES IN DATA MINING**

Copyright © 2006 by World Scientific Publishing Co. Pte. Ltd.

*All rights reserved. This book, or parts thereof, may not be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or any information storage and retrieval system now known or to be invented, without written permission from the Publisher.*

For photocopying of material in this volume, please pay a copying fee through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA. In this case permission to photocopy is not required from the publisher.

ISBN 981-256-802-6

## PREFACE

The explosion of information technology, which continues to expand data-driven markets and business, has made data mining an even more relevant topic of study. Books on data mining tend to be either broad and introductory or focus on some very specific technical aspect of the field.

*Lecture Notes in Data Mining* is a series of seventeen “written lectures” that explores in depth the core of data mining (classification, clustering and association rules) by offering overviews that include both analysis and insight. Written by graduate students from the University of Tennessee as part of an Advancements in Data Mining course, *Lecture Notes* is an ideal companion to either an introductory data mining textbook or a technical data mining book.

The Advancements in Data Mining Course was developed not only to allow graduate students an opportunity to investigate certain topics in data mining, but it was also planned as a vehicle for graduate students to sharpen their writing and presentation skills. After selecting one of the assigned topics, researching, and writing his or her paper, each student was required to make a lengthy presentation to the class. Both the presentation and the paper was extensively peer reviewed and each student was required to make revisions before submitting a final paper. The discussion following the presentations was spirited but respectful, as each student was aware that at some point in the semester — he or she would be facing their peers as well.

The initial chapters of *Lecture Notes* lay a framework of data mining techniques by explaining some of the basics such as applications of Bayes Theorem, similarity measures and decision trees. Before focusing on the pillars of classification, clustering and association rules, the book also provides information about alternative candidates such as point estimation and genetic algorithms.

The book’s discussion of classification includes an introduction to decision tree algorithms, rule-based algorithms (a popular alternative to decision trees) and distance-based algorithms. Five of the lecture-chapters are

devoted to the concept of clustering or unsupervised classification. The functionality of clustering's hierarchical and partitional algorithms is also covered as well as the efficient and scalable cluster algorithms used in large databases. The concept of association rules in terms of basic algorithms, parallel and distributive algorithms and advanced measures that help determine the value of association rules are included in this book too. The final chapter covers spatial mining algorithms.

In addition to the technical descriptions of the algorithms and methods, the students were encouraged to provide simple examples and to provide commentary based on references that could assess both strengths and weaknesses of the respective techniques. This type of analysis gives the student an understanding of what it is to go behind just reporting the particulars of an algorithm and the value of providing additional insights. With this in mind, each student was required to include a complete bibliography of journal articles, books, conference papers, technical reports, electronic journals and web sites for each report. These references have been compiled and appear at the end of the book.

*Lecture Notes* can be used as a supplement that would accompany a data mining textbook for graduate level computer students. With its exploration of data mining's core competencies, *Lecture Notes* also appeals to the database professional who is looking for a concise background and evaluation of the various tools that are now available to data miners.

One of the more common mistakes data miners make is becoming too focused on one area of technique. It's not unusual to become an expert in one area and forget that there are other approaches that merit consideration based on the problem at hand. *Lecture Notes* offers a good sampling of what other techniques are available presented with a fresh set of eyes. Hopefully, these hardworking students will open your senses to the burgeoning world of data mining.

All proceeds from the sales of this book will be donated to the student travel fund in the Department of Computer Science, University of Tennessee, Knoxville.

Michael W. Berry, Murray Browne  
Knoxville, TN  
December 2005

# CONTENTS

<b>Preface</b>	v
<b>1 Point Estimation Algorithms</b>	<b>1</b>
1. Introduction	1
2. Motivation	2
3. Methods of Point Estimation	2
3.1. The Method of Moments	2
3.2. Maximum Likelihood Estimation	4
3.3. The Expectation-Maximization Algorithm	6
4. Measures of Performance	8
4.1. Bias	9
4.2. Mean Squared Error	9
4.3. Standard Error	10
4.4. Efficiency	10
4.5. Consistency	11
4.6. The Jackknife Method	11
5. Summary	13
<b>2 Applications of Bayes Theorem</b>	<b>15</b>
1. Introduction	15
2. Motivation	16
3. The Bayes Approach for Classification	17
3.1. Statistical Framework for Classification	17
3.2. Bayesian Methodology	20
4. Examples	22
4.1. Example 1: Numerical Methods	22
4.2. Example 2: Bayesian Networks	24
5. Summary	25

<b>3</b>	<b>Similarity Measures</b>	<b>27</b>
1.	Introduction . . . . .	27
2.	Motivation . . . . .	28
3.	Classic Similarity Measures . . . . .	28
3.1.	Dice . . . . .	30
3.2.	Overlap . . . . .	30
3.3.	Jaccard . . . . .	31
3.4.	Asymmetric . . . . .	31
3.5.	Cosine . . . . .	31
3.6.	Other Measures . . . . .	32
3.7.	Dissimilarity . . . . .	32
4.	Example . . . . .	33
5.	Current Applications . . . . .	35
5.1.	Multi-Dimensional Modeling . . . . .	35
5.2.	Hierarchical Clustering . . . . .	36
5.3.	Bioinformatics . . . . .	37
6.	Summary . . . . .	38
<b>4</b>	<b>Decision Trees</b>	<b>39</b>
1.	Introduction . . . . .	39
2.	Motivation . . . . .	41
3.	Decision Tree Algorithms . . . . .	42
3.1.	ID3 Algorithm . . . . .	43
3.2.	Evaluating Tests . . . . .	43
3.3.	Selection of Splitting Variable . . . . .	46
3.4.	Stopping Criteria . . . . .	46
3.5.	Tree Pruning . . . . .	47
3.6.	Stability of Decision Trees . . . . .	47
4.	Example: Classification of University Students . . . . .	48
5.	Applications of Decision Tree Algorithms . . . . .	49
6.	Summary . . . . .	50
<b>5</b>	<b>Genetic Algorithms</b>	<b>53</b>
1.	Introduction . . . . .	53
2.	Motivation . . . . .	54
3.	Fundamentals . . . . .	55
3.1.	Encoding Schema and Initialization . . . . .	56
3.2.	Fitness Evaluation . . . . .	57

3.3.	Selection . . . . .	58
3.4.	Crossover . . . . .	59
3.5.	Mutation . . . . .	61
3.6.	Iterative Evolution . . . . .	62
4.	Example: The Traveling-Salesman . . . . .	63
5.	Current and Future Applications . . . . .	65
6.	Summary . . . . .	66
<b>6</b>	<b>Classification: Distance-based Algorithms</b>	<b>67</b>
1.	Introduction . . . . .	67
2.	Motivation . . . . .	68
3.	Distance Functions . . . . .	68
3.1.	City Block Distance . . . . .	69
3.2.	Euclidean Distance . . . . .	70
3.3.	Tangent Distance . . . . .	70
3.4.	Other Distances . . . . .	71
4.	Classification Algorithms . . . . .	72
4.1.	A Simple Approach Using Mean Vector . . . . .	72
4.2.	$K$ -Nearest Neighbors . . . . .	74
5.	Current Applications . . . . .	76
6.	Summary . . . . .	77
<b>7</b>	<b>Decision Tree-based Algorithms</b>	<b>79</b>
1.	Introduction . . . . .	79
2.	Motivation . . . . .	80
3.	ID3 . . . . .	80
4.	C4.5 . . . . .	82
5.	C5.0 . . . . .	83
6.	CART . . . . .	84
7.	Summary . . . . .	85
<b>8</b>	<b>Covering (Rule-based) Algorithms</b>	<b>87</b>
1.	Introduction . . . . .	87
2.	Motivation . . . . .	88
3.	Classification Rules . . . . .	88
4.	Covering (Rule-based) Algorithms . . . . .	90
4.1.	1R Algorithm . . . . .	91
4.2.	PRISM Algorithm . . . . .	94
4.3.	Other Algorithms . . . . .	96

5. Applications of Covering Algorithms . . . . .	97
6. Summary . . . . .	97
<b>9 Clustering: An Overview</b>	<b>99</b>
1. Introduction . . . . .	99
2. Motivation . . . . .	100
3. The Clustering Process . . . . .	100
3.1. Pattern Representation . . . . .	101
3.2. Pattern Proximity Measures . . . . .	102
3.3. Clustering Algorithms . . . . .	103
3.3.1. Hierarchical Algorithms . . . . .	103
3.3.2. Partitional Algorithms . . . . .	105
3.4. Data Abstraction . . . . .	105
3.5. Cluster Assessment . . . . .	105
4. Current Applications . . . . .	107
5. Summary . . . . .	107
<b>10 Clustering: Hierarchical Algorithms</b>	<b>109</b>
1. Introduction . . . . .	109
2. Motivation . . . . .	110
3. Agglomerative Hierarchical Algorithms . . . . .	111
3.1. The Single Linkage Method . . . . .	112
3.2. The Complete Linkage Method . . . . .	114
3.3. The Average Linkage Method . . . . .	116
3.4. The Centroid Method . . . . .	116
3.5. The Ward Method . . . . .	117
4. Divisive Hierarchical Algorithms . . . . .	118
5. Summary . . . . .	120
<b>11 Clustering: Partitional Algorithms</b>	<b>121</b>
1. Introduction . . . . .	121
2. Motivation . . . . .	122
3. Partitional Clustering Algorithms . . . . .	122
3.1. Squared Error Clustering . . . . .	122
3.2. Nearest Neighbor Clustering . . . . .	126
3.3. Partitioning Around Medoids . . . . .	127
3.4. Self-Organizing Maps . . . . .	131

4. Current Applications . . . . .	132
5. Summary . . . . .	132
<b>12 Clustering: Large Databases</b>	<b>133</b>
1. Introduction . . . . .	133
2. Motivation . . . . .	134
3. Requirements for Scalable Clustering . . . . .	134
4. Major Approaches to Scalable Clustering . . . . .	135
4.1. The Divide-and-Conquer Approach . . . . .	135
4.2. Incremental Clustering Approach . . . . .	135
4.3. Parallel Approach to Clustering . . . . .	136
5. BIRCH . . . . .	137
6. DBSCAN . . . . .	139
7. CURE . . . . .	140
8. Summary . . . . .	141
<b>13 Clustering: Categorical Attributes</b>	<b>143</b>
1. Introduction . . . . .	143
2. Motivation . . . . .	144
3. ROCK Clustering Algorithm . . . . .	145
3.1. Computation of Links . . . . .	146
3.2. Goodness Measure . . . . .	147
3.3. Miscellaneous Issues . . . . .	148
3.4. Example . . . . .	148
4. COOLCAT Clustering Algorithm . . . . .	149
5. CACTUS Clustering Algorithm . . . . .	151
6. Summary . . . . .	152
<b>14 Association Rules: An Overview</b>	<b>153</b>
1. Introduction . . . . .	153
2. Motivation . . . . .	154
3. Association Rule Process . . . . .	154
3.1. Terminology and Notation . . . . .	154
3.2. From Data to Association Rules . . . . .	157
4. Large Itemset Discovery Algorithms . . . . .	158
4.1. Apriori . . . . .	158
4.2. Sampling . . . . .	160
4.3. Partitioning . . . . .	162
5. Summary . . . . .	163

**15 Association Rules: Parallel and Distributed Algorithms** **169**

- 1. Introduction . . . . . 169
- 2. Motivation . . . . . 170
- 3. Parallel and Distributed Algorithms . . . . . 171
  - 3.1. Data Parallel Algorithms on Distributed Memory Systems . . . . . 172
    - 3.1.1. Count Distribution (CD) . . . . . 172
  - 3.2. Task Parallel Algorithms on Distributed Memory Systems . . . . . 174
    - 3.2.1. Data Distribution (DD) . . . . . 174
    - 3.2.2. Candidate Distribution (CaD) . . . . . 174
    - 3.2.3. Intelligent Data Distribution (IDD) . . . . . 175
  - 3.3. Data Parallel Algorithms on Shared Memory Systems . . . . . 176
    - 3.3.1. Common Candidate Partitioned Database (CCPD) . . . . . 176
  - 3.4. Task Parallel Algorithms on Shared Memory Systems . . . . . 177
    - 3.4.1. Asynchronous Parallel Mining (APM) . . . . . 177
- 4. Discussion of Parallel Algorithms . . . . . 177
- 5. Summary . . . . . 179

**16 Association Rules: Advanced Techniques and Measures** **183**

- 1. Introduction . . . . . 183
- 2. Motivation . . . . . 184
- 3. Incremental Rules . . . . . 184
- 4. Generalized Association Rules . . . . . 185
- 5. Quantitative Association Rules . . . . . 187
- 6. Correlation Rules . . . . . 188
- 7. Measuring the Quality of Association Rules . . . . . 189
  - 7.1. Lift . . . . . 189
  - 7.2. Conviction . . . . . 189
  - 7.3. Chi-Squared Test . . . . . 190
- 8. Summary . . . . . 191

<b>17 Spatial Mining: Techniques and Algorithms</b>	<b>193</b>
1. Introduction and Motivation . . . . .	193
2. Concept Hierarchies and Generalization . . . . .	194
3. Spatial Rules . . . . .	196
4. STING . . . . .	197
5. Spatial Classification . . . . .	199
5.1. ID3 Extension . . . . .	200
5.2. Two-Step Method . . . . .	201
6. Spatial Clustering . . . . .	202
6.1. CLARANS . . . . .	202
6.2. GDBSCAN . . . . .	203
6.3. DBCLASD . . . . .	204
7. Summary . . . . .	204
<b>References</b>	<b>207</b>
<b>Index</b>	<b>219</b>

This page is intentionally left blank

## CHAPTER 1

### POINT ESTIMATION ALGORITHMS

Huadong Liu

*Department of Computer Science, University of Tennessee  
203 Claxton Complex, Knoxville, TN 37996-3450, USA  
hliu@cs.utk.edu*

#### Overview

Point estimation can be used in both predictive and descriptive data mining tasks. Three classical point estimation methods — the method of moments, maximum likelihood estimation, and the Expectation-Maximization algorithm — are discussed in this chapter, followed by a review of measurements of estimation performance. This chapter intends to introduce basic concepts and methods of point estimation. These concepts and methods are the basis for more advanced estimation techniques.

**Keywords:** Bias, EM algorithm, maximum likelihood estimation, mean squared error, method of moments, point estimation, standard error.

#### 1. Introduction

Statistics is the science of collecting, analyzing and presenting data. Many statistical techniques are used to perform data mining tasks. These techniques include point estimation, interval estimation, regression and many others. For a population whose distribution is known but depends on one or more unknown parameters, point estimation predicts the value of the unknown parameter and interval estimation determines the range of the unknown parameter. Point estimation techniques and algorithms will be discussed in this chapter. These classical techniques and algorithms are illustrated with examples and are not meant to reflect the state of the art

in this area. Many other useful techniques such as robust estimation methods [152] and re-sampling methods [105] have been developed and ongoing research continues to advance estimation techniques.

## 2. Motivation

Point estimation is a well-known and computationally tractable tool for learning the parameters of a data mining model. It can be used for many data mining tasks such as *summarization* and *time-series prediction*. Summarization is the process of extracting or deriving representative information about the data. Point estimation is used to estimate mean, variance, standard deviation, or any other statistical parameter for describing the data. In time-series prediction, point estimation is used to predict one or more values appearing later in a sequence by calculating parameters for a sample.

In this chapter, Sec. 3 discusses methods of point estimation, including the method of moments, maximum likelihood estimation, and the EM algorithm. Criteria to measure the performance of estimation methods, including bias, mean squared error, standard error, efficiency, and consistency are reviewed in Sec. 4. Finally, the summarization of the chapter is provided in Sec. 5.

## 3. Methods of Point Estimation

Several methods exist for obtaining point estimates, including least squares, the method of moments, maximum likelihood estimation, Bayes estimators, and robust estimation. The method of moments and maximum likelihood estimation for deriving estimates for parameters will be discussed in this section with simple examples. The EM algorithm for finding maximum-likelihood estimates will also be described.

A few formal definitions are needed before discussing methods for point estimation. Let  $X_1, X_2, \dots, X_n$  be a random sample, and let  $\Theta = \{\theta_1, \dots, \theta_k\}$  be the set of population parameters. An *estimator* is a function that maps a random sample  $X_1, \dots, X_n$  to a set of parameter values  $\hat{\Theta} = \{\hat{\theta}_1, \dots, \hat{\theta}_k\}$ , where  $\hat{\theta}_j$  is the *estimate* of parameter  $\theta_j$ .

### 3.1. The Method of Moments

The *method of moments*, introduced by Karl Pearson circa 1894, is one of the oldest methods of determining estimates [99]. In [149], the method of

moments was defined as follows: let  $X_1, X_2, \dots, X_n$  be a random sample from a population whose density function depends on a set of unknown parameters  $\Theta = \{\theta_1, \theta_2, \dots, \theta_k\}$ . Assume that the first  $k$  population moments exist as functions  $\phi_r(\Theta)$  of the unknown parameters, where  $r = 1, 2, \dots, k$ . Let

$$\hat{\phi}_r = \frac{1}{n} \sum_{i=1}^n X_i^r \quad (1)$$

be the  $r$ th sample moment. By equating  $\hat{\phi}_r$  to  $\phi_r$ , where  $r = 1, \dots, k$ ,  $k$  equations in  $k$  unknown parameters can be obtained.

Therefore, if there are  $k$  population parameters to be estimated, the method of moments consists of the following two steps:

- (i) Express the first  $k$  population moments in terms of the  $k$  population parameters  $\theta_1, \theta_2, \dots, \theta_k$ ;
- (ii) Equate the population moments obtained from step (i) to the corresponding sample moments calculated using Eq. (1) and solve  $\theta_1, \theta_2, \dots, \theta_k$  as the estimates of parameters.

**Example 1:** This is an example adapted from [105]. Suppose one wanted to find estimates for parameters of the gamma distribution using the method of moments. The gamma probability density function

$$f(x; \lambda, t) = \frac{\lambda e^{-\lambda x} (\lambda x)^{t-1}}{\int_0^\infty e^{-y} y^{t-1} dy}, \quad x \geq 0,$$

has two parameters, the shape parameter  $t$  and the scale parameter  $\lambda$ .

Since two parameters are unknown, the first step is to express  $E(X)$  and  $E(X^2)$  in terms of  $t$  and  $\lambda$ . Though the probability density function of the gamma distribution looks complicated, the mean and variance of a gamma random variable are quite simple. The mean and variance are

$$E[X] = \frac{t}{\lambda}, \quad (2)$$

and

$$V(X) = E[X^2] - (E[X])^2 = \frac{t}{\lambda^2}, \quad (3)$$

respectively.

The next step is to solve the above two equations for  $t$  and  $\lambda$  in terms of  $E(X)$  and  $E(X^2)$ . Substituting  $t$  in Eq. (3) with  $\lambda E[X]$ , which can be

derived from Eq. (2), yields

$$E[X^2] - (E[X])^2 = \frac{\lambda E[X]}{\lambda^2}. \quad (4)$$

Rearranging Eq. (4) gives the following expression for  $\lambda$

$$\lambda = \frac{E[X]}{E[X^2] - (E[X])^2}. \quad (5)$$

By substituting Eq. (5) for  $\lambda$  in Eq. (2), the parameter  $t$  is obtained in terms of  $E(X)$  and  $E(X^2)$ :

$$t = \frac{(E[X])^2}{E[X^2] - (E[X])^2}. \quad (6)$$

To get the estimates for  $\lambda$  and  $t$ , just substitute  $E[X]$  and  $E[X^2]$  with sample moments in Eqs. (5) and (6). This yields

$$\hat{t} = \frac{\bar{X}^2}{\frac{1}{n} \sum_{i=1}^n X_i^2 - \bar{X}^2},$$

and

$$\hat{\lambda} = \frac{\bar{X}}{\frac{1}{n} \sum_{i=1}^n X_i^2 - \bar{X}^2}.$$

### 3.2. Maximum Likelihood Estimation

Sir Ronald A. Fisher circa 1920 introduced the method of maximization of likelihood functions [82]. Given a random sample  $X_1, \dots, X_n$  distributed with the density (mass) function  $f(x; \Theta)$ , the *likelihood function* of the random sample is the joint probability density function, denoted by

$$L(\Theta; X_1, \dots, X_n) = f(X_1, \dots, X_n; \Theta). \quad (7)$$

In Eq. (7),  $\Theta$  is the set of unknown population parameters  $\{\theta_1, \dots, \theta_k\}$ . If the random sample consists of random variables that are independent and identically distributed with a common density function  $f(x; \Theta)$ , the likelihood function can be reduced to

$$L(\Theta; X_1, \dots, X_n) = f(X_1; \Theta) \times \dots \times f(X_n; \Theta),$$

which is the product of individual density functions evaluated at each sample point.

A *maximum likelihood estimate*, therefore, is a set of parameter values  $\hat{\Theta} = \{\hat{\theta}_1, \dots, \hat{\theta}_k\}$  that maximizes the likelihood function of the sample. A

well-known approach to find  $\hat{\Theta}$  is to take the derivative of  $L$ , set it equal to zero and solve for  $\Theta$ . Thus,  $\hat{\Theta}$  can be obtained by solving the *likelihood equation*

$$\frac{\partial}{\partial \Theta} L(\Theta) = 0.$$

It is important to note that a solution to the likelihood equation is not necessarily a maximum; it could also be a minimum or a stationary point (in the case of  $L(\Theta) = \Theta^3$ , for example). One should ensure that the solution is a maximum before using it as a maximum likelihood estimate.

It is sometimes easier, especially when working with an exponential function, to solve the logarithm of the likelihood function,  $\log L(\Theta)$ , that is,

$$\frac{\partial}{\partial \Theta} \log L(\Theta) = 0, \quad \text{where } \log L(\Theta) = \sum_{i=1}^n \log f(X_i; \Theta).$$

Since the logarithm function is monotonically increasing, which means that if  $x_1 < x_2$ ,  $\log(x_1) < \log(x_2)$ , the likelihood function  $L(\Theta)$  and its logarithm  $\log L(\Theta)$  are maximized by the same  $\Theta$ .

**Example 2:** Consider a population of balls with colors {red( $r$ ), blue( $b$ ), green( $g$ )}. Assume the color of a ball occurs with the following probabilities as a function of the parameter  $\theta$  ( $0 < \theta < 1$ ):

$$\begin{aligned} f(r; \theta) &= \theta^2, \\ f(b; \theta) &= 2\theta(1 - \theta), \\ f(g; \theta) &= (1 - \theta)^2. \end{aligned}$$

If a sample of three balls  $X_1 = r, X_2 = b, X_3 = r$  is observed, then

$$L(\theta; X_1, X_2, X_3) = f(r, b, r; \theta) = f(r, \theta)f(b, \theta)f(r, \theta) = 2\theta^5(1 - \theta).$$

Taking the derivative of the logarithm of  $L(\theta; X_1, X_2, X_3)$  and setting it to zero, the likelihood equation is obtained

$$\frac{\partial \log L(\theta)}{\partial \theta} = \frac{5}{\theta} - \frac{1}{1 - \theta} = 0,$$

which has the unique solution  $\theta = \frac{5}{6}$ . Because

$$\frac{\partial^2 \log L(\theta)}{\partial \theta^2} = -\frac{5}{\theta^2} - \frac{1}{(1 - \theta)^2} < 0$$

for all  $\theta \in (0, 1)$ ,  $\theta = \frac{5}{6}$  maximizes  $L(\theta)$ .

**Example 3:** This is an example taken from [99]. Suppose one wanted to find estimates of a normal distribution with unknown mean  $\mu$  and unknown variance  $v$ . The likelihood function for a random sample of size  $n$  is

$$L(\Theta) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi v}} e^{-\frac{(X_i - \mu)^2}{2v}} = \left( \frac{1}{2\pi v} \right)^{\frac{n}{2}} e^{-\frac{1}{2v} \sum_{i=1}^n (X_i - \mu)^2}. \quad (8)$$

Since Eq. (8) has an exponential expression, the logarithm can be used to obtain

$$\log L(\Theta) = -\frac{n}{2} \log(2\pi v) - \frac{1}{2v} \sum_{i=1}^n (X_i - \mu)^2. \quad (9)$$

By taking the partial derivative of Eq. (9) with respect to  $\mu$  and  $v$ , the following two likelihood equations can be obtained:

$$\frac{\partial \log L}{\partial \mu} = \frac{1}{v} \sum_{i=1}^n (X_i - \mu), \quad (10)$$

and

$$\frac{\partial \log L}{\partial v} = -\frac{n}{2v} + \frac{1}{2v^2} \sum_{i=1}^n (X_i - \mu)^2. \quad (11)$$

By setting Eqs. (10) and (11) to zero and solving them for  $\mu$  and  $v$  respectively, the maximum likelihood estimates are obtained with

$$\hat{\mu} = \bar{X} = \frac{1}{n} \sum_{i=1}^n X_i,$$

and

$$\hat{v} = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2.$$

### 3.3. The Expectation-Maximization Algorithm

The Expectation-Maximization (EM) algorithm is a method for finding maximum-likelihood estimates of population parameters of an underlying distribution from a given incomplete data set. It provides an iterative scheme for obtaining maximum likelihood estimates, converting a hard problem into a sequence of simpler problems. The EM algorithm obtains the initial estimates for population parameters either by random guess or previous knowledge of the data. Then it iteratively uses the estimates for

the missing data to obtain new estimates and continues until estimates converge.

The Basic EM algorithm was defined in [16] as follows: let  $\mathcal{X}$  be an incomplete data set observed or generated by some distribution with unknown parameters  $\theta_1, \theta_2, \dots, \theta_k$  and  $\mathcal{Y}$  be the unknown data set. To simplify the notation,  $\Theta$  is used to represent these unknown parameters. Assume that a complete data set  $\mathcal{Z} = (\mathcal{X}, \mathcal{Y})$  exists and assume a joint density function,

$$p(z; \Theta) = p(x, y; \Theta) = p(y; x, \Theta)p(x; \Theta),$$

where  $x \in \mathcal{X}$ ,  $y \in \mathcal{Y}$  and  $z \in \mathcal{Z}$ . With this joint density function, the complete-data likelihood function can be defined as

$$L(\Theta; \mathcal{Z}) = L(\Theta; \mathcal{X}, \mathcal{Y}) = p(\mathcal{X}, \mathcal{Y}; \Theta).$$

The EM algorithm first finds the expected value of the complete-data log-likelihood  $\log p(\mathcal{X}, \mathcal{Y}; \Theta)$  with respect to the unknown data  $\mathcal{Y}$ , given the observed data  $\mathcal{X}$  and current parameter estimates.

Define

$$Q(\Theta, \hat{\Theta}^{i-1}) = E(\log p(\mathcal{X}, \mathcal{Y}; \Theta) | \mathcal{X}, \hat{\Theta}^{i-1}), \quad (12)$$

where  $\hat{\Theta}^{i-1}$  is the current parameter estimates used to evaluate the expectation and  $\Theta$  are the new parameters optimized to increase the value of  $Q$ . On the right-hand side of Eq. (12),  $\mathcal{X}$  and  $\hat{\Theta}^{i-1}$  are constants,  $\Theta$  is a normal variable to be adjusted, and  $\mathcal{Y}$  is a random variable governed by the distribution of the data. So, the right-hand side of Eq. (12) can be re-written as:

$$E(\log p(\mathcal{X}, \mathcal{Y}; \Theta) | \mathcal{X}, \hat{\Theta}^{i-1}) = \int_{y \in \Upsilon} \log p(\mathcal{X}, y; \Theta) f(y; \mathcal{X}, \hat{\Theta}^{i-1}) dy,$$

where  $f$  is the marginal distribution of the unknown data  $\mathcal{Y}$  that depends on both the observed data  $\mathcal{X}$  and the current parameter estimates  $\hat{\Theta}^{i-1}$ , and  $\Upsilon$  is the space of values  $y$  can take on. The evaluation of  $E(\log p(\mathcal{X}, \mathcal{Y}; \Theta) | \mathcal{X}, \hat{\Theta}^{i-1})$  is called the E-step of the EM algorithm.

The second step of the EM algorithm maximizes  $E(\log p(\mathcal{X}, \mathcal{Y}; \Theta) | \mathcal{X}, \hat{\Theta}^{i-1})$  and sets

$$\hat{\Theta}^i = \underset{\Theta}{\operatorname{argmax}} Q(\Theta, \hat{\Theta}^{i-1}).$$

This is called the M-step. The E-step and M-step steps are repeated as necessary. Each iteration is guaranteed to increase the log-likelihood of the

complete-data and the algorithm is guaranteed to converge to a local maximum of the likelihood function [16]. The EM algorithm can be summarized as follows:

---

**Algorithm 1** EM Algorithm

---

**Input:**

Data set distribution with unknown parameters  $\Theta = \{\theta_1, \dots, \theta_k\}$ ;  
 Incomplete data set  $\mathcal{X}$ ;  
 Convergence threshold  $\epsilon$ ;

**Output:**

Parameter estimates  $\hat{\Theta} = \{\hat{\theta}_1, \dots, \hat{\theta}_k\}$

Initialize  $\hat{\Theta}^0$

$n \leftarrow 0$ ;

**repeat**

$n \leftarrow n + 1$ ;

$Q(\Theta, \hat{\Theta}^{n-1}) \leftarrow E(\log p(\mathcal{X}, \mathcal{Y}; \Theta) | \mathcal{X}, \hat{\Theta}^{n-1})$ ;

Compute the maximum likelihood estimates of  $\Theta$  to maximize  $Q(\Theta, \hat{\Theta}^{n-1})$ ;

$\hat{\Theta}^n \leftarrow \operatorname{argmax}_{\Theta} Q(\Theta, \hat{\Theta}^{n-1})$ ;

**until**  $|\hat{\Theta}^n - \hat{\Theta}^{n-1}| < \epsilon$

$\hat{\Theta} \leftarrow \hat{\Theta}^n$ ;

---

The EM algorithm is useful in computational pattern recognition [16], image retrieval [159], computer vision [101], and many other fields. In data mining, the EM algorithm can be used when the data set has missing values due to limitations of the observation process. It is especially useful when maximizing the likelihood function directly is analytically intractable. In that case, the likelihood function can be simplified by assuming that the hidden parameters are known.

#### 4. Measures of Performance

As discussed in Sec. 3, there are several different ways (estimators) to estimate unknown parameters. In order to assess the usefulness of estimators, some criteria are necessary to measure the performance of estimators. In this section, five criteria used to assess estimators — bias, mean squared error, standard error, efficiency, and consistency will be discussed. At the end of this section, the Jackknife method will be introduced to estimate

bias and standard error of an estimator. As discussed before, an estimator is a function that maps a random sample to a set of parameter estimates. Furthermore, if the sample obtained is a random sample, an estimator is also a random variable since the estimates are calculated using the sample. In the following discussion,  $\hat{\theta}$  is denoted as the estimator (random variable) of an unknown parameter  $\theta$ .

#### 4.1. Bias

The bias of an estimator provides a measure of the average error in the estimator  $\hat{\theta}$  of a parameter  $\theta$ . The *bias* of an estimator is defined as the difference between the expected value of the estimator and the actual value

$$\text{bias}(\hat{\theta}) = E[\hat{\theta}] - \theta. \quad (13)$$

An estimator is *unbiased* if the expected value of the estimator equals the true parameter value, i.e.,  $E[\hat{\theta}] = \theta$ . Otherwise, the estimator is *biased*. For example, the maximum likelihood estimate of the mean for a normal distribution is unbiased, since  $E[\hat{\mu}] = \mu$  [109]. However, this is not the case for the maximum likelihood estimate of the variance  $\hat{v}$  [68]. It can be shown that  $E[\hat{v}] = \frac{(n-1)}{n}v$ , where  $n$  is the sample size.

To determine the expected value in Eq. (13), the distribution of the statistic  $\hat{\theta}$  must be known to analytically calculate the bias. If the distribution of the statistic is not known, then some methods such as the Jackknife (see Sec. 4.6) can be used to estimate the bias of  $\hat{\theta}$ .

#### 4.2. Mean Squared Error

The *mean squared error* (MSE) is the expected value of the squared error. Let  $\theta$  be a parameter and  $\hat{\theta}$  be an estimator of the parameter, the mean squared error of the estimator is defined as

$$MSE(\hat{\theta}) = E[(\hat{\theta} - \theta)^2]. \quad (14)$$

It is sometimes more useful to rewrite the MSE equation in terms of the bias and the variance [109]. The first step of the rewriting is to expand the expected value on the right-hand side of Eq. (14) to get

$$MSE(\hat{\theta}) = E[(\hat{\theta}^2 - 2\hat{\theta}\theta + \theta^2)] = E[\hat{\theta}^2] - 2\theta E[\hat{\theta}] + \theta^2. \quad (15)$$

The next step of the rewriting is to add to and subtract  $(E[\hat{\theta}])^2$  from the right-hand side of Eq. (15) so that

$$MSE(\hat{\theta}) = E[\hat{\theta}^2] - (E[\hat{\theta}])^2 + (E[\hat{\theta}])^2 - 2\theta E[\hat{\theta}] + \theta^2. \quad (16)$$

By simplifying Eq. (16), the mean squared error can be written as

$$MSE(\hat{\theta}) = E[\hat{\theta}^2] - (E[\hat{\theta}])^2 + (E[\hat{\theta}] - \theta)^2 = V(\hat{\theta}) + [bias(\hat{\theta})]^2. \quad (17)$$

Equation (17) shows how the mean squared error, variance and bias of an estimator are related. Since the mean squared error is the sum of the variance and the squared bias, two non-negative quantities, the error will be small when the variance and the absolute value of the bias are both small. When  $\hat{\theta}$  is unbiased, the mean squared error is equal to the variance.

### 4.3. Standard Error

The standard error gives a measure of the precision of the estimators. The *standard error* of an estimator  $\hat{\theta}$  is defined as the standard deviation of its sampling distribution

$$SE(\hat{\theta}) = \sqrt{V(\hat{\theta})} = \sigma_{\hat{\theta}}.$$

The sample mean can be used as an example to illustrate the concept of standard error. Let  $f(x)$  represent a probability density function with finite variance  $\sigma^2$  and mean  $\mu$ . Let  $\bar{X}$  be the sample mean for a random sample of size  $n$  drawn from this distribution. By the Central Limit Theorem [105], the distribution of  $\bar{X}$  is approximately normally distributed with mean  $\mu$  and variance  $\frac{\sigma^2}{n}$ . So the standard error is given by

$$SE(\bar{X}) = \sigma_{\bar{X}} = \frac{\sigma}{\sqrt{n}}.$$

When the standard deviation  $\sigma$  for the underlying population is unknown, then an estimate  $S$  for the parameter can be used as a substitute for it and leads to the estimated standard error

$$\widehat{SE}(\bar{X}) = \hat{\sigma}_{\bar{X}} = \frac{S}{\sqrt{n}}.$$

### 4.4. Efficiency

Another measure used to compare estimators is called efficiency. Suppose there are two estimators  $\hat{\theta}$  and  $\hat{\theta}'$  for a parameter  $\theta$  based on the sample  $X_1, \dots, X_n$ . If the MSE of one estimator is less than the MSE of the other,

i.e.,  $MSE(\hat{\theta}) < MSE(\hat{\theta}')$ , then the estimator  $\hat{\theta}$  is said to be more *efficient* than  $\hat{\theta}'$ . The *relative efficiency* of  $\hat{\theta}$  with respect to  $\hat{\theta}'$  is defined as the ratio

$$eff(\hat{\theta}, \hat{\theta}') = \frac{MSE(\hat{\theta}')}{MSE(\hat{\theta})}.$$

If this ratio is greater than one, then  $\hat{\theta}$  is a more efficient estimator of the parameter  $\theta$ . When the estimator is unbiased, the ratio is just the ratio of their variance, and the most efficient estimator would be the one with minimum variance.

#### 4.5. Consistency

Unlike the four measures defined in previous subsections, consistency is defined for increasing sample sizes, not a fixed sample sizes. Like the efficiency, consistency is also defined using the MSE. Let  $\hat{\theta}_n$  be the estimator of a parameter based on a sample of size  $n$ , then an estimator is said to be *consistent* if

$$\lim_{n \rightarrow \infty} MSE(\hat{\theta}_n) = 0, \quad (18)$$

or

$$\lim_{n \rightarrow \infty} [V(\hat{\theta}_n) + (bias(\hat{\theta}_n))^2] = 0 \quad (19)$$

when  $MSE$  is written in terms of bias and variance. Thus, Eq. (18) or Eq. (19) holds if and only if both variance and bias of  $\hat{\theta}_n$  tend to zero as  $n$  approaches infinite.

#### 4.6. The Jackknife Method

Given a random sample and a parameter  $\theta$ , its estimate is also a random variable and has some error associated with it. Estimates of bias and standard error of the estimator  $\hat{\theta}$  can assess the accuracy of the results. The Jackknife method is a technique for estimating bias and standard error of statistics [44].

The Jackknife obtains the estimate of a parameter from a set of observed data by generating that statistic repeatedly on the data set excluding a single data value during each iteration. The Jackknife method consists of taking repeated sub-samples of the original sample of  $n$  independent observations by omitting a single observation at a time. Thus, each sub-sample

---

**Algorithm 2** Jackknife
 

---

**Input:**

An estimator  $\hat{\theta} = \tau(X_1, \dots, X_n)$ ; a random sample  $X_1, \dots, X_n$ ;

**Output:**

Jackknife estimate of bias of  $\hat{\theta}$ ; Jackknife estimate of standard error of  $\hat{\theta}$ ;

**for**  $i = 1$  to  $n$

Leave out the sample point  $X_i$ ;

Calculate the value of the statistic using remaining sample points to obtain  $\hat{\theta}_{(i)}$ ;

**end for**

Calculate the overall Jackknife estimate using Eq. (20), the Jackknife estimate of bias of  $\hat{\theta}$  using Eq. (21), and the Jackknife estimate of standard error of  $\hat{\theta}$  using Eq. (22).

---

consists of  $n - 1$  observations formed by deleting a different observation from the sample. The Jackknife estimate and its standard error are then calculated from these truncated sub-samples.

Suppose that there is a set of  $n$  values  $X_1, \dots, X_n$ , the  $i$ th Jackknife estimate is calculated by omitting the  $i$ th value

$$\hat{\theta}_{(i)} = \tau(X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n).$$

For example, the  $i$ th Jackknife estimate for the mean  $\mu$  would be

$$\hat{\mu}_{(i)} = (1/n - 1) \sum_{j=1}^{i-1} X_j + (1/n - 1) \sum_{j=i+1}^n X_j.$$

Given a set of Jackknife estimates,  $\hat{\theta}_{(i)}$ ,  $i = 1, 2, \dots, n$ , an overall estimate,  $\hat{\theta}_{(\cdot)}$  can be obtained by

$$\hat{\theta}_{(\cdot)} = (1/n) \sum_{i=1}^n \hat{\theta}_{(i)}. \quad (20)$$

The estimate of the bias of  $\hat{\theta}$  obtained by the Jackknife method is given by [44]

$$\widehat{Bias}_{jack}(\hat{\theta}) = (n - 1)(\hat{\theta}_{(\cdot)} - \hat{\theta}). \quad (21)$$

The estimated standard error of  $\hat{\theta}$  using the Jackknife method is defined as

$$\widehat{SE}_{jack}(\hat{\theta}) = \left[ \frac{n-1}{n} \sum_{i=1}^n (\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)})^2 \right]^{\frac{1}{2}}. \quad (22)$$

## 5. Summary

In this chapter, the method of moments, maximum likelihood estimation, and the EM algorithm have been discussed with simple examples. Even though classical point estimation is a useful theoretical topic, it requires some knowledge about the data involved and violates an important principle of data mining — avoid making any assumptions about the data. Also, point estimation is too simple for data mining applications that have huge data sets and complex processing models. The need to solve real problems has driven the evolution of estimation techniques and algorithms. It has progressed from least squares to the method of moments, to maximum likelihood, to Bayes and empirical Bayes procedures, to risk-reduction approaches, to robustness, and to re-sampling techniques [82]. Readers interested in further details of these advanced topics will benefit from reading [95].

This page is intentionally left blank

## CHAPTER 2

### APPLICATIONS OF BAYES THEOREM

Erika Fuentes

*Department of Computer Science, University of Tennessee  
203 Claxton Complex, Knoxville, TN 37996-3450, USA  
efuentes@cs.utk.edu*

#### Overview

Classification is an important task in data mining because it helps to address a variety of problems. Statistical techniques can provide the means to solve these problems in a simple way. Specifically, the Bayesian approach provides a natural and flexible way to approach classification problems and other probability-related questions. The Bayes Theorem is the basis of this methodology, and it can also be used as a building block and starting point for more complex methodologies such as the popular Bayesian networks.

**Keywords:** Bayes decision rule, Bayesian networks, Bayes Theorem, classification.

#### 1. Introduction

Although classification or discrimination can be seen as a problem of prediction and class assignment, it is also a learning process and is often used in data mining applications and in pattern recognition. Statistical methods for classification are widely used and they provide a simple starting point for addressing many problems. They are relatively easy to implement, but should be used judiciously since they typically require many assumptions. Success using these techniques depends both on the size of the data set and previous knowledge regarding the set.

The Bayes Theorem is a statistical concept that can be used as a basis for data mining techniques, such as pattern classification and discrimination. It can also be used as a starting point for implementing more complex data mining and Knowledge Discovery Database (KDD) techniques such as Bayesian networks [122]. The Bayes approach can also be used in combination with other methodologies such as Markov Chains and other probabilistic techniques, many of which are used to build innovative models such as the Relational Bayesian Classifiers for relational data sets [112].

The Bayes methodology has a wide range of applications in many different fields such as medicine, genetics and business. The simplicity and natural interpretation of its output makes it suitable for many applications, yet it has some disadvantages since in many cases Bayesian models tend to be naive or too simple for certain types of data.

## **2. Motivation**

In general, data mining is the process of automatically extracting useful knowledge from large data sets. It has become increasingly important as the amount of data being generated continues to grow. By using data mining, the structure in data can be identified, resulting in recognized patterns, statistical or predictive models of the data, and relationships among parts of it. Furthermore, this information can be used for future predictions [85].

The basic problems of data mining involve the classification, discrimination and clustering of data. Classification involves the ordering of a set of objects described by high-dimensional data into small units, or classes that give a better understanding, control, and interpretation and retrieval of the data. The main goal of classification is to assign an instance to a class depending on the values of descriptive features. One of the most popular and successful methods in classification is statistical pattern classification [85].

Statistics provides methods for data analysis, such as sampling, stochastic modeling, predictions, experimental design, and exploratory data analysis. With the increasing size and complexity of data, new types of statistical models and applications are becoming more closely related to computer science [85]. The Bayesian approach in statistical data analysis is based on a probability model, which is based on observed or given information. This approach determines the conditional probability distributions for the possible classifications of the data of being produced. In other words, it estimates the likelihood of some property given some input data as evidence. The Bayes Theorem is used in particular to compute these conditional posterior

probability distributions (with the aid of probability density estimation methods) [163]. It expresses the problem in terms of probabilities that can be used to make classification decisions.

In Bayes classification, the learning process estimates probabilities instead of finding an explicit rule. The advantage of this approach is that the classifier will reach the minimum error when the data set is large and the methods for estimating particular probabilities are consistent. With large sample sets, these probabilities will converge to the underlying probability distribution of the classes. If the designer has some prior knowledge about the data that would lead to the belief that some model is a particularly good approximation to the probability distribution, the classifier will have a very good performance. The problem is that the model used in the classification might not be the best estimator of the probability distribution, but unrealistic models that make naive assumptions are not necessarily bad and often will lead to relatively good performance. One should note that Bayesian classification could give you the “correct” classification, while not providing the acceptable solution to the real problem since the cost of the errors could be unacceptable (for example, classifications in cancer diagnosis or tornado warnings) [85, 144].

This chapter gives a brief introduction to the basic methodology of the Bayesian approach, and some examples of its application. The next section covers the general framework for statistical classification techniques and in more detail the Bayesian methodology. Section 4 contains two examples of the Bayesian approach. Section 5 is a brief summary addressing some advantages and disadvantages of this methodology.

### **3. The Bayes Approach for Classification**

#### **3.1. *Statistical Framework for Classification***

Statistical methods are widely used in discrimination and pattern recognition problems, because they provide a simple framework for classification. Given a data set of interest as input, the main steps (see also Fig. 1) of these methods are [40]:

- (i) Feature extraction. The features (or characteristics) should be representative of the classes and adequate for the method.
- (ii) Information gathering. Obtain a sample set of these features for the different classes of interest. Select variables and measurements that can best discriminate between the classes; this includes, in many

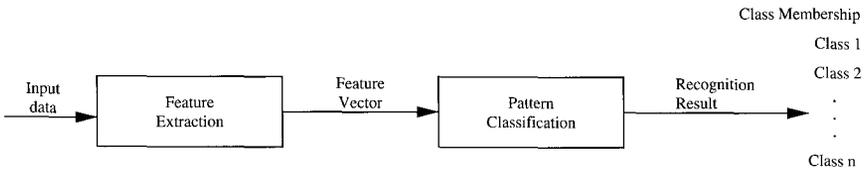


Fig. 1. General flow diagram for the statistical approach to data classification.

cases, preprocessing of the data, compression and sampling. Also collect information about the statistical properties of the classes. This step usually involves the creation of training and validation data sets.

- (iii) Creation of a methodology for classification. Construct a “good” classifier using the information available from the existing classes. Based on the information collected, choose or design classifiers that take as input a set of feature measurements (feature vectors) from a training set. For each element of a validation set indicate its class.
- (iv) Evaluation of the classification results. Measure the classifier’s performance, determine class membership for each of the samples, and verify if the selected class is indeed the correct one.
- (v) Apply the classifier to new objects.

Each of the objects from a data set, described by a vector  $x$  of  $k$  components or features, are originally known to belong to one of the target classes. The goal of classification is to find this unknown class. Hence, it must be “reconstructed” or guessed from the values of  $x$  by means of a function, which can be used as a regression function for the data distribution (of each class). This function is a prediction or estimate of the true but unknown class of the object [85]. In theory, this might be possible if each class is described by a particular data distribution, and the objects in a class have a specific behavior and have feature values that are distinctive of the class. Unfortunately (and partly due to the randomness of the input data), the target classes are not always independent and separated from each other. For this reason, their data distributions will very likely overlap, leading to misclassifications [163]. One of the objectives when building classifiers is to minimize the number of misclassifications. This problem will be addressed in more detail with Bayesian classifiers in the next subsection.

There are two main approaches for building classifiers used in statistical pattern recognition:

- **Parametric Classifiers:** also called *supervised* methods estimate the parameters for a presumed probability distribution function such as the Normal or Gaussian distribution. This function is used as a basis for making decisions [42]. The problem with this approach is that it sometimes requires a great deal of knowledge about the data involved [105]. A typical example of this approach is the Bayes method, where a class-specific probability model is designed for the random observations, given that a learning sample of representative data vectors is provided for each class.
- **Non-parametric Classifiers:** also called *unsupervised* methods do not assume the existence of any specific probability distribution. Instead, the parameters and the decision function are built specifically from the collected data. They can be better for data mining applications since they are data-driven and there are no explicit equations to determine the models. However, they require large amounts of input data to perform better. Examples of these are clustering techniques and decision trees [40, 42].

Figure 2 shows an overview of the parametric approach used by the Bayes methodology. It is based on the idea of using part of the input data to “teach” the classifier to recognize the data [38]. The important steps of this type of classification are:

- (i) Using a set of training (learning) data for which we know the class membership of each sample, derive a decision rule.
- (ii) Apply this decision rule to a new set of validating (testing) data samples, whose classification is not known, and assign each sample to a class.

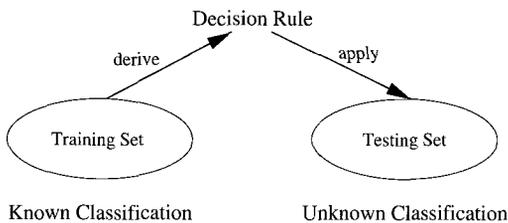


Fig. 2. Overview of the parametric statistical methods.

- (iii) Evaluate the classification: verify, if possible, that each sample was assigned to the correct class. One measure of accuracy is the proportion of samples that were classified correctly.

The next section contains a more detailed description of the Bayesian methodology for classification.

### 3.2. Bayesian Methodology

This Bayes approach defines the classification problem in terms of probabilities. More specifically, the three main concepts required are conditional probability, Bayes Theorem, and the Bayes decision rule.

The conditional probability  $P(A|B)$ , which is used to define independent events [105], is defined by

$$P(A|B) = \frac{P(A \cup B)}{P(B)},$$

where  $P(A|B)$  is the probability that event  $A$  happens, given that  $B$  is observed. Similarly,

$$P(B|A) = \frac{P(A \cup B)}{P(A)},$$

where  $P(B|A)$  is the probability that event  $B$  happens, given that  $A$  is observed. It then follows (by substitution) that

$$P(A \cap B) = P(A)P(B|A).$$

The premise of Bayes Theorem starts with an initial degree of belief that an event will occur, and then with new information this degree of belief can be “updated” [105]. These two degrees are represented, respectively, by the *prior* probability  $P(A|B)$  and the *posterior* probability  $P(B|A)$ , which are related by

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}.$$

The Bayes decision rule states that based on the posterior probabilities, it is possible to assign an element  $x$  to a class with the largest probability. In particular, for the classifying problem, the conditional probabilities described above can be defined as follows: let  $x$  be a data sample (vector of features) and  $\omega_i$  one of the possible classes [40]. Then,  $P(x|\omega_i)$  is the *prior* probability, because it can be obtained based on prior knowledge

(i.e., the distribution constructed from training data). Given class  $i$ , it specifies the probability of finding  $x$  within this class. Similarly,  $P(\omega_j|x)$  is the *posterior* probability, because it is computed based on posterior knowledge. Given sample  $x$ , it specifies the probability that  $x$  belongs to class  $j$ . For a given  $x$ , if

$$P(\omega_1|x) > P(\omega_2|x),$$

where

$$P(\omega_i|x) = \frac{P(x|\omega_i)P(\omega_i)}{P(x)}, \quad (23)$$

then  $x$  belongs to class 1, otherwise to class 2.

The denominator term of Eq. (23) is the overall probability of  $x$  in all the classes. For a given  $x$ , one must compute the posterior probabilities for all  $\omega_i$  classes, then assign  $x$  to the class that yields the maximum posterior probability [38]. Figure 3 illustrates how the Bayes decision rule can be used for single-feature (one-dimensional) classification. The two probability distributions shown demonstrate the values obtained for the posterior probabilities for each class. The shaded regions indicate the areas of possible misclassification, i.e., those areas where the probability distribution of the other class is greater.

Generally speaking, the Bayesian methodology for classification follows these five steps [40]:

- (i) Collect data, and estimate parameters such as mean and covariance for each class (for the parametric approach we assume that all the probability density functions have a Gaussian behavior).
- (ii) Choose a set of features.
- (iii) Choose a model and derive a decision rule with these parameters.

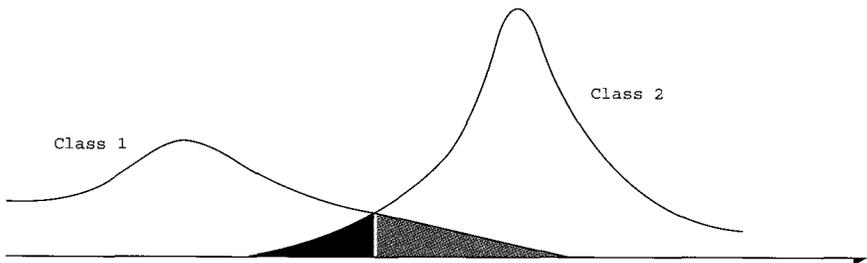


Fig. 3. Probability distributions for two classes using a single-feature.

- (iv) Train the classifier and apply the decision rule by using a *discriminant function* (a way to represent a pattern classifier), and apply it to a test data set to classify each sample.
- (v) Evaluate the decision rule. Measure the accuracy/error rate in order to improve the choice of features and the overall design of the classifier.

The next section provides specific examples of how this methodology can be used for predicting or classifying different types of objects.

## 4. Examples

In the first example, the Bayes decision rule is used in numerical method selection, where there are many alternative algorithms that can be used to solve the same numerical problem (e.g., solving the linear system of equations  $Ax = b$  for the vector  $x$ ). Statistical data mining can play an important role in matching the most appropriate algorithm for a particular numerical problem. In the second example of this section, Bayesian networks are briefly introduced for modeling gene-to-gene functional relationships.

### 4.1. Example 1: Numerical Methods

The Bayes decision rule is one of the applications of Bayes Theorem. This example shows how it could be used in numerical method decision-making. Solving problems using algorithms from numerical linear algebra can consume considerable computation time. By using a statistical approach, it might be possible to find the method(s) that more efficiently solve a problem by extracting and characterizing features from the underlying matrices, and assigning them to classes corresponding to methods that will work well on similar problems with similar features. While this idea is still in an experimental phase [38], it is a good example that shows how a statistical approach can be used to identify important features of data sets, and more specifically, how the Bayes Theorem can be used for classification tasks.

The strategy for determining appropriate numerical algorithms by the Bayesian approach is summarized as follows:

- (i) Solve a large collection of test problems by every available method, that is, every choice of algorithm, and suitable setting of parameters.
- (ii) Assign each problem to the algorithm class corresponding to the method that gave the fastest solution.
- (iii) Compute a probability density function for each class.
- (iv) Using the Bayesian classification, assign new problems to the classes and test the efficiency of the classifier.

The output of this process is a function  $P_i(x) = P(w_i|x)$ , where  $i$  ranges over all classes (methods), and  $x$  lies in the space of the feature vectors for the input problems. Given a new problem and its feature vector  $x$ , the goal is to select the method  $i$  for which  $P_i(x)$  is maximized.

One of the crucial steps in the classification process is the extraction of features from the application. In the context of numerical problems there are many features, but not all of them are relevant for the decision-making process, so several combinations are tested. Some features (such as *structural* characteristics of a matrix), have to be pre-processed before being fed into a classifier. Only after a testing phase is it possible to determine which features are actually significant for the decision process. The set of features chosen is known as the *feature vector*.

The most expensive step in this example is the training stage. This essential part of the training process assigns each numerical problem to a class, where the classes correspond to the solution methods. All the sample problems are solved by each available method, and then a data set is built by finding for each method (class) a number of problems that are best solved by it. All the results from this stage are divided into two sets — a training set and a validation set. With the training data available, the Bayes methodology is used to train a classifier. Then, using the feature vectors from this data, a vector of mean values  $\bar{\mu}$  and a correlation matrix  $\Sigma$  can be computed for each class [42], i.e.,

$$\Sigma = \frac{1}{n} \sum_{i=1}^n (\bar{x}_i - \mu)(\bar{x}_i - \mu)^T,$$

where  $\{\bar{x}_i\}_{i=1}^n$  are each of the samples in the appropriate class.

Using the mean value vector and the correlation matrix, a multivariate density function for each method (class)  $j$  can be given by

$$P_j(\bar{x}) = \frac{1}{2\pi^{|\Sigma|^{1/2}}} e^{-(1/2)(\bar{x}-\bar{\mu})\Sigma^{-1}(\bar{x}-\bar{\mu})}.$$

With a probability density function for each class, it is possible to compute the posterior probabilities (see Sec. 3.2) using the Bayes Theorem. For example, the probability that  $\bar{x}$  belongs in class  $j$  can be determined [40] by

$$P(w_i|\bar{x}) = \frac{P(\bar{x}|w_j)P(w_j)}{P_j(\bar{x})}.$$

The classifier is then built using these posterior probabilities and once it has been established, it can be used to classify and validate the data set. Given a new problem and its associated feature vector, the classifier is then

used to select the numerical method (class) which yields the maximum posterior probability. To evaluate the accuracy of the model one can compare the class assigned by the classifier to the one that corresponds to the class of the numerical method that was experimentally judged to be the most efficient.

Following this approach, a series of experiments have been run on a number of matrices using two different sets of features. Classifiers were then evaluated using the validation set. Each entry tabulated below represents the rate of accuracy with which new test problems were correctly classified [38]:

	Feature Set 1	Feature Set 2
Method 1	70%	40%
Method 2	98%	86%
Method 3	93%	45%

From these results a couple of observations can be made about the Bayes approach for classification:

- The performance is good for one set of features, but not for the second set. This is why it is critical to have as much knowledge regarding the data sets and the relevant features that best describe or represent this data and the application that is being modeled. This can be either an advantage or a disadvantage of the Bayesian methodology depending on expertise in the field and prior knowledge of the source data.
- When a good set of relevant features is chosen, the Bayes approach can produce good results, even though many naive assumptions have to be made.

#### **4.2. Example 2: Bayesian Networks**

Bayesian networks are an example of the use of the Bayes Theorem as a part of more complex applications. Bayesian networks provide a blueprint to help understand relations among a large number of variables. They are useful in modeling problems in which there is a large amount of data involved, and in recent years they have been especially used in biomedical areas for feature extraction and classification [46].

A Bayesian network is basically a directed acyclic graph that models the probabilistic dependencies of a group of nodes [122]. This example concerns the use of Bayesian networks in genetics, for combining data from various heterogeneous sources for gene function prediction. The Bayesian network

can combine data from diverse sources and try to predict if two proteins are functionally related. For each pair of genes one may ask: "What is the probability, based on the evidence presented, that products of genes  $i$  and  $j$  have a functional relationship?" The network is then built based on the data from gene-gene relationship (prior probability) matrices  $S = [s_{ij}]$  where element  $s_{ij} \neq 0$  if genes  $i$  and  $j$  are believed to have a functional relationship and  $s_{ij} = 0$ , if they do not. The output of the matrix is a posterior probability matrix whose scores  $S_{i,j}$  correspond to the strength of each method's posterior belief in the existence of relationship between genes  $i$  and  $j$ , and the final goal is to create gene groupings (classification) based on this data [144].

Expert knowledge is generally needed to assess the prior probabilities (e.g., from microarray analysis and molecular biology), and Bayesian networks constructed from probabilities provided by experts in the field have been successfully used [144]. Such results can be compared and enriched with results from other methodologies such as microarray analysis, so this approach can also be used as a starting point for more complex techniques.

## 5. Summary

Statistical techniques are very useful in data mining for classification and prediction tasks. These techniques are not always very efficient or accurate, but they are simple to implement and to understand. Their performance is strongly related to the amount of available knowledge of the data and the understanding of the problem being modeled. More efficient models are implemented when it is possible to extract a relevant set of features, and obtain good data distributions that can more accurately describe the behavior of the data. The Bayesian approach is one of these techniques, and is based on conditional probabilities that represent prior and posterior knowledge of the data. This method is very versatile but can be naive since certain assumptions need to be made. However, there are several modifications that can improve its performance [85, 112, 122].

This page is intentionally left blank

## CHAPTER 3

### SIMILARITY MEASURES

Kevin E. Heinrich

*Department of Computer Science, University of Tennessee  
203 Claxton Complex, Knoxville, TN 37996-3450  
heinrich@cs.utk.edu*

#### Overview

Similarity measures provide the framework on which many data mining decisions are based. Tasks such as classification and clustering usually assume the existence of some similarity measure, while fields with poor methods to compute similarity often find that searching data is a cumbersome task. Several classic similarity measures are discussed, and the application of similarity measures to other fields are addressed.

**Keywords:** Bioinformatics, distance metrics, information retrieval, multi-dimensional modeling, ontologies, similarity measures.

#### 1. Introduction

The goal of information retrieval (IR) systems is to meet users' needs. In practical terms, a need is usually manifested in the form of a short textual query entered in the text box of some search engine online. IR systems typically do not directly answer a query; instead, they present a ranked list of documents that are judged relevant to that query by some similarity measure [115]. Since similarity measures have the effect of clustering and classifying information with respect to a query, users will commonly find new interpretations of their information need that may or may not be useful to them when reformulating their query. In the case when the query is a document from the initial collection, similarity measures can be used to cluster and classify documents within a collection. In short, similarity

measures can add a rudimentary structure to a previously unstructured collection.

## 2. Motivation

Similarity measures used in IR systems can distort one's perception of the entire data set. For example, if a user types a query into a search engine and does not find a satisfactory answer in the top ten returned web pages, then he/she will usually try to reformulate his/her query once or twice. If a satisfactory answer is still not returned, then the user will often assume that one does not exist. Rarely does a user understand or care what ranking scheme a particular search engine employs.

An understanding of the basic similarity measures, however, is crucial in today's business world. Many business decisions are often based on answers to questions that are posed in a way similar to how queries are given to search engines. Data miners do not have the luxury of assuming that the answers given to them from a database or IR system are correct or all-inclusive — they must know the drawbacks of any similarity measure used and adjust their business decisions accordingly.

This chapter will discuss classic similarity measures from [132, 133] such as Dice, Overlap, Jaccard, and Cosine. Asymmetric and distance measures will also be discussed.<sup>a</sup> A simple example will be presented demonstrating the effectiveness of the various measures. Finally, uses of similarity measures in applications such as multi-dimensional modeling, hierarchical clustering, and bioinformatics will be discussed.

## 3. Classic Similarity Measures

A similarity measure is defined as a mapping from a pair of tuples of size  $k$  to a scalar number. By convention, all similarity measures should map to the range  $[-1, 1]$  or  $[0, 1]$ , where a similarity score of 1 indicates maximum similarity. Similarity measures should exhibit the property that their value will increase as the number of common properties in the two items being compared increases [42].

A popular model in many IR applications is the vector-space model, where documents are represented by a vector of size  $n$ , where  $n$  is the size of the dictionary. Thus, document  $i$  is represented by a vector

---

<sup>a</sup>For a more extensive listing of similarity and related measures, the reader is directed to [29, 133].

$d_i = (w_{1i}, \dots, w_{ni})$ , where  $w_{ki}$  denotes the weight associated with term  $k$  in document  $i$ . In the simplest case,  $w_{ki}$  is the frequency of occurrence of term  $k$  in document  $i$ .<sup>b</sup> Queries are formed by creating a *pseudo*-document vector  $q$  of size  $n$ , where  $w_{kq}$  is assumed to be non-zero if and only if term  $k$  occurs in the query.

Given two similarity scores  $\text{sim}(q, d_i) = s_1$  and  $\text{sim}(q, d_j) = s_2$ ,  $s_1 > s_2$  means that document  $i$  is judged more relevant than document  $j$  to query  $q$ . Since similarity measures are a pairwise measure, the values of  $s_1$  and  $s_2$  do not imply a relationship between documents  $i$  and  $j$  themselves.<sup>c</sup>

From a set theoretic standpoint, assume that a universe  $\Omega$  exists from which subsets  $A, B$  are generated. From the IR perspective,  $\Omega$  is the dictionary, while  $A$  and  $B$  are documents with  $A$  usually representing the query. Some similarity measures are more easily visualized via set theoretic notation.

As a simple measure,  $A \cap B$  denotes the number of shared index terms. However, this *Simple coefficient* takes no information about the sizes of  $A$  and  $B$  into account. The Simple coefficient is analogous to the binary weighting scheme in IR that can be thought of as the frequency of term co-occurrence with respect to two documents. Although the Simple coefficient is technically a similarity measure, it will not be further discussed in this chapter.

Most similarity measures are themselves evaluated by precision and recall. Let  $A$  denote the set of retrieved documents and  $B$  denote the set of relevant documents. [45] defines *precision* and *recall* as

$$P(A, B) = \frac{|A \cap B|}{|A|}$$

and

$$R(A, B) = \frac{|A \cap B|}{|B|},$$

respectively. Informally, precision is the ratio of returned relevant documents to the total number of documents returned, while recall is the ratio of returned relevant documents to the total number of relevant documents. Precision is often evaluated at varying levels of recall (namely,  $i = 1, \dots, |B|$ ) to produce a precision-recall graph. Ideally, IR

---

<sup>b</sup>Weighting schemes are beyond the scope of this chapter. For a more in-depth discussion of weighting schemes, the reader is directed to [12, 15].

<sup>c</sup>From the author's experience, this observation is often misunderstood in multidisciplinary fields such as bioinformatics.

systems generate high precision at all levels of recall. In practice, however, most systems exhibit lower precision values at higher levels of recall.

When applicable, the similarity measures presented in the next subsections will be defined in set theoretic notation followed by the equivalent matrix notation. While the different notation styles may not yield exactly the same numeric values for each pair of items, the ordering of the items within a set is preserved.

### 3.1. *Dice*

The *Dice coefficient* is a generalization of the harmonic mean of the precision and recall measures. A system with a high harmonic mean should theoretically be closer to an ideal retrieval system in that it can achieve high precision values at high levels of recall. The harmonic mean for precision and recall is given by

$$E = \frac{2}{\frac{1}{P} + \frac{1}{R}}, \quad (24)$$

while the Dice coefficient is denoted by

$$\text{sim}(q, d_j) = D(A, B) = \frac{|A \cap B|}{\alpha|A| + (1 - \alpha)|B|} \quad (25)$$

$$\cong \frac{\alpha \sum_{k=1}^n w_{kq} w_{kj}}{\alpha \sum_{k=1}^n w_{kq}^2 + (1 - \alpha) \sum_{k=1}^n w_{kj}^2}, \quad (26)$$

with  $\alpha \in [0, 1]$ . To show that the Dice coefficient is a weighted harmonic mean, let  $\alpha = \frac{1}{2}$  and note that Eq. (25) is equivalent to Eq. (24).

### 3.2. *Overlap*

As its name implies, the *Overlap coefficient* attempts to determine the degree to which two sets overlap. The Overlap coefficient is computed as

$$\begin{aligned} \text{sim}(q, d_j) = O(A, B) &= \frac{|A \cap B|}{\min(|A|, |B|)} \\ &\cong \frac{\sum_{k=1}^n w_{kq} w_{kj}}{\min\left(\sum_{k=1}^n w_{kq}^2, \sum_{k=1}^n w_{kj}^2\right)}. \end{aligned}$$

The Overlap coefficient is sometimes calculated using the `max` operator in place of the `min`. Note that the denominator does not necessarily normalize the similarity values produced by this measure. As a result, the Overlap values are typically higher in magnitude than most other similarity measures.

### 3.3. Jaccard

The percentage of relevance covered by two sets is known as the *Jaccard coefficient* and is given by

$$\begin{aligned} \text{sim}(q, d_j) = J(A, B) &= \frac{|A \cap B|}{|A \cup B|} \\ &\cong \frac{\sum_{k=1}^n w_{kq} w_{kj}}{\sum_{k=1}^n w_{kq}^2 + \sum_{k=1}^n w_{kj}^2 - \sum_{k=1}^n w_{kq} w_{kj}}. \end{aligned}$$

This measure is fairly intuitive and often one of the more widely used measures when comparing IR systems. In a set theoretic sense, the Jaccard measure signifies the degree of relevance covered by the union of two sets.

### 3.4. Asymmetric

Similarity measures do not necessarily have to demonstrate symmetry. Consider the *asymmetric measure*

$$\text{sim}(q, d_j) = A(q, d_j) = \frac{\sum_{k=1}^n \min(w_{kq}, w_{kj})}{\sum_{k=1}^n w_{kq}}.$$

In general,  $\text{sim}(d_i, d_j) \neq \text{sim}(d_j, d_i)$  because the denominator is only dependent upon the first argument. This behavior can be useful to assess inclusion relations between document vectors. That is, document  $d_i$  is included in  $d_j$  if and only if  $w_{ki} \neq 0 \Rightarrow w_{kj} \neq 0$  for  $k = 1, \dots, n$ . This inclusion property is often useful when clustering or classifying documents within an hierarchical framework.

### 3.5. Cosine

Since the classic vector space model represents documents by an  $n$ -dimensional vector, one can visualize the entire document collection as a set of  $n$ -dimensional vectors. From a geometric standpoint, the cosine of the angle between two vectors  $q$  and  $d_j$  is given by

$$\text{sim}(q, d_j) = C(A, B) = \sqrt{RP} = \frac{|A \cap B|}{\sqrt{|A||B|}} \quad (27)$$

$$\cong \frac{\vec{q} \cdot \vec{d}_j}{|\vec{q}| \times |\vec{d}_j|} = \frac{\sum_{k=1}^n w_{kq} w_{kj}}{\sqrt{\sum_{k=1}^n w_{kq}^2 \sum_{k=1}^n w_{kj}^2}}. \quad (28)$$

Note that Eq. (27) is also the geometric mean of precision and recall. Since this is a cosine measure with a normalizing factor in the denominator, all

similarities produced will be in the range  $[-1, 1]$ . The document vectors that are pointing in the most similar direction in  $n$ -dimensional space will receive the highest score.

### 3.6. Other Measures

Similarity measures are often application dependent, and different measures can be defined as hybrids of the measures previously mentioned. Other modifications to the aforementioned measures are not at all uncommon.<sup>d</sup> In general, if a measure can take into account any kind of *a priori* knowledge to help distinguish documents from each other, then that measure is likely to be more effective than one that does not. For example, [133] suggests that

$$\begin{aligned} sim(q, d_j) = & \alpha[\text{content identifier similarity}] \\ & + \beta[\text{objective term similarity}] \\ & + \gamma[\text{citation similarity}] \end{aligned}$$

would intuitively yield more effective precision and recall than an ordinary measure that ignored document structure simply because it takes into account more information.

### 3.7. Dissimilarity

Just as similarity measures can indicate how *close* one document is to another, dissimilarity or distance measures can show how *far* documents are from each other. A distance measure can be defined as  $1 - s$ , where  $s$  is some similarity measure with values in  $[0, 1]$ . Since distance is often associated with maps, there are two standard distance measures. The *Euclidean distance*, given by

$$dis(q, d_j) = d_E(q, d_j) = \sqrt{\sum_{k=1}^n (w_{kq} - w_{kj})^2},$$

is simply the straight-line distance between any two points in space, while the *Manhattan* or *city block distance*,

$$dis(q, d_j) = d_M(q, d_j) = \sum_{k=1}^n |w_{kq} - w_{kj}|,$$

---

<sup>d</sup>Note that the quadratic terms in the denominators of all the previous equations with the exception of the Cosine are oftentimes alternatively defined to be linear. This does not, however, affect the performance of the measure in question.

can be thought of as the distance between two points if one can only travel along the axes defined and could only make right angle turns. In the 2D case, this is analogous to walking across city blocks (hence the name). Like similarity measures, other distance measures exist and are often application dependent [13].

#### 4. Example

Since many of the similarity measures are rooted in IR, it follows to consider a simple example based on a vector-space IR model. Consider the document collection shown in Table 1.

The document collection consists of the titles of papers cited in this chapter, while the query has been constructed arbitrarily. Each title represents one document, and only the boldface words are considered tokens.<sup>e</sup> A term-by-document matrix  $A = [a_{ij}]$  can be constructed as in Fig. 4, where the terms correspond to the rows of the matrix and the documents are represented by the columns of the matrix. A vector representation of the query is also depicted in Fig. 4. The element  $a_{ij}$  is defined to be the frequency with which term  $i$  occurs within document  $j$ .

Table 1. Sample document collection and query.

T1: conceptual	D1: A <b>Search Engine</b> for 3D <b>Models</b>
T2: database	D2: Design and Implementation of a String <b>Database</b>
T3: dependence	<b>Query Language</b>
T4: documents	D3: Ranking of <b>Documents</b> by <b>Measures</b> Considering
T5: domain	<b>Conceptual Dependence</b> between Terms
T6: engine(s)	D4: Exploiting Hierarchical <b>Domain Structure</b> to
T7: information	Compute <b>Similarity</b>
T8: language	D5: An Approach for <b>Measuring Semantic Similarity</b>
T9: measur(es, ing)	between Words Using Multiple <b>Information</b> Sources
T10: models	D6: Determining <b>Semantic Similarity</b> among Entity
T11: ontologies	Classes from Different <b>Ontologies</b>
T12: query	D7: Strong <b>Similarity Measures</b> for Ordered Sets of
T13: retrieval	<b>Documents</b> in <b>Information Retrieval</b>
T14: search(ing)	
T15: semantic	Q: <b>Semantic Similarity Measures</b> Used by <b>Search</b>
T16: similarity	<b>Engines</b> and Other <b>Information Searching</b>
T17: structure	<b>Mechanisms</b>

<sup>e</sup>The non-boldface words are not considered to be part of the dictionary for simplification purposes. Here, tokens are defined to be dictionary words. In general, however, tokens can be defined to be word phrases or parts of a word.

$$[A|q] = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Fig. 4. Sample term-document matrix for the collection in Table 1.

Given the document vectors that represent the document collection and the query, each of the similarity measures and distance metrics discussed in Sec. 3 can be computed in a pairwise fashion as seen in Table 2. Note that each measure is assumed to be taken between query  $q$  and document  $d_j$  except the second asymmetric measure, which considers  $d_j$  then  $q$ .

The asymmetric measures demonstrate that document D5 is included in  $q$  and not *vice versa*. By inspection, one can infer that

$$J \leq D \leq C \leq O$$

holds for all pairs of arbitrary documents [45] asserts that this relation can be verified algebraically by elementary calculations. In this example, no

Table 2. Similarity measures between the query  $q$  and the document collection. The similarity measures depicted are Dice ( $D$ ), Overlap ( $O$ ), Jaccard ( $J$ ), Cosine ( $C$ ), and Asymmetric ( $A$ ).  $d_E$  and  $d_M$  denote the Euclidean and Manhattan distances, respectively.

	$D$	$O$	$J$	$C$	$A$	$A(d_j, q)$	$d_E$	$d_M$
D5	0.62	1	0.44	0.67	1	0.57	2.24	3
D1	0.5	1	0.33	0.58	0.67	0.29	2.45	6
D6	0.33	0.67	0.20	0.38	0.67	0.29	2.83	6
D7	0.31	0.50	0.18	0.33	0.50	0.29	3	7
D4	0.17	0.33	0.09	0.19	0.33	0.14	3.16	8
D3	0.15	0.25	0.08	0.17	0.25	0.14	3.32	9
D2	0	0	0	0	0	0	3.46	10

measure does a *better* job than another since the ultimate information need of the user is unknown. It should be noted, however, that some measures such as the Jaccard and Cosine do not score any two distinct documents identically with respect to the given query. This characteristic is one of the reasons why many retrieval systems use the Jaccard and Cosine measures as a baseline for evaluation.

## 5. Current Applications

Similarity measures will always play a vital role in any application where full automation (of a search) is desired. This section discusses three current applications of similarity measures, which is by no means an exhaustive list. Similarity measures are applied to fields such as speech recognition, video summarization, homeland security, and others in addition to the ones mentioned later in this section.

### 5.1. Multi-Dimensional Modeling

There are numerous text models available for use on the Web. Generally speaking, after text parsing, corpus analysis is performed. Parsing text into tokens or atomic units is usually not a very difficult process. Unfortunately, parsing cannot be easily generalized to other data types. According to [52], the information revolution for 3D data<sup>f</sup> is still in its relative infancy compared to other forms of data such as text, images, audio, and video. As computers become better able to handle more complex models of information, the number of models publicly available increases. In particular, methods to efficiently query 3D models need to be developed in order to search the growing number of 3D models available on the Web.

One of the core challenges of searching 3D models is deciding on which characteristics to index the models. Extending existing 2D indexing methods based on shape contours such as arc length parameterization to the equivalent 3D surface boundary parameterization problem is nontrivial. Shape matching methods that utilize geometric hashing techniques become increasingly more expensive to compute when adding the extra dimension to address the 3D case. In short, 2D techniques simply do not scale up well to their 3D counterparts.

---

<sup>f</sup>The specific type of 3D data considered here are 3D models of isolated objects that are often used in computer graphics. The models usually consist of coordinates, triangle mesh data, or other information needed to render 3D models.

As is often the case when attempting to develop a new standard method for querying a relatively new data type, much thought must be given to user interfaces and the type of queries allowable by the system. In the case of 3D data, sketch queries, text queries, and multi-modal queries are being investigated.

Sketch queries are performed by a user physically drawing a representation of an object or selecting a generic object from a predefined list. Once a query shape is generated, several shape matching algorithms can be applied. Algorithms exist that attempt to compare models based on their statistical properties such as shape histograms, moments, 2D shape distributions, etc. Such algorithms, however, tend to blend shape information from different parts of the object into one statistic. There is also work based on spherical harmonics that can identify shape descriptors that are rotation invariant. This allows users to query models without having to worry about the orientation of the model on its principal axes, which is one of the downfalls of many of the statistical similarity measures. Once two spherical harmonic descriptors are produced, the Euclidean distance (discussed earlier in Sec. 3.7) is computed to determine the dissimilarity between the two models.

Assuming manual construction of representative documents of 3D models, text queries can be performed over the set of models. Such queries and documents typically address function and more abstract qualities of the model, and any of the measures mentioned in Sec. 3 can be used. Multi-modal queries combine sketch and text queries into one measure. In most cases, text queries are effective at identifying classes of objects, while sketch queries are most useful at distinguishing the best object within a class.

## 5.2. *Hierarchical Clustering*

Often the actual words used in a query can affect the rankings presented to the user. For example, all of the similarity measures in Sec. 4 determined that D2 was unrelated to the query. A human judgment, however, would probably give D2 at least some similarity to the query (that paper is, after all, a reference for this chapter). To address this problem, many researchers are developing ways to abstract meaning from simple query terms [54, 97, 111, 129] all propose different methods to incorporate concept hierarchies or ontologies such as WordNet<sup>§</sup> to add context to a query word in an attempt to increase precision and recall.

---

<sup>§</sup>WordNet can be found at <http://wordnet.princeton.edu>.

Concept hierarchies are usually highly specialized thesauri that place words within a parent-child relationship. For example, a person may be a type of life form, while there are numerous types of people (e.g., male, female, child, athlete, student) under the concept of a person. Many hierarchies are subjective and their use is usually application dependent. These hierarchies, however, are an attempt to create basic knowledge about certain terms, which can ultimately lead to more effective retrieval. With a concept hierarchy, a similarity measure can become a function of the depth of a word within the hierarchy. For example, a query for a college student will be deemed more relevant to a person concept as opposed to an animal concept such as a bear.

If used effectively, ontologies can be used to perform query refinement. That is, given a query, replace a word within the query with its *synset*, or set of words that are its synonyms [129], with the hope that the synonyms will more fully cover the concept the original query word intended to identify. In some measures, similarity is not computed as a function of the query words themselves but as a function of the overlap between the neighborhoods of the query words in question. Cross-ontology similarity measures are also being investigated with marginal success. Since each ontology defines its word relationships in a potentially unique way, standard cross-ontology measures can be difficult to apply to ontologies in a general case.

### 5.3. Bioinformatics

Text can be used to represent data structures different from the typical text interpretation of words and sentences. In particular, data from bioinformatics are encoded into text strings, where each symbol in the string represents some biological entity such as a protein or nucleic acid. A particular pattern of nucleic acids in a DNA sequence may, for example, imply the presence of a certain gene that may be associated with a certain disease. As a result, biologists often attempt to find similarities between known patterns in DNA sequences and other suspected DNA sequences to predict disease likelihood and other characteristics.

Similarity between sequences in the computational molecular biology sense is often measured via alignments. The *relatedness* or alignment between two sequences is usually computed as the edit distance between two strings. The *edit* or *Levenshtein distance* is defined as the total number of operations required to transform one string into another, where valid operations are symbol replacement, insertion, or deletion. Variants of the edit

distance place different weights or costs for each operation type performed. According to [58], the edit distance can be interpreted in an evolutionary sense as the number of point mutations between two sequences. Other distance measures are also defined, and currently bioinformatics is attempting to find ways to compute similarity between complex, multi-dimensional gene models in an effort to understand the relationship between gene form and function.

## **6. Summary**

Similarity measures play a crucial role in information systems today. Comprehending the basic foundations of classic similarity measures such as the Dice, Overlap, Jaccard, and Cosine will help develop a better intuition about the behavior of many retrieval systems. As the complexity of data increases, methods for searching those data need to maintain a simple interface without compromising precision and recall. Often, hybrids or simple extensions of the classic measures are found to be effective in retrieving complex data. However, as data complexity increases to the point that extending existing models becomes computationally unfeasible, new methods need to be created to help organize that data in some manner.

## CHAPTER 4

### DECISION TREES

Olufemi A. Omitaomu

*Department of Industrial & Information Engineering  
University of Tennessee, 411 East Stadium Hall  
Knoxville, TN 37996-0700  
fermi@utk.edu*

#### Overview

Decision trees are one of the fundamental techniques used in data mining. They are tree-like structures used for classification, clustering, feature selection, and prediction. Decision trees are easily interpretable and intuitive for humans. They are well suited for high-dimensional applications. Decision trees are fast and usually produce high-quality solutions. Decision tree objectives are consistent with the goals of data mining and knowledge discovery. This chapter reviews the concept of decision trees in data mining.

**Keywords:** Decision trees, classification, clustering, prediction, feature selection.

#### 1. Introduction

A decision tree (DT) is a tree-like structure used for classification, decision theory, clustering, and prediction functions. It depicts rules for dividing training data into groups based on the regularities in the data. A DT can be used for categorical and continuous response variables. When the response variables are continuous, the DT is often referred to as a *regression tree*. If the response variables are categorical, it is called a *classification tree* [56]. However, the same concepts apply to both types of trees. Decision trees are widely used in computer science for data structures, in medical sciences for diagnosis, in botany for classification, in psychology for decision theory

[153], and in economic analysis for evaluating investment alternatives [23]. These trees may differ in how they are created. For example, in some cases the trees are created top to bottom, while in other cases they are created from left to right. Decision trees have been described as universal approximators since they map linear and nonlinear relationships [91]. However, they do not require as much training data as other universal approximators, such as neural networks.

A decision tree consists of a root and internal nodes. The root and the internal nodes are labeled with questions in order to find a solution to the problem under consideration. The root node is the first state of a DT. This node is assigned to all of the examples from the training data. If all examples belong to the same group, no further decisions need to be made to split the data set. If the examples in this node belong to two or more groups, a test is made at the node that results in a split. A DT is binary if each node is split into two parts, and it is nonbinary (multi-branch) if each node is split into three or more parts (Fig. 5). According to Wilkinson [153], multi-branch trees are not superior to binary trees because “each tree is a permutation of the other.”

If an internal node cannot be split further, it becomes a terminal node (Fig. 5). The paths to each internal or terminal node are mutually exclusive; that is, no more than one group can possibly be chosen. The process is repeated for each of the internal nodes until a completely discriminating tree is obtained or the terminal node is reached. When a terminal node is reached, its stored value (or content) is returned.

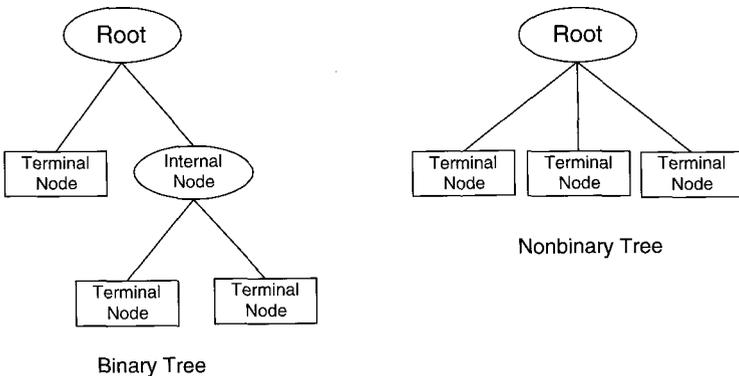


Fig. 5. Types of decision tree.

## 2. Motivation

Decision trees are popular for partitioning data and identifying local structures in small and large databases. Decision tree models have two objectives: producing an accurate classifier, and understanding the predictive structure of the problem [20]. The first goal deals with the accuracy of decision tree classification, while the second goal aims at “developing understandable patterns that can be interpreted as interesting knowledge” [46]. In addition, there are some unique characteristics of decision trees that make them the first choice for data mining experts. Decision trees are appealing because of the clear depiction of the relationships between the input data and the target outputs. They accept several types of variables: nominal, ordinal, and interval. A variable can be of any type regardless of whether it serves as an input or as the target. A DT can be implemented with little or no consideration for converting *odd* variables (e.g., opinions, biases, or risks) to more appealing types. Decision trees are also robust with respect to missing values and distribution assumptions about the inputs, and well suited for high-dimensional applications.

DTs can produce fast nonlinear prediction methods and may employ dynamic feature selection [153]. Several approaches, such as principal component analysis and decision trees, are used to filter or transform several features into a reduced number of features. Using DTs for feature selection is particularly advantageous in large feature spaces. The selected features are the outputs, and the unused features are deleted from the database. The reduced database is then processed by other time-consuming prediction methods, such as neural networks. Decision trees are easily interpretable, amenable to graphical display, and intuitive for humans. As a tree size increases, however, the quality of such explanation decreases. The size of a tree can grow much larger since the rules are mutually exclusive. Decision trees are also excellent benchmarks for evaluating the performance of other techniques.

For the remaining part of this chapter, decision trees will be examined from a data mining perspective. Section 4.3 discusses decision tree algorithms, including methods of improving the algorithms. An example of the use of decision tree algorithms is discussed in Sec. 4.4. Real world applications of decision trees are discussed in Sec. 4.5. Section 4.6 presents a summary of the chapter.

### 3. Decision Tree Algorithms

A decision tree model consists of two parts: creating the tree and applying the tree to the database [42]. To achieve this, decision trees use several different algorithms. The most widely-used algorithms by computer scientists are *ID3*, *C4.5*, and *C5.0* [126]. The most popular algorithm in the statistical community is Classification and Regression Trees (*CART*) [20]. This algorithm helps decision trees gain credibility and acceptance in the statistics community [59]. It creates binary splits on nominal or interval inputs for a nominal, ordinal, or interval target. The first version of *C4.5* and *C5.0* were limited to categorical predictors; however, the most recent versions are similar to *CART* [56]. Other algorithms include Chi-Square Automatic Interaction Detection (*CHAID*) for categorical outputs [80], *CLS*, *AID*, *TREEDISC*, and *Angoss KnowledgeSEEKER*. These algorithms use different approaches for splitting variables. *CART* uses the statistical approach, while *CLS*, *ID3*, and *C4.5* use an approach in which the number of branches off a nonterminal node is equal to the number of possible categories. Another common approach, used by *AID*, *CHAID*, and *TREEDISC*, is the one in which the number of nodes on a nonterminal node varies from two to the maximum number of possible categories. *Angoss KnowledgeSEEKER* uses a combination of these approaches. Each algorithm employs different mathematical processes to determine how to group and rank variables.

According to Quinlan [126] the original idea of constructing a decision tree dates back to the work of Hoveland and Hunt in the late 1950s. The skeleton of Hunt's methods for constructing a decision tree from a set  $T$  of training cases is as follows:

Let the classes be denoted by  $\{C_1, C_2, \dots, C_n\}$ . There are three possibilities:

- (i)  $T$  contains one or more cases, but all belonging to a single class  $C_j$ .  
The decision tree for  $T$  is a leaf identifying class  $C_j$ .
- (ii)  $T$  contains no cases. The decision tree is also a leaf in this case, but the class to be associated with the leaf must be determined from sources other than  $T$ .
- (iii)  $T$  contains cases that belong to a mixture of classes.  $T$  is partitioned into subsets  $T_1, T_2, \dots, T_k$ , where  $T_i$  contains all cases in  $T$  that have outcome  $O_i$  of the chosen test. The decision tree for  $T$  consists of a decision node identifying the test, and one branch for each possible outcome. This process is applied recursively to each subset of the training

cases, so that the  $i$ th branch leads to the decision tree constructed from the subset  $T_i$  of the training cases.

Generally, a decision tree algorithm is most appropriate for the third case. In this case, the decision tree algorithm can be stated as follows:

- From the training data set, identify a target variable and a set of input variables.
- Examine each input variable one at a time:
  - Create two or more groupings of the values of the input variables, and measure how similar items are within each group and how different items are between groups.
  - Select the grouping that maximizes similarity within groupings and differences between groupings.
- Once the groupings have been calculated for each input variable, select the single input variable that maximizes similarity within groupings and differences between groupings.

This process is repeated in each group that contains a convincing percentage of information in the original data. The process is not terminated until all divisible groups have been divided.

### 3.1. *ID3 Algorithm*

Below is the decision tree algorithm for *ID3* that describes the general layout for DT algorithms [124]. This algorithm uses *gain ratio* (see Sec. 3.2) as the evaluating test.

### 3.2. *Evaluating Tests*

The final classification of a decision tree model depends on the choice of tests used in evaluating the training data. DT algorithms are *greedy* in the sense that once a test has been selected to partition the training set, the consequences of alternative choices are not explored. Therefore, to ensure a stable and predictable final tree the choice of tests must be correct. Several tests could be evaluated so that the most appropriate test can be chosen. *C4.5* contains mechanisms for proposing three types of tests [126]:

- The *standard* test on a discrete attribute, with one outcome and branch for each possible value of that attribute.

---

**Algorithm 3** ID3 Decision Tree Algorithm
 

---

**Given** Examples ( $S$ ); Target attribute ( $C$ ); Attributes ( $R$ )

**Initialize** Root

**Function** ID3 ( $S, C, R$ )

**Create** a Root node for the tree

**IF**  $S = \text{empty}$ , return a single node with value Failure;

**IF**  $S = C$ , return a single node with  $C$ ;

**IF**  $R = \text{empty}$ , return a single node with most frequent target attribute ( $C$ );

**ELSE**

**BEGIN**

Let  $D$  be the attribute with largest Gain Ratio ( $D, S$ ) among attributes in  $R$ ;

Let  $\{d_j | j = 1, 2, \dots, n\}$  be the values of attribute  $D$ ;

Let  $\{S_j | j = 1, 2, \dots, n\}$  be the subsets of  $S$  consisting respectively of records with value  $d_j$  for attribute  $D$ ;

Return a tree with root labeled  $D$  arcs  $d_1, d_2, \dots, d_n$  going respectively to the trees;

**For** each branch in the tree

**IF**  $S = \text{empty}$ , add a new branch with most frequent  $C$ ;

**ELSE**

$ID3(S_1, C, R - \{D\}), ID3(S_2, C, R - \{D\}), \dots, ID3(S_n, C, R - \{D\})$

**END**  $ID3$

**Return** Root

---

- A more complex test, based on a discrete attribute, in which the possible values are allocated to a variable number of groups with one outcome for each group rather than each value.
- If attribute  $A$  has continuous numeric values, a binary test with outcomes  $A \leq Z$  and  $A > Z$  for some threshold  $Z$ .

These tests are evaluated using the *gain* or *gain ratio* criterion. The gain criterion selects a test to maximize the mutual information between the test and the class. The process of determining the gain for a test is as follows [126]:

Imagine selecting one case at random from a set  $S$  of cases and announcing that it belongs to some class  $C_j$ . Let  $freq(C_j, S)$  denote the frequency

of class  $C_j$  cases in set  $S$  so that this message has the probability

$$\frac{\text{freq}(C_j, S)}{|S|}.$$

The information the message conveys is defined by

$$-\log_2 \left( \frac{\text{freq}(C_j, S)}{|S|} \right) \text{ bits.}$$

The expected information from such a message pertaining to class membership is the sum over the classes in proportion to their frequencies in  $S$ ; that is,

$$\text{Info}(S) = - \sum_{j=1}^k \frac{\text{freq}(C_j, S)}{|S|} \times \log_2 \left( \frac{\text{freq}(C_j, S)}{|S|} \right) \text{ bits.}$$

When applied to the set of training cases,  $\text{Info}(T)$  measures the average amount of information needed to identify the class of a case in set  $T$ . This quantity is also known as the *entropy* of the set  $T$ .

Now consider a similar measurement after  $T$  has been partitioned (denoted by  $T_i$ ) in accordance with the  $n$  outcomes of a test  $X$ . The expected information requirement is the weighted sum over the  $n$  subsets:

$$\text{Info}_X(T) = \sum_{i=1}^n \frac{|T_i|}{|T|} \times \text{Info}(T_i).$$

The quantity

$$\text{gain}(X) = \text{Info}(T) - \text{Info}_X(T)$$

measures the information that is gained by partitioning  $T$  in accordance to the test  $X$ .

Even though the gain criterion yields good results, it has a serious deficiency — it is biased towards tests with many outcomes. To correct this deficiency, a gain ratio criterion has been developed [126]. The bias in the gain criterion can be corrected by normalizing the apparent gain of tests. By analogy, the definition of *split info* is given by

$$\text{split info}(X) = - \sum_{i=1}^n \frac{|T_i|}{|T|} \times \log_2 \left( \frac{|T_i|}{|T|} \right).$$

According to [126], this represents the “potential information generated by dividing  $T$  into  $n$  subsets, whereas the information gain measures the

information relevant to classification that arises from the same division.” Then,

$$\text{gain ratio}(X) = \frac{\text{gain}(X)}{\text{split info}(X)}$$

expresses the useful portion of the generated information by the split (that appears useful for classification). The gain ratio selects a test to maximize the ratio above, subject to the constraint that the information gain must be large — at least as large as the average gain over all tests examined.

### 3.3. Selection of Splitting Variable

All DT algorithms search for the variable with the best split. Selecting variables with the best splits will help in constructing a less complicated tree. According to Loh and Shih [100] splitting each variable significantly biases the selection towards nominal variables with many categories. This may be a concern when a decision tree is used for feature selection, rather than when used for prediction. An algorithm that searches for a better fit on each variable will typically require more computation time.

### 3.4. Stopping Criteria

A decision tree is easy to interpret when its size is manageable. Therefore, the size of a tree may be of greater importance than splitting of variables. Trees should not be too small nor too large. Smaller trees do not describe the training data very well; therefore, they may not perform well on new data sets. When trees are too large, they have several leaves with little data to make any dependable predictions when applied to new data. Stopping criteria have been proposed to handle this problem. However, stopping the algorithm early may not produce a tree that has taken into consideration almost all information in the training data. Allowing the algorithm to make use of all data available, however, results in a larger tree. The latter scenario is usually more desirable because it guarantees that all the information has been captured. Some algorithms including *CHAID* have a stopping rule that accounts for the predictive reliability of the data. Unfortunately, such a stopping rule turns out to be problematic in three related aspects: choice of statistical test, accommodations for multiple comparisons, and the choice of a threshold [80].

### 3.5. Tree Pruning

Many experts agree that stopping rules cannot work based on the above mentioned problems. Therefore, *tree pruning* is required to reduce full grown trees into manageable sizes. Tree pruning is a technique developed for trimming larger trees to appropriate sizes. The pruning process may evaluate subtrees instead of individual splits. Pruning is necessary to avoid over-fitting the data. Error estimation techniques play a major role in tree pruning [8]. There are several tree pruning algorithms. One tree pruning algorithm is *reduced-error pruning*. This algorithm finds the subtree with the smallest error on pruning data [125]. Another tree pruning algorithm is *cost-complexity pruning* [20]. In this case, the algorithm uses a separate “pruning data set” to evaluate the subtrees. Subtrees that over-fit the training data will perform poorly on the pruning data. *Cost-complexity pruning* is another tree pruning algorithm. This algorithm evaluates different sizes of the subtree in sequence using new data. The larger tree is trimmed to the subtree with the optimum assessment. This algorithm is based on cross validation. Another popular pruning algorithm called *pessimistic pruning* [126] is based on an inflated error rate of the training data in each node. However, tree pruning is not required if decision trees are used for feature selection. The objective in this case is “the set of features selected by the tree, not the tree structure or the numerical threshold” [153].

### 3.6. Stability of Decision Trees

Tree pruning, however, does not guarantee a stable tree. Decision trees, like some other classifiers such as neural networks, are usually not stable. The instability of a decision tree refers to when the same algorithm applied to slightly different data produces a very different model. The unstable nature of tree-based models makes the interpretation of trees tricky. Several methods, including *bagging*, *arcing*, and *boosting*, have been developed to make trees more stable and provide accurate predictions [93]. Bagging (bootstrap aggregation) is the process of creating an ensemble of models using the same algorithm on data sets sampled with replacement from a single training data set [18]. Bagging uses independent samples of the original data. Arcing (adaptive resampling and combining) uses a sample of the training data that the current ensemble predicts relatively poorly. That is, the first model trains on all the original data. Cases that the first model incorrectly classified are sampled for the second training data set. A second

model is trained, and the two models are combined. Cases for which the combined model performed poorly are sampled for the third training data. This process is repeated until a more stable model is developed. This process applies only to categorical targets [19]. Boosting was formulated by Freund and Schapire [50], but reformulated without a new name by Friedman, Hastie, and Tibshirani [51]. Boosting is more commonly used than arcing, but they essentially mean the same thing. The reformulated method creates an ensemble consisting of weighted averages of models. However, the same data sets are used for all the models.

#### 4. Example: Classification of University Students

This section provides an example to illustrate the steps involved in constructing decision trees. This example from [42] is simple, but it is detailed enough to illustrate the important points in using a decision tree algorithms.

Students in a particular university are to be classified as short, medium, or tall based on the information in the database using a decision tree algorithm. Assume that the database schema is {name, address, gender, height, age, year, major}. The decision tree is constructed using the following steps:

- (i) Selection of splitting attributes. The selection of attributes to use for the classification is the first step. One of the possible attributes to select is height. Name, address, year, and major are not important because they are irrelevant to the objective of this problem. Therefore, height, age, and gender are chosen. A female student whose height is greater than 1.8m is defined as tall, less than 1.3m is short, and at least 1.3m but less than 1.8m is considered of medium height. Moreover, a male student whose height is greater than 2m is tall, less than 1.5m is considered short, and at least 1.5m but less than 2m is defined as medium.
- (ii) Filtration of the training data. The training data is filtered to remove outliers (based on the criteria set above). In addition, students less than 17 years of age will be considered outliers since they are not typical of most university students. With the removal of outliers, the number of attributes can actually be reduced to two: gender and height. This reduction helps construct a less complicated tree.
- (iii) Constructing the tree. The decision tree algorithm constructs a tree based on a sample of the database with known classification values. This training data forms the basis for how the tree is constructed. The root is split into two internal nodes (branches), being the maximum

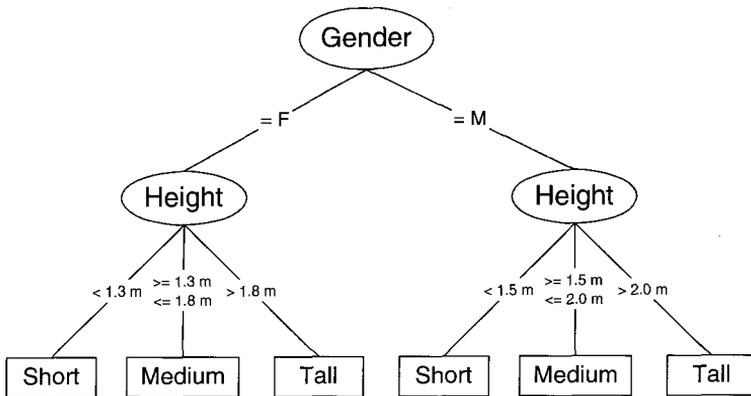


Fig. 6. A decision tree with two internal nodes.

number of categories possible for the labeled question: What is the gender of the student? Each internal node can be split into two or three branches depending on the algorithm used. If the internal nodes are split into three branches (the maximum number of categories possible), these branches are the terminal nodes, so no further division is necessary. One of the possible resulting trees in this case is shown in Fig. 6.

Nevertheless, if each internal node is split into two branches, one of the branches is a terminal node and the other is another internal node. These new internal nodes are further divided into two branches to complete the tree construction. A possible resulting tree in this other case is shown in Fig. 7.

**Step 4 — Interpretation of the Final Tree:** Both trees show that a female student is short, if her height is less than 1.3 m; tall, if her height is more than 1.8 m; and medium, otherwise. A similar conclusion can be said of a male student. These trees are simple and easy to interpret, so they do not require pruning.

## 5. Applications of Decision Tree Algorithms

Decision tree algorithms are widely used in several areas in business, science, government, and engineering. There is hardly an area in which

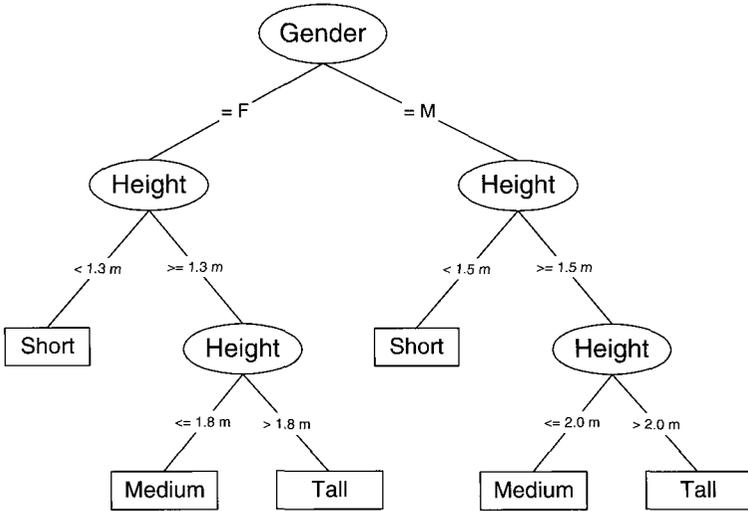


Fig. 7. A decision tree with four internal nodes.

decision tree algorithms have not been used. Decision trees have been used in the retail businesses to study customers' purchasing patterns for business planning purposes. They are also used in the medical sciences to group and identify potential risks associated with certain diseases. Decision trees are used by banks and credit card companies to identify potential credit risk customers, by chemometrics to classify chemical data [67], and by geographers to classify land cover [118]. Decision trees can be used by government agencies to identify the features of a potential terrorist. Decision tree algorithms are becoming so prevalent that current research topics include methods for enhancing decision tree algorithms and of combining decision tree algorithms with other computational techniques, such as fuzzy systems, neural networks, and genetic algorithms. For example, a fuzzy decision tree technique called *soft decision tree* has been developed by Olaru and Wehenkel [117].

## 6. Summary

In this chapter the basic concepts of decision tree models as they relate to data mining were presented. Decision trees are used for several data mining functions such as clustering, feature selection, prediction, and classification.

They are easy to interpret, amenable to graphical display, and intuitive to humans. DTs are robust with respect to missing data and can be used with nominal, interval, and ordinal variables. Current research involves combining decision tree algorithms with other computational techniques, such as fuzzy systems, neural networks, and genetic algorithms.

This page is intentionally left blank

## CHAPTER 5

### GENETIC ALGORITHMS

Geoffrey Mazeroff

*Department of Computer Science, University of Tennessee  
203 Claxton Complex, Knoxville, TN 37996-3450, USA  
mazeroff@cs.utk.edu*

#### Overview

This chapter describes genetic algorithms in relation to optimization-based data mining applications. Emphasis is placed on introducing terminology and the fundamental phases of a standard genetic algorithm framework. An application to the traveling-salesman problem is discussed, and references to current genetic algorithm use are presented. The chapter concludes with a synopsis of genetic algorithm applicability to general data mining tasks.

**Keywords:** Allele set, crossover, evolutionary algorithm, fitness function, genetic algorithm, mutation, traveling-salesman.

#### 1. Introduction

There exists a large class of problems for which no computationally-efficient algorithms have been developed. Many of these problems contain globally optimal solutions within a large search space. For small optimization applications, one can successfully employ exhaustive search methods. However, such methods become impractical as the applications become more complex and the amount of data becomes significantly large. It is often possible to find an effective yet computationally practical algorithm whose solution is approximately optimal. One approach in dealing with such problems is the use of genetic algorithms, which are based on the principles of natural evolution.

Table 3. Basic concepts in genetic algorithms [77].

Concept in Natural Evolution	Concept in Genetic Algorithms
Chromosome	String
Gene	Features in the string
Locus	Position in the string
Allele	Position value (usually 0 or 1)
Genotype	String structure
Phenotype	Set of characteristics (features)

Genetic algorithms (GAs) are stochastic-optimization methods first proposed by James Holland in 1975 [69]. The concept of such an algorithm was revealed by several biologists who were performing computer simulations of natural genetic systems. The terminology and concepts used in GA theory have analogs in the field of biology. See Table 3.

GAs encode each point in a solution space into a string called a *chromosome*. Unlike other data mining samples, which are often predefined and not likely to change as the data mining algorithm is carried out, samples in GAs change or “evolve” during their execution. The features of interest are located in different portions of the string. The string structures in the chromosomes undergo operations similar to the natural-evolution process to yield increasingly optimal solutions. The quality of the solution is based on a “fitness” value, which relates to the objective function for the optimization problem.

In data mining, GAs can be used for prediction, clustering, and association rule inference [42]. For each of these uses, a starting model is assumed, then iteratively refined to find the optimal model for the given application. One particular algorithm does not necessarily hold for all problems; however, the general concepts used in applying GAs to data mining remain the same.

## 2. Motivation

A common aphorism in the realm of data mining is “the curse of dimensionality.” This situation occurs when dealing with problems that have a large number of attributes. High dimensionality imposes a larger search space in a combinatorially explosive manner. Furthermore, an increased number of training samples are typically required to generate reliable results. GAs are

of interest because they provide an alternative to traditional machine learning algorithms, which tend to perform poorly when the search space is large. GAs have been shown to solve problems that were considered intractable due to their expansive search spaces. In short, GAs can help “break” the curse of dimensionality [104].

This chapter covers the fundamental components of genetic algorithms, beginning with the choice of an initial population and ending with a description of how the algorithm as a whole allows the population to “evolve” over several iterations. Next, an example GA application in the area of optimization, specifically the traveling-salesman problem, will be discussed. Finally, several current and future applications of GAs will be presented.

### 3. Fundamentals

GAs consist of five major phases: schema encoding, fitness evaluation, parent selection, crossover, and mutation (see Fig. 8).

Once the encoding schema has been applied, the remaining phases recur iteratively, thus causing the initial population to evolve until a specific criterion is met. The subsequent sections present definitions of these components and the corresponding notations. Each section also contains an explanation of the given topic in relation to a small-scale example; however, general examples of GA applications are presented in Sec. 4.

The example used to demonstrate the phases of a GA in the following subsections attempts to optimize

$$f(x) = x^2.$$

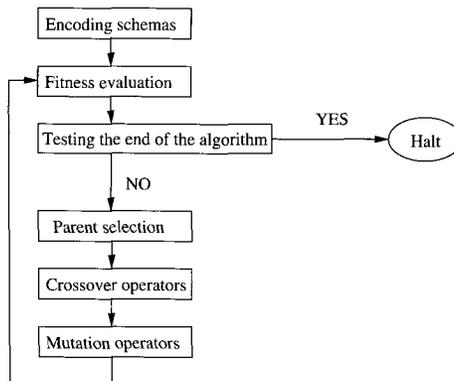


Fig. 8. Major phases of a genetic algorithm [77].

The goal is to find  $x$  from the range  $[0, 31] \in \mathbf{N}$  which maximizes  $f(x)$ . The approximate optimal solution is relatively easy to find while demonstrating the development and use of a GA.

### 3.1. Encoding Schema and Initialization

The first phase of a genetic algorithm is to design a representation of a solution for the given problem. Although solution forms vary by application, the most common representation is a string of characters used for feature representation. The characters are members of a fixed alphabet whose size determines the number of features that can be represented by each character. In general, solution encodings are binary strings [127]. Each feature's value can be coded with a string from a set of discrete values called the *allele set*, which is defined based on the needs of the problem. The encoding of the allele set must be "minimal but completely expressive" [77].

Encoding schemes provide a way of mapping problem-specific knowledge into the framework of GAs. A set of all feature values encoded into a string represents a *chromosome*. A set of chromosomes forms a collection called a *population*. The initial population can be established in several ways, such as a random set of chromosomes or a predefined starting set. The size of the population also has an impact on GA performance as well as the quality of the resulting solution. If the initial population is too small, the GA may converge quickly, thus finding only a local optimum. Conversely, if the initial population is too large, the GA may waste computational resources and will most likely require more time to converge.

With regard to the example, the range of the solution is analyzed to determine how the allele set is to be coded. Because the range is  $[0,31]$ , a minimum five-bit code is needed (i.e.,  $2^5 = 32$ ). The mapping from a real number to a binary code and *vice versa* is defined by the relations

$$\text{Code} = \text{binary}(x_{\text{decimal}})$$

and

$$x = \text{decimal}(\text{Code}_{\text{binary}}).$$

For example,  $x = 5$  has a corresponding code of 00101. Likewise, the code 11001 represents the decimal value  $x = 25$ . The initial population will

consist of four randomly chosen chromosomes:

$$\text{CR}_1 = 01101,$$

$$\text{CR}_2 = 11000,$$

$$\text{CR}_3 = 01000,$$

$$\text{CR}_4 = 10011.$$

### 3.2. Fitness Evaluation

After creating a population, a fitness value is calculated for each member in the population because each chromosome is a candidate solution. The *fitness* of a solution is a comparative measure for judging the quality (optimality) of candidate solutions. The choice of an appropriate *fitness function* depends significantly on the given problem. In certain cases complex analytical formulas work best, whereas observations from real-world problem settings work better in other scenarios.

Fitness values are typically positive and can represent rankings of members in a population. The latter approach has the advantage of not requiring complete accuracy, as long as the fitness function can provide the correct ranking information.

For the example, the four chromosomes  $\text{CR}_1$  to  $\text{CR}_4$  correspond to the following values for input variable  $x$ :

$$x_1(\text{CR}_1) = 13,$$

$$x_2(\text{CR}_2) = 24,$$

$$x_3(\text{CR}_3) = 8,$$

$$x_4(\text{CR}_4) = 19.$$

The evaluation function would rate them respectively as

$$f(x_1) = 13^2 = 169,$$

$$f(x_2) = 24^2 = 576,$$

$$f(x_3) = 8^2 = 64,$$

$$f(x_4) = 19^2 = 361.$$

An additional metric can be calculated, which [77] defines as the “expected reproduction.” This metric indicates the likelihood of certain chromosomes being reproduced in the next generation. The expected reproduction  $E_R$  in

this example is calculated using

$$E_R(x) = \frac{f(x)}{\bar{f}},$$

where  $\bar{f}$  is the mean value of  $f(x)$ . The higher the value of  $E_R$ , the more likely the chromosome will be reproduced in the next generation. The  $E_R$  values for each of the chromosomes are

$$E_R(x_1) = 0.58,$$

$$E_R(x_2) = 1.97,$$

$$E_R(x_3) = 0.22,$$

$$E_R(x_4) = 1.23.$$

### 3.3. Selection

Once the initial population and fitness function have been defined, a new population must be created. The *selection* operation determines which parent chromosomes are involved in producing the next generation of offspring. Typically, parents are selected for mating with a probability that is proportional to their fitness values. The most common way to perform selection is to define the probability  $p_i$  of selecting the  $i$ th chromosome

$$p_i = \frac{f_i}{\sum_{k=1}^n f_k},$$

where  $n$  is the population size and  $f_i$  is the fitness value for the  $i$ th chromosome. The effect of this method is to allow members with above-average values to reproduce, thus replacing members with below-average fitness values.

The process of selecting a chromosome is analogous to a “roulette wheel” with slots whose sizes correspond to fitness for each chromosome [77]. There are four steps for constructing the roulette wheel:

- (i) Calculate the fitness value  $f(v_i)$  for each chromosome  $v_i$ .
- (ii) Find the total fitness of the population:

$$F = \sum_{i=1}^n f(v_i).$$

- (iii) Calculate the probability of a selection  $p_i$  for each chromosome  $v_i$ :

$$p_i = \frac{f(v_i)}{F}.$$

(iv) Calculate a cumulative probability  $q_i$  after each chromosome  $v_i$  is included:

$$q_i = \sum_{j=1}^i p_j,$$

where  $q_i$  increases from 0 to a maximum value of 1. Because probabilities lie in the range  $[0,1]$ , the corresponding maximum for  $q_i$  shows that all chromosomes from the population are included.

The selection process is based on “spinning the roulette wheel”  $n$  times, where  $n$  is the size of the population. A single chromosome for a new population is selected for each spin. The following algorithm formalizes the selection process.

- (i) Generate a random number  $r$  from the range  $[0, 1]$ .
- (ii) If  $r < q_1$  then select the first chromosome  $v_1$ ; otherwise, select the  $i$ th chromosome  $v_i$  such that  $q_{i-1} < r \leq q_i$ .

Some chromosomes may be selected more than once; however, this is in accordance with GA theory [127]. Genetic algorithms maintain a population of potential solutions that undergo a simulated evolution. In each generation the “more fit” solutions reproduce while the “weaker” solutions die. A chromosome may be selected depending on the objective function value or the fitness function value. For maximization problems, a higher fitness value for a given chromosome corresponds to a higher likelihood of it being selected for the next generation.

Returning to the example, the probabilities of selection,  $p_i$ , are

$$p_1 = 0.14,$$

$$p_2 = 0.49,$$

$$p_3 = 0.06,$$

$$p_4 = 0.31.$$

A corresponding roulette wheel can be constructed to select chromosomes for the next population (see Fig. 9).

### 3.4. Crossover

The core of genetic algorithms involves exchanging chromosome information between highly-fit individuals. To exploit similarities between chromosomes, a *crossover function* is used. The probability of crossover,  $p_{cr}$ , is

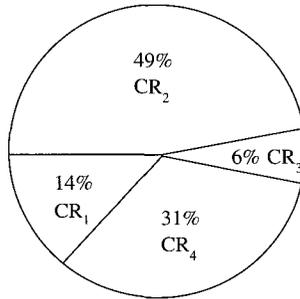


Fig. 9. Roulette wheel for selection of the next population [77].

the parameter that defines the expected number of chromosomes,  $p_{cr} \times n$ , which undergo the crossover operation. The following steps formalize the selection (or exclusion) of each chromosome for crossover:

- (i) Generate a random number  $r$  from the range  $[0, 1]$ .
- (ii) If  $r < p_{cr}$ , select the given chromosome for crossover.

For example, if  $p_{cr} = 1$ , all chromosomes in the population will be subject to the crossover operation. Similarly, if  $p_{cr} = 0.5$ , only half of the population will be subject to crossover, while the other half remains unchanged.

Two types of crossover functions are discussed here: one-point crossover and two-point crossover. *One-point* crossover involves a random selection of a crossover point on the genetic code. Given two parent chromosomes, the values in the chromosome to the left of the crossover point are swapped. *Two-point* crossover involves the selection of two points on the genetic code. In this scenario, the values in the chromosomes between the two crossover points are swapped between the two parents. Examples of one- and two-point crossover are shown in Fig. 10.<sup>h</sup>

In the general case,  $n$ -point crossover can be defined, where parts of strings between points 1 and 2, 3 and 4, and ultimately  $n - 1$  and  $n$  are swapped.

The process of crossover is analogous to the evolutionary process of exchanging gene segments between two parents and then passing them on to their children. This process allows children to inherit desirable traits from their increasingly-optimal parents.

<sup>h</sup>Part(a) of Fig. 10 is corrected from the original source.

Selected point for one-point crossover (after fourth position in the string)



a) One-point crossover

Selected points for two-point crossover (after second and fifth positions in the string)



b) Two-point crossover

Fig. 10. Crossover operators [77].

Pertaining to the example, the probability of crossover is set to 1,

$$p_{cr} = 1,$$

indicating that all of the chromosomes will be included in the crossover operation. Two random pairs of parents are selected for crossover: the randomly selected crossover point (for one-point crossover) is after the third position in the string. After crossover the selected strings (crossover bits indicated by underlines)

$$\begin{aligned} \text{First pair} \quad \text{CR}_1 &= 01101 \\ &\text{CR}_2 = 11000 \end{aligned}$$

$$\begin{aligned} \text{Second pair} \quad \text{CR}_2 &= 11000 \\ &\text{CR}_4 = 10011 \end{aligned}$$

will become a new population:

$$\begin{aligned} \text{CR}'_1 &= 01100, \\ \text{CR}'_2 &= 11001, \\ \text{CR}'_3 &= 11011, \\ \text{CR}'_4 &= 10000. \end{aligned}$$

### 3.5. Mutation

Even though crossover exploits the potential of existing chromosomes, the population may not contain all the encoded information needed to solve

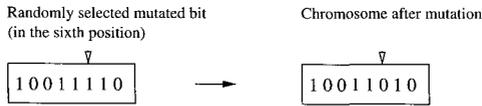


Fig. 11. Mutation operator [77].

a given problem. To address this issue, a *mutation* operator capable of generating new “genetic” information is introduced. The most common method of implementing mutation on binary strings is to invert a given bit based on a probability-based mutation rate (see Fig. 11).

A mutation operator is effective in preventing any single bit from converging to a value throughout the entire population. More importantly, the mutation operator can prevent the population from converging and “stagnating” at local optima. The mutation rate is typically set to a low probability such that “good” chromosomes obtained from crossover are not lost. High mutation rates cause the GA performance to approach that of a “primitive random search” [77].

In the example problem, the probability of mutation is 0.1%, meaning that only one out of every 1,000 bits will be mutated. Given the simplistic example, several iterations of the algorithm would have to take place before any significant mutation could be observed.

### 3.6. Iterative Evolution

In the natural evolutionary process, the aforementioned steps occur simultaneously. In the context of GAs, the phases occur iteratively to facilitate implementation and experimentation. As shown in Fig. 10, the process of evaluating fitness, selecting parents, and applying crossover and mutation operators occurs until a given stopping criterion is met. Several example stopping criteria are as follows:

- a certain number of iterations is reached;
- the difference between two successive iterations is less than some pre-defined threshold;
- a sufficient fitness value is achieved;
- a certain amount of time has elapsed.

Continuing the example from Sec. 3.4 (crossover), statistics for the second generation can be computed as shown in Table 4.

Table 4. Evaluation of the second generation of chromosomes [77].

CR <sub><i>i</i></sub>	Code	<i>x</i>	<i>f(x)</i>	<i>p<sub>i</sub></i>	Expected Reproduction
1	01100	12	144	0.08	0.32
2	11001	25	625	0.36	1.44
3	11011	27	729	0.42	1.68
4	10000	16	256	0.14	0.56

Table 5. First and second generation statistics.

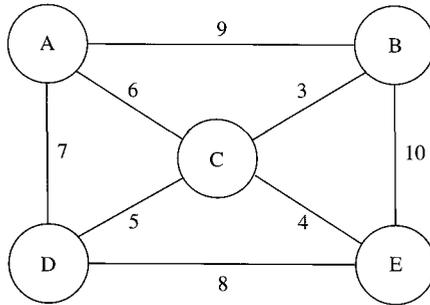
First Generation	Second Generation
$\Sigma f(x) = 1170$	$\Sigma f(x) = 1754$
$\bar{f}(x) = 293$	$\bar{f}(x) = 439$
max = 576	max = 729

Comparing the sum of the evaluation function values for each of the chromosomes from the first and second generations as shown in Table 5, one notes that the new population is approaching the maximum of the function  $f(x)$ . As more iterations occur, the solution population evolves to contain the optimal solution of  $x = 31$ .

#### 4. Example: The Traveling-Salesman

The traveling-salesman problem is a well-known problem in computational theory. In the problem a salesman must visit  $n$  cities. The cities and the paths connecting them can be modeled as a complete graph with  $n$  vertices that the salesman must “tour.” This tour starts and ends at the same city, and the salesman must visit each city exactly once. Additionally, the paths (i.e., edges in the graph) are weighted; therefore, various tours may have different “costs” associated with them. The optimization version of the traveling-salesman problem involves finding the *minimum cost tour* given a set of cities and weighted paths connecting them.

Formally, the traveling-salesman problem is defined as:  $G = (V, E)$  is a complete graph,  $c$  is a cost function from  $V \times V \rightarrow Z$ ,  $k \in Z$ , and  $G$  has a traveling-salesman tour with cost at most  $k$ . The decision version of the problem has been shown to be NP-complete, meaning that there does



Optimal solution:  $A \xrightarrow{9} B \xrightarrow{3} C \xrightarrow{4} E \xrightarrow{8} D \xrightarrow{7} (A)$

Fig. 12. Graphical representation of the traveling-salesman problem with a corresponding optimal solution [77].

not exist a deterministic polynomial-time bounded algorithm to indicate whether or not a given graph has a minimal tour of size  $k$  [34]. Figure 12 presents an example instance and optimal solution of the traveling-salesman problem.

The initial population in this case corresponds to permutations of the cities; however, not every permutation represents a valid solution because some cities are not directly connected (e.g., cities A and E). One method of handling potentially invalid solutions is to assign an artificially large distance between cities that are not directly connected, thereby forcing the invalid solutions to be extremely sub-optimal.

The objective in the traveling-salesman problem is to *minimize* the total distance of each tour. A simple fitness function that can be employed is  $f(T) = d$ , where  $f$  returns the length of a given tour  $T$ .

Additional insight is needed into the crossover function because resultant chromosomes may be invalid permutations. For example, a crossover of two solutions after the third position in the strings (characters to exchange are underlined)

A D E B C  
A E C D B

will produce new strings

A E C B C  
A D E D B

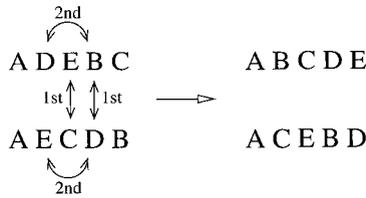


Fig. 13. Result of partially-matched crossover operation.

which are invalid solutions. To avoid this problem, a *partially-matched crossover* operation is used to directly operate on permutations. Given two solutions that are permutations of the same symbols, two random crossing points can be chosen within each chromosome (indicated by underlines).

$$\begin{array}{c}
 \text{A D } \underline{\text{E}} \underline{\text{B}} \text{ C} \\
 \text{A E } \underline{\text{C}} \underline{\text{D}} \text{ B}
 \end{array}$$

The crossover operation requires an exchange of symbols between chromosomes — E with C, and B with D — represented by ordered pairs (E, C) and (B, D) respectively. The next step is to permute each of the two-element permutations in each string. Within the first chromosome, for example, C and E are swapped. Likewise in the second chromosome, B and D are swapped. The chromosomes produced by the partially-matched crossover are shown in Fig. 13.

According to [77] the application of a GA in this manner outperforms a random search for optimal solutions to the traveling-salesman problem; however, the algorithm leaves much room for improvement. Typical results from the algorithm when applied to 100 randomly-generated cities gave a value of the whole tour 9.4% above minimum after 20,000 generations.

## 5. Current and Future Applications

Many published works can be found on the application of genetic algorithms to data mining tasks. As a starting point [48] provides a survey of GAs for data mining and knowledge discovery. This publication focuses on classification and addresses GA usage in a data mining context, including various types of data pre- and post-processing techniques. A book by the same author [49], is dedicated entirely to the application of GAs to data mining tasks; however, these topics are presented in much greater detail than the previously mentioned publication [36] addresses the specific issue

of mutation and presents other mutation operators and their respective performance in solving optimization problems.

Other publications of interest include applications of GAs to clustering, association rules, classification, and Bayesian networks [47] proposes a method of vector clustering with application to image compression. Both related to Web-based hypermedia systems, [130] and [107] present methods for applying GAs to association rule inference and optimized classification respectively. Also, [72] discusses a technique of using GAs to learn structures of Bayesian networks.

## 6. Summary

Genetic algorithms aim to find optimal solutions to large-scale optimization problems through a unique combination of stochastic and directed search techniques. When contrasted with other data mining methods, GAs offer the advantage of maintaining a population of potential solutions, whereas other knowledge discovery techniques process a single point of the search space. GAs also lend themselves well to parallelized search procedures, in turn decreasing the overall computation time for a given problem. GAs are applicable to both continuous- and discrete-based problems, and accommodate structure and parameter identification in complex models.

GAs also have drawbacks when attempting to put genetic algorithm theory into practice. First, the encoding of the problem often moves the GA to function in a different space than that of the problem. Second, there are practical limits on the theoretically unlimited number of generations produced by a GA. Finally, population size cannot always be assumed to be unbounded. Despite these disadvantages, genetic algorithms provide an intuitive and efficient means of finding optimal solutions to a wide variety of data mining problems.

## CHAPTER 6

### CLASSIFICATION: DISTANCE-BASED ALGORITHMS

Zeqian Shen

*Department of Computer Science, University of California, Davis  
One Shields Avenue, Savis, CA 95616, USA  
zqshen@ucdavis.edu*

#### Overview

Distance-based algorithms are nonparametric methods that can be used for classification. These algorithms classify objects by the dissimilarity between them as measured by distance functions. Several candidate distance functions are reviewed in this chapter along with two particular classification algorithms. Some of the current applications related to distance-based algorithms are also addressed.

**Keywords:** Classification, distance measures,  $K$ -nearest neighbors, Euclidean distance, city block distance, tangent distance.

#### 1. Introduction

Distance-based algorithms assume that an object is more similar to the objects within the same class as opposed to objects from other classes. Therefore, the classification of the target object is affected by the objects that are similar to it. The concept of distance is used to measure the dissimilarity between objects. In other words, two similar objects can be considered close to each other in the sample space. The two key issues in distance-based classification are choosing the proper distance function and the design of the classification algorithm. Many kinds of distance functions can be used, such as city block distance or Euclidean distance. Different distances have different characteristics, which fit various types of data. Classification algorithms must determine the class of target object according to objects *close*

to it. One of the most effective techniques is  $K$ -Nearest Neighbors (KNN). Using the  $K$ -closest objects, the target object is assigned the class that contains the most objects. KNN is widely used in text classification [162], web mining [92] and stream data mining [84].

## 2. Motivation

The classification methods mentioned in previous chapters, such as Bayes rule, are under the assumption that the underlying distribution of data is known. Nevertheless, this assumption is suspect in most practical applications. Distance-based algorithms are of interest because they are nonparametric classification methods that do not require the prior knowledge of the data and can work with arbitrary distributions.

Besides data classification, the idea of using distance to measure the dissimilarity is important. It is applied in many other areas, such as machine learning [151], query similarity [17] and outlier detection [86].

The remainder of this chapter is organized as follows: several distance functions are reviewed in Sec. 3. Two classification algorithms are addressed in Sec. 4. Current applications of distance-based algorithms are discussed in Sec. 5, and a summary of the advantages and disadvantages of the algorithms is provided in Sec. 6

## 3. Distance Functions

Distance-based algorithms rely on distance functions to measure the dissimilarity between the objects. Selecting a distance function is not only the first step of the algorithms, but also a critical step. Different distance functions have different characteristics, which fit various types of data. There does not exist a distance function that can deal with every type of data. So the performance of the algorithm heavily depends on whether a proper distance function is chosen for that particular data. For a set  $X$ , the distance function  $d: X \times X \rightarrow \mathcal{R}$ , for all  $x, y, z \in X$ , satisfies

$$\begin{aligned} d(x, y) &\geq 0, \\ d(x, y) &= 0 \text{ if and only if } x = y, \\ d(x, y) &= d(y, x) \text{ (symmetry law), and} \\ d(x, z) &\leq d(x, y) + d(y, z) \text{ (triangle inequality).} \end{aligned}$$

Interestingly, several distance functions used in practice do not necessarily satisfy all four of the constraints listed above. For example, the squared Euclidean distance does not satisfy the triangle inequality and the

Kullback–Leibler distance function used in document clustering is not symmetric [14, 37]. A good distance function should be invariant to the natural data transformations that do not affect the class of the objects. This criteria will be explained and discussed in detail in the following review of distance functions.

### 3.1. City Block Distance

*City block distance*, sometimes called *Manhattan distance* is defined as

Let  $x, y \in X$ , where  $x = \{x_1, x_2, \dots, x_k\}$  and  $y = \{y_1, y_2, \dots, y_k\}$ .

Then,  $d_{CityBlock}(x, y) = \sum_{i=1}^k |x_i - y_i|$ .

This measure reflects the sum of the absolute distances along each coordinate axis. In Fig. 14, the city block distance between  $P_1$  and  $P_2$  is given by

$$d(P_1, P_2) = |1 - 5| + |3 - 1| = 6.$$

Although the city block distance is easy to compute, it is variant to scaling, rotation and many other transformations. In other words, the similarity is not preserved by the city block distance after these transformations. Such a distance measure would not be appropriate for many types of data (e.g., images) which may be invariant to rotation and scaling.

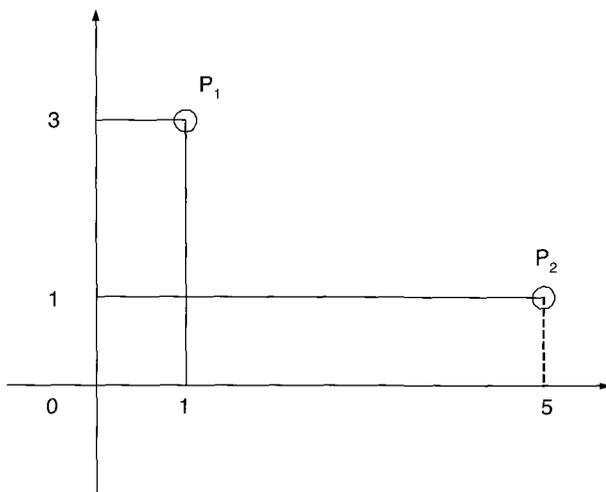


Fig. 14. City block distance between two points in 2D space.

### 3.2. Euclidean Distance

Euclidean distance is the most common distance used as the dissimilarity measure. It is defined as

$$d_{Euclidean}(x, y) = \left( \sum_{i=1}^k |x_i - y_i|^2 \right)^{1/2} .$$

Figure 15 illustrates the effects of rotation and scaling on Euclidean distance in a 2D space. It is obvious from Fig. 15 that dissimilarity is preserved after rotation. But after scaling the  $x$ -axis, the dissimilarity between objects is changed. So Euclidean distance is invariant to rotation, but not to scaling. If rotation is the only acceptable operation for an image database, Euclidean distance would be a good choice.

### 3.3. Tangent Distance

In order to achieve invariance of different transformations, the *tangent distance* is often used. Let  $x \in X$  be an object and  $T(x, \alpha)$  denote a

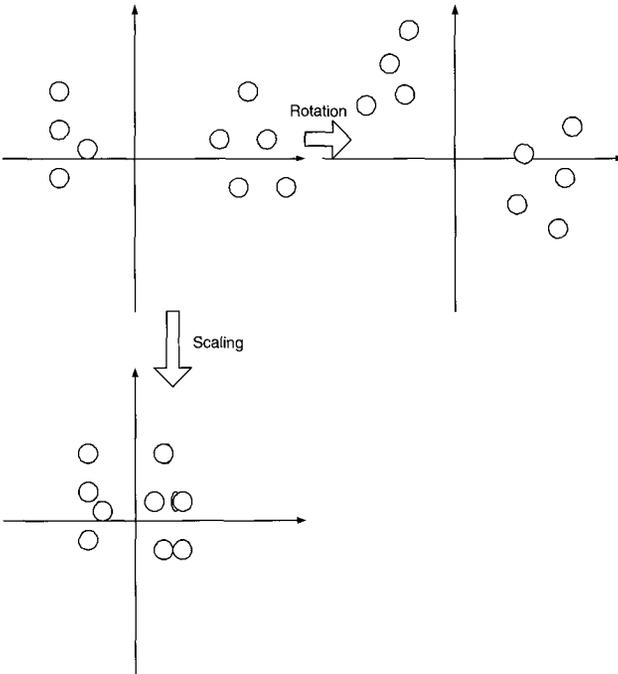


Fig. 15. Effects of rotation and scaling on Euclidean distance.

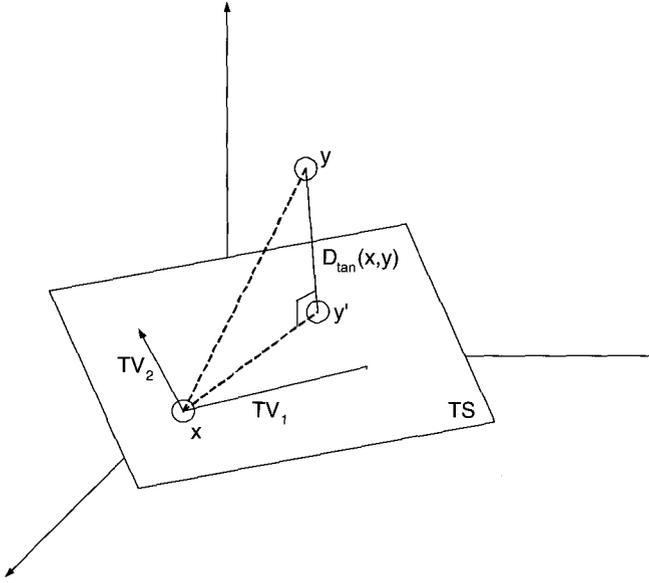


Fig. 16. Tangent distance in 3D sample space.

transformation of  $x$  by  $\alpha$  that is natural to the data. For example,  $T(x, \alpha)$  could represent the image of rotating  $x$  by an angle  $\alpha$ . A *tangent vector* is defined as  $TV = T(x, \alpha) - x$ . Suppose the data has  $n$  natural transformations  $T_i(x, \alpha)$ ,  $i = 1, 2, \dots, n$ , and the *tangent space*  $TS$  is the space spanned by all the  $n$  tangent vectors. The tangent distance between objects  $x$  and  $y$  is the distance from  $y$  to tangent space  $TS$ . In Fig. 16,  $y'$  is the projection of  $y$  on to the tangent space  $TS$ . The tangent distance is calculated as

$$d_{tan}(x, y) = \|y - y'\|.$$

Because all the transformations of  $x$  are in the same tangent space  $TS$ , they have the same tangent distance to  $y$ . Therefore, the tangent distance is an invariance distance [138]. Because of this invariance, the tangent distance may be used to fit data from diverse sources: speech [103], images [131], and handwriting [94].

### 3.4. Other Distances

There are also many other distances that can be used for different data. Edit distance fits sequence and text data [128]. The Tanimoto distance is suitable for data with binary-valued features [41].

Actually, data normalization is one way to overcome the limitation of the distance functions. For example, normalizing the data to the same scale can overcome the scaling problem of Euclidean distance. However, normalization may lead to information loss and lower classification accuracy.

#### 4. Classification Algorithms

After a proper distance function is chosen, another key issue is the method to determine the class of the object. There are several different algorithms based on the class of objects close to the target. One simple approach assigns the closest class to the target object. The distance between a class and an object is defined as the distance between the representative vector of the class and the object. Another approach is the KNN algorithm (or technique) mentioned earlier. This algorithm classifies the target object by assigning it the class that is most frequently represented among the  $K$ -nearest objects. In other words, the classification is made by checking the classes of the  $K$ -closest neighbors and taking a majority vote. For simplicity, both algorithms are discussed using Euclidean distance.

##### 4.1. A Simple Approach Using Mean Vector

The most straightforward method assigns the object to its most similar class. The classification problem [42] is formulated as follows. Let database  $D = \{x_1, x_2, \dots, x_n\}$ , where each object  $x_i = \{x_{i1}, x_{i2}, \dots, x_{ik}\}$  contains numeric values. Name a set of classes  $C = \{C_1, C_2, \dots, C_m\}$  where each class  $C_j = \{C_{j1}, C_{j2}, \dots, C_{jk}\}$  has numeric values. The classification problem is to assign each  $x_i$  to the class  $C_j$  such that  $d(x_i, C_j) \leq d(x_i, C_l) \forall C_l \in C$ , where  $C_l \neq C_j$ .

The representative vector  $\{C_{j1}, C_{j2}, \dots, C_{jk}\}$  of class  $C_j$  is the key component of this method. One way to define this vector is to use the mean vector of objects in class  $C_j$ , i.e.,

$$C_{jt} = \text{mean}(x_{it}), \text{ where } x_i \text{ is labeled as class } C_j \text{ and } t = 1, 2, \dots, k.$$

This classification algorithm can be described as follows.

**Example 4:** Figure 17 illustrates how Algorithm 4 can be used to classify an unknown object in 2D space [42]. The representative vectors of class  $A$ ,  $B$ ,  $C$  are  $[4, 7.5]$ ,  $[2, 2.5]$  and  $[6, 2.5]$  respectively. The input object is

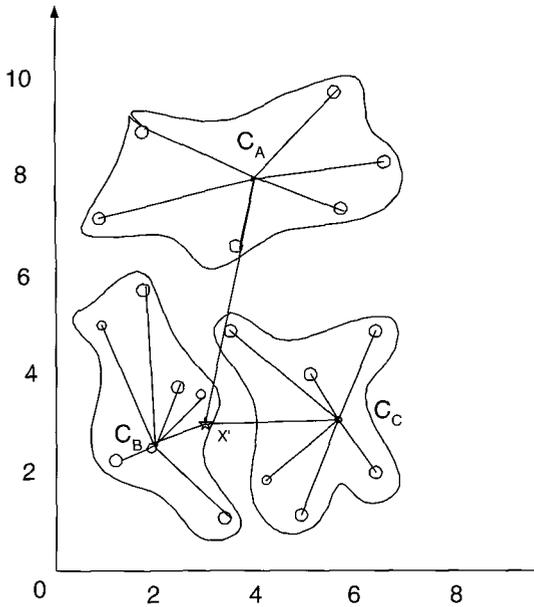


Fig. 17. A simple approach of distance-based algorithm.

---

**Algorithm 4** Simple Distance-based algorithm (using mean vector)
 

---

*Input :*

$T$  {Training set}

$x'$  {Input object to classify}

*Output :*

$c$  {Class to which  $x'$  is assigned}

$C_{jt} \leftarrow \text{mean}(x_{it})$ , where  $x_{it} \in T$  and is labeled class  $C_{jt}$  {Calculate the mean vector of each class}

$dist \leftarrow \infty$ ;

**for**  $i := 1$  to  $m$  **do**

**if**  $d(C_i, x') < dist$

$c \leftarrow i$

$dist \leftarrow d(C_i, x')$

**end if**

**end for**

---

$x'$  with the values  $[3, 3]$ . The distance between  $x'$  and three classes are  $d(x', C_A) = 4.61$ ,  $d(x', C_B) = 1.12$ , and  $d(x', C_C) = 3.04$ . So  $x'$  is assigned to class  $C$ . This algorithm assumes that the mean vector is the unbiased representative of the class. This assumption can be dubious in practical applications since the training set may not be sufficiently large to represent all characteristics of the classes.

#### 4.2. $K$ -Nearest Neighbors

KNN (Algorithm 5) is a more complicated and robust classification algorithm that labels the target based on the classes of the  $K$ -nearest objects. The KNN algorithm [42] is illustrated by the following example.

---

#### Algorithm 5 KNN Algorithm

---

*Input :*

$T$  {Training Data}

$K$  {Number for classify}

$x'$  {Input object to classify}

*Output :*

$c$  {Class to which  $x'$  is assigned}

$N \leftarrow 0$

**for all**  $v \in T$  **do**

**if**  $|N| \leq K$  **then**

$N \leftarrow N \cup \{v\}$

**else if**  $\exists u \in N$  **such that**  $d(x', u) \geq d(x', v)$  **then**

$N \leftarrow N - \{u\}$

$N \leftarrow N \cup \{v\}$

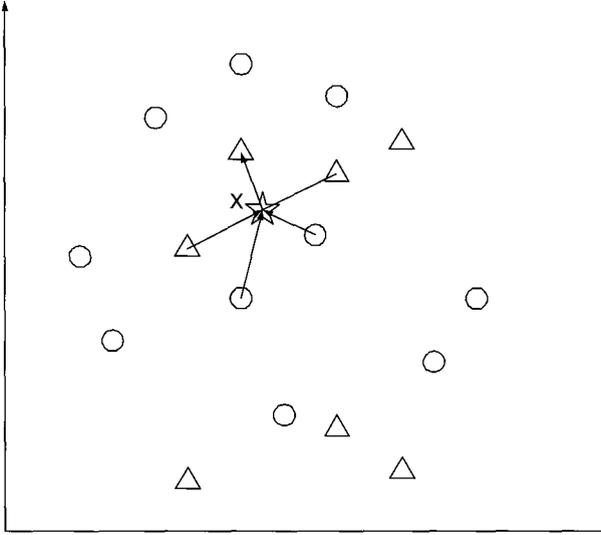
**end if**

**end for**

$c =$  class to which the most  $u \in N$  are classified.

---

**Example 5:** Figure 18 illustrates an example of KNN in 2D space. In Fig. 18, the star  $x'$  is the target object. The triangles indicate the objects belong to class  $A$  and the circles indicate those belonging to class  $B$ . Among the five nearest objects of  $x'$ , there are three triangles and two circles. So  $x'$  is assigned to class  $A$ .

Fig. 18.  $K$ -nearest neighbors ( $k = 5$ ).

KNN is a nonparametric classification method that is based on the estimation of *posteriori* probabilities. According to the Bayes Theorem (see Chap. 2), the probability of object  $x_i$  belonging to class  $C_j$  is

$$P(C_j|x_i) = \frac{p(x_i|C_j) \times P(C_j)}{p(x_i)}.$$

The classification is based on the objects inside the region surrounding the target object. Define the region as  $R$  and let  $V$  be the volume enclosed by  $R$ . Given a training set with the size  $n$ , let the probability of an object belong to class  $C_i$  within region  $R$  be  $P_n(C_i)$ . It is given by [41]

$$P_n(C_i) = \int_R p_n(x', C_i) d(x'). \quad (29)$$

If  $p_n(x', C_i)$  is continuous and the region  $R$  is so small that  $p_n(x', C_i)$  does not vary appreciably within it,  $P_n(C_i)$  can be approximated by

$$\int_R p_n(x', C_i) d(x') \cong p_n(x', C_i) \times V, \quad (30)$$

or

$$P_n(C_i) \cong p_n(x', C_i) \times V. \quad (31)$$

Suppose all the  $n$  objects in a training set are drawn independently and identically distributed (i.i.d.). Let  $k_i$  be the number of objects that fall in

$R$  and belong to class  $C_i$  and  $\sum k_i = K$ . Then,

$$P_n(C_i) = k_i/n. \quad (32)$$

Combining Eqs. (31) and (32), yields

$$p_n(x', C_i) * V = k_i/n,$$

or

$$p_n(x', C_i) = \frac{k_i/n}{V}. \quad (33)$$

If the training set is large enough, i.e.,  $n$  approaches  $\infty$ ,  $p_n(x', C_i)$  can be taken as an approximation of  $p(x', C_i)$ . By Eq. (33),

$$P(C_i|x') = \frac{p(x'|C_i) \times P(C_i)}{p(x')} = \frac{p(x', C_i)}{\sum_{i=1} p(x', C_i)} \quad (34)$$

$$\cong \frac{p_n(x', C_i)}{\sum_{i=1} p_n(x', C_i)} = \frac{k_i}{\sum k_i} = \frac{k_i}{K}. \quad (35)$$

According to the Bayes Theorem, the class with the maximum *posteriori* probability  $P(C_i|x')$  will be assigned to the object. Therefore, the class with the largest  $k_i$ , which is the number of objects belong to class  $C_i$  in the set of  $K$ -closest objects, will be assigned to the target object  $x'$  (the expected result for the KNN algorithm). The key to optimal performance of the KNN algorithm is the value of  $K$ . The rule that  $K \leq \sqrt{n}$  is suggested in [83].

KNN is a typical lazy classification method that does not build a classifier until a new object needs to be classified. During classification, the distances between the new object and each object in the training set are computed. So the time complexity is  $O(n)$ . The time expense is high compared to eager classification methods, such as those decision tree based methods which build a search structure before examining new objects. Some research has been done to improve the performance of lazy classifiers, especially KNN [123].

## 5. Current Applications

Distance-based classification algorithms, especially KNN, are widely used in practical applications. KNN has been applied to text categorization since the early stages of research and is one of the top-performing methods [162]. In [92], KNN is supplemented with a feature selection method and a term-weighting scheme to achieve a good performance of web page classification. KNN has been adapted into a data-mining-ready structure in the

stream data mining project [84] and yields higher classification accuracy as well as higher speed. Distance-based algorithms have also been applied in other contexts. For example, decision tree approaches based on the distance between objects have been used for similarity searches in high-dimensional generic databases [30, 164].

## 6. Summary

Distance-based algorithms classify objects according to the objects close to them. So the first step is to choose a proper distance function for the particular data. Euclidean distance and city block distance are easy to calculate. However, they are variant to many transformations. Tangent distance is an invariance distance and can fit many types of data, such as images and speech.

The second step of a distance-based algorithm is to classify the objects. *K*-Nearest Neighbors is a nonparametric method that is often used in practical data mining applications. It is easy to implement and yields good performance. However, KNN does not build the classifier until a new object needs to be classified. If some classifier structures can be built before new tasks are received, the performance of classification would be even better.

This page is intentionally left blank

## CHAPTER 7

### DECISION TREE-BASED ALGORITHMS

Farial Shahnaz

*Information International Associates, Inc.*  
*P.O. Box 16221, Knoxville, TN 37996, USA*  
*shahnaz@cs.utk.edu*

#### Overview

Decision tree-based algorithms serve as the fundamental step in application of the decision tree method, which is a predictive modeling technique for classification of data. This chapter provides a broad overview of decision tree-based algorithms that are among the most commonly used methods for constructing classifiers.

**Keywords:** Boosting, attribute, data mining, decision trees, domain, entropy, information gain, pruning.

#### 1. Introduction

Decision trees (DTs) are widely used in data mining for classification purposes. Given a heterogeneous data collection and a set of attributes that describe the data, decision trees use the values of these attributes to divide the data set into smaller, more homogeneous subsets. Implementation of decision trees differ primarily in how the trees are constructed and several techniques exist for constructing or modeling the trees. These modeling techniques are commonly referred to as decision tree-based algorithms, or simply, decision tree algorithms. The objective of all DT algorithms is to minimize the size of the tree while maximizing the accuracy of the classification.

## 2. Motivation

Decision tree-based algorithms greatly affect the performance of DT approaches to data mining. Without the use of an efficient DT algorithm, a decision tree might contain several nodes that have little or no relevant information for classifying data. The presence of nodes containing insignificant attributes results in increased depth, which detracts from the effectiveness of DTs. Moreover, DT algorithms not only choose the best splitting attribute for a node, but also decide what values or how many branches to assign to that node. Poorly designed decision tree algorithms that assign random values often cause an ineffective rendering of the DT technique.

This chapter will discuss the three most common algorithms used for DT construction: ID3, C4.5, C5.0, and CART. Each algorithm will be discussed with respect to the success of these algorithms in minimizing cost while maximizing information gain.

## 3. ID3

Interactive Dichotomizer 3 or ID3, uses a basic tree induction algorithm that assigns an attribute to a tree node based on how much information is gained from that node. The ID3 method allows an attribute to have two or more values at a node or splitting point, thereby facilitating formation of an  $n$ -ary tree where  $n$  can be greater than two. Table 6 contains the input and output parameters for the BuildDT algorithm — the basic tree induction method used in ID3.

Table 6. The input and output parameters for algorithm BuildDT.

---

### Input:

Data set  $D = \{d_1, d_2, \dots, d_n\}$ .

Collection of attributes  $A = \{a_1, a_2, \dots, a_m\}$ .

Domain of each attribute  $a_i$ ,  $V_i = \{v_{i1}, v_{i2}, \dots, v_{ik}\}$ .

Each  $v_{ij}$  represents a value of the attribute  $a_i$ .

$k$  is the cardinality of the attribute domain  $V_i$ .

### Output:

Decision tree  $T = \{t_1, t_2, \dots, t_s\} \cup \{c_1, c_2, \dots, c_r\}$ .

Each  $t_i$  represents an internal node, i.e., an attribute.

$s$  is the number of internal nodes.

Each  $c_i$  represents an external or leaf node, i.e., a class or category.

$r$  is the number of external nodes.

---

---

**Algorithm 6** BuildDT Algorithm

---

```

Select best splitting criterion  $a_i$  from set A
Create root node  $t_i$  and assign it label  $a_i$ 
Add a branch to node  $t_i$  for each  $v_{ij}$  in  $V_i$ 
Add  $t_i$  to set  $T$ 
for each branch of node  $t_i$ 
{
    Set  $D =$  subset of  $D$  created by applying attribute  $a_i$  to  $D$ 
    if stopping point reached for this path, then
    {
        Create leaf node  $c_i$ 
        Add  $c_i$  to set  $T$ 
    }
    else
{
     $T' = \text{BuildDT}(D)$ 
    Add  $T'$  to set  $T$ 
}
}

```

---

BuildDT (Algorithm 6) assumes knowledge of optimal attribute selection and accurate stopping point detection. Ideally, a stopping point is reached when the data set is perfectly classified; however, situations may arise where reaching the accurate stopping point becomes an obstacle to creating a tractable decision tree. In such cases, a trade-off is often made between accuracy and performance by setting a stopping criterion at a certain depth.

Selecting the splitting criterion or optimal attribute at each node involves choosing the attribute that maximizes information gain. In other words, the optimal attribute minimizes the information needed in the resulting subtree to classify the data set [77]. Information gain is also referred to as *entropy reduction*, where entropy denotes the measure of randomness in a data collection. After selecting the lowest entropy attribute, branches representing the attribute values or choices are added to the node. Since the number of branches depends on the cardinality of the attribute domain, domains with a high cardinality can adversely affect the performance of a DT. Much like the ideal stopping point quandary, accuracy is often compromised by limiting the available choices to enhance performance.

Two major issues that remain beyond the scope of ID3 are processing continuous data and handling incomplete data sets. Nevertheless, considerable progress has been made with regard to these issues by ID3's successor algorithms C4.5 and C5.0, which are discussed in the following sections.

#### 4. C4.5

The C4.5 algorithm utilizes the same basic inductive tree creation approach as ID3, but extends its capabilities to classification of continuous data by grouping together discrete values of an attribute into subsets or ranges. Another advantage of C4.5 is that it can predict values for data with missing attributes based on knowledge of the relevant domains [42]. C4.5 also provides a way to prune or reduce the size of the tree without any significant decrease in accuracy. Pruning occurs in two forms [42]: *subtree replacement* and *subtree raising*. In case of the former, a subtree is replaced with a leaf node, and in the second method, a subtree is replaced with its most frequently used subtree.

In both cases, replacement is acceptable only when the original tree undergoes minimal distortion as a result of pruning. In situations where tree pruning does not effectively reduce the complexity of the DT structure, C4.5 generates decision rules based on the choices associated with a path, which is defined as a set of branches connecting two nodes. The following example, with the aid of Fig. 19, illustrates the generation process of decision rules. It should be noted that the complexity of this example is not indicative of the actual complexity of typical classification problems.

The decision Rules for tree  $T$  in Fig. 19 are shown below:

Rule 1: If ( $A = x_1$  and  $B = y_1$ ), then Classification = Class 1;

Rule 2: If ( $A = x_2$  and  $C = z_1$ ), then Classification = Class 2;

Rule 3: If ( $A = x_2$  and  $C = z_2$ ), then Classification = Class 1.

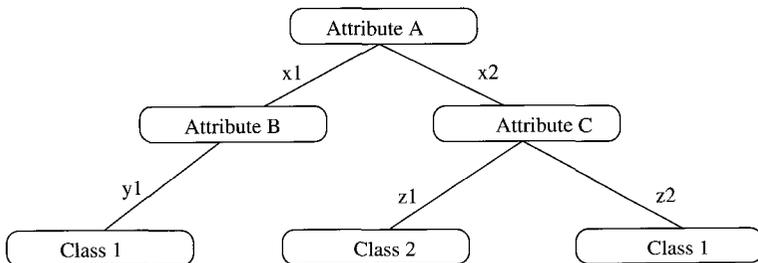


Fig. 19. A simple decision tree  $T$ .

In this set of decision rules, Rule 1, 2, and 3 are respectively generated by the paths between nodes A and Class 1, A and Class 2, and A and Class 1. Multiple decision rules can be merged to further reduce the complexity of a tree. For example, grouping multiple rules together results in a smaller set of decision rules for  $T$ :

If ( $A = x_1$  and  $B = y_1$ ) or ( $A = x_2$  and  $C = z_2$ ),  
then Classification = Class 1;

If ( $A = x_2$  and  $C = z_1$ ), then Classification = Class 2.

It is worth noting that merging should decrease the entropy of the data set; if it does not, then it should be avoided. This feature of rewriting multiple decision rules in a conjunctive form is unique to C4.5 and is not available in its commercial version C5.0.

## 5. C5.0

Algorithm C5.0, known as See5 on the MS Windows platform, provides significant improvements on memory usage and runtime. It also produces better results than C4.5 in terms of minimizing decision trees and generating decision rules. The precise algorithm for C5.0 has not been made public, but one of the chief contributors to the performance augmentations achieved by C5.0 is a new feature called *boosting*. Boosting is a method for creating multiple training sets from the original data set by assigning weights to each record in the data collection [42]. The example below demonstrates the application of boosting to a contrived data set  $D$ .

Data set  $D = \{d_1, d_2, \dots, d_5\}$

Set of attributes,  $A = \{\text{shape}, \text{color}\}$

Domain of attribute 'shape' = {square, round}

Domain of attribute 'color' = {red (r), green (g), blue (b)}

record  $d_1 = \{\text{square}, r\}$

record  $d_2 = \{\text{square}, g\}$

record  $d_3 = \{\text{square}, b\}$

record  $d_4 = \{\text{round}, r\}$

record  $d_5 = \{\text{round}, g\}$ .

A straightforward approach to construction produces the classifier  $T$  (Fig. 20).

Applying boosting to  $D$  produces  $k$  sets, where  $k$  is the number of training sets desired by the user. These training sets can be written as  $D_i = \{w_{i1} * d_1, w_{i2} * d_2, w_{i3} * d_3, w_{i4} * d_4, w_{i5} * d_5\}$ , where "\*" denotes association,  $i = \{1, \dots, k\}$  enumerates the new data sets,  $w_{ij}$  is the weight

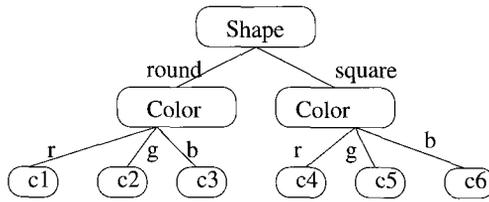


Fig. 20. A simple decision tree  $T$  with six classes.

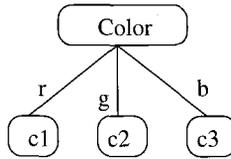


Fig. 21. Decision tree  $T_i$  for data set  $D_i$ .

associated with record  $d_j$  of training set  $D_i$ , and  $w_{ij}$  denotes the importance of record  $d_j$  to the classification for  $D_i$  [42].

If all the weights are equal,  $D_i$  is equivalent to the original set  $D$ . Otherwise, the resulting classifier may be significantly different from  $T$  (Fig. 20). Suppose the set of weights associated with some data set  $D_i$  is  $\{0.5, 0.5, 0.5, 0, 0\}$ . This weight distribution implies records 4 and 5 are not important for classifying  $D_i$ . Since records 1, 2, and 3 all have the same value for “shape,” it can be inferred that the “shape” attribute has negligible impact on the construction of a classifier for  $D_i$  (Fig. 20). By applying different combinations of weights to the original data set, many such classifiers (Fig. 21) can be constructed.

Algorithm C5.0 provides an efficient mechanism for emulating this process of boosting for large data sets. During classification of a new record, C5.0 assigns a vote to each of the decision trees created through boosting. Then voting is performed, and the new record is assigned to the class with the most votes [42]. The use of multiple classifiers is the key to increasing the accuracy of C5.0.

## 6. CART

The Classification and Regression Trees (CART) algorithm adheres to the same basic tree induction algorithm used by C5.0 and its predecessors for selecting optimal attributes; unlike the previously discussed algorithms, CART only allows the construction of binary trees. Given a data set, the

CART method performs *binary recursive partitioning* based on attributes that can be expressed as questions with “yes” or “no” answers [96]. At each node, all possible splits are compared and the attribute with the highest degree of homogeneity is selected. CART also includes a pruning technique quite similar to the pruning features available in C4.5 and C5.0 — pruning is performed only if the reduction in complexity is not offset by a considerable increase in entropy or inaccuracy.

The CART algorithm is amenable to processing continuous data sets. Moreover, it is capable of overcoming the problem imposed by incomplete data sets. For any record in the data set that lacks a value for a splitting attribute, CART assigns a surrogate value to it [96] rather than discarding the entire record or data point. As shown below, a surrogate value is derived by examining other records in the data set and choosing a data point most similar to the one with a missing attribute.

Suppose data set  $D = \{d_1, d_2, d_3, d_4\}$  so that each  $d_i$  is a data point or record containing three numerical attributes  $\{X, Y, Z\}$ . Sample data records are listed below:

record  $d_1 = \{1, 2, 3\}$   
record  $d_2 = \{1, 4, 2\}$   
record  $d_3 = \{1, -, 3\}$   
record  $d_4 = \{2, 4, 3\}$ .

If  $Y$  is the optimal attribute at any node and record  $d_3$  is being considered for a split, scrutiny of the data set would yield 2 as the surrogate value since record  $d_1$  is the most similar to  $d_3$ .

Notwithstanding its array of features, the most appealing aspect of the CART method still remains its binary format. This approach is largely responsible for the acceptance of decision trees as a tool for predictive modeling by the statistics community.

## 7. Summary

The DT algorithms described in this chapter represent the most effective methods available for constructing decision trees. For all decision tree-based algorithms, the challenge lies in maintaining the accuracy of a classifier without paying a high price for computational complexity. Many algorithms espouse techniques such as pruning or setting a stopping point at a certain depth to prevent the creation of computationally intractable trees. Although strides made in this direction by algorithms such as C5.0 deserve considerable acclaim, there still remains much room for improvement.

This page is intentionally left blank

## CHAPTER 8

### COVERING (RULE-BASED) ALGORITHMS

Shakhina Pulatova

*National Instruments*

*11500 N. Mopac Expressway, Austin, TX 78759-3504, USA*

*shakhina@gmail.com*

#### Overview

Classification rules (if-then rules) are an intuitive way of performing classification, and they are a popular alternative to decision trees. Covering algorithms are one of the most well-studied methods for inducing classification rules from training sets. This chapter discusses basic covering algorithms and illustrates them with simple examples that capture the essence of the problems. In addition, some of the current applications of covering algorithms are reviewed.

**Keywords:** Covering algorithms, 1R, PRISM, classification rules, rule-based.

#### 1. Introduction

Classification is one of the most developed and applied techniques in data mining. Given a collection of records each containing a set of attributes, classification is a model that accurately assigns records to specific classes depending on the values of their attributes. Classification methods developed in data mining are widely used in image and pattern recognition, medical diagnosis, homeland security, and loan approval.

One direct way of performing classification is generating sets of if-then rules, where the data is assigned to some class if certain conditions are met. Covering (rule-based) algorithms provide mechanisms that generate compact, easy-to-interpret, and accurate rules by concentrating on a specific

class at a time and maximizing the probability of the desired classification [156]. Simple covering algorithms can be utilized to gain insight into the data and to obtain baseline performance while more sophisticated covering algorithms can be employed to tackle challenging classification tasks.

## 2. Motivation

One way of generating classification rules is to traverse existing decision trees. The disadvantage of this approach is that rules generated this way may be unnecessarily complex and incomprehensible. Covering algorithms, however, produce more compact, easy-to-understand and easy-to-modify rules with comparable accuracy. Covering algorithms are thus widely used for classification in data mining and machine learning.

This chapter introduces the concept of classification rules in Sec. 3 and briefly discusses the advantages and disadvantages of directly deriving the rules from decision trees as opposed to constructing them by other means. Basic covering algorithms are introduced in Sec. 4 and are explained with simple examples. A review of some of the applications of covering algorithms is provided in Sec. 5, followed by a summary in Sec. 6.

## 3. Classification Rules

As illustrated in Chaps. 4 and 7, the decision tree is a widely used technique for classification purposes. Another popular alternative to decision trees is classification rules (also known as if-then rules). A classification rule is defined as  $r = (a, c)$ , where  $a$  (antecedent/precondition) is a series of tests that can be evaluated as true or false, and  $c$  (consequent/conclusion) is the class or classes that apply to instances covered by rule  $r$  [42, 156]. For example, one could have the following set of rules to classify the weather:

If temperature  $< 50^\circ\text{F}$ , then weather = cold.

If temperature  $\geq 50^\circ\text{F}$  AND temperature  $\leq 80^\circ\text{F}$ , then weather = warm.

If temperature  $> 80^\circ\text{F}$ , then weather = hot.

Although any of the logical expressions are allowed, preconditions are usually connected with the AND operation. One straightforward way of constructing classification rules is to derive them from an existing decision tree, where each rule corresponds to a different path from the root of the tree to its leaf. Figure 22 provides a simple example of building classification rules based on a decision tree.

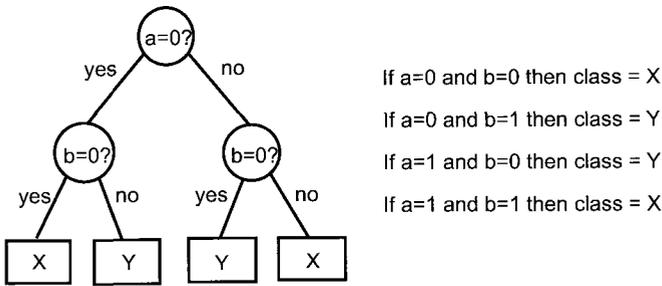


Fig. 22. Rules constructed from a decision tree.

One advantage of this method is that rules are order-independent, that is, regardless of the order of rules executed, the same classification of the classes is reached [156]. The disadvantage, however, is that the generated rules are often more complex than necessary and contain redundant information. The reason usually lies in decision trees' inability to "express the disjunction implied between the different rules in a set," leading to a *repliated subtree problem*, as illustrated by the following example [156]. Consider the rules:

If  $a$  AND  $b$ , then  $x$ .  
 If  $c$  AND  $d$ , then  $x$ .

Since a single test condition must be chosen for a root node, the resulting decision tree for these simple rules becomes far more complicated and incomprehensible (see Fig. 23). Notice that the tree in the figure contains duplicate subtrees rooted at node  $c$ . Moreover, the addition of an extra test or rule to a set requires reshaping of the whole tree.

In contrast, rules obtained without decision trees are usually easier for humans to understand and modify. They are more compact and are useful for providing insights about regularities in the data. On the other hand, some algorithms generate rules that are intended to be followed in order (called *decision lists*) and can lead to incorrect classifications if the correct ordering is not met [156]. In addition, more than one classification can be obtained for the same instance, or the rule can fail to classify completely. Different strategies can be employed in this situation ranging from selecting the most frequently used class to reaching no conclusion at all. These different decisions have led to a vast number of rule-based algorithms available today.

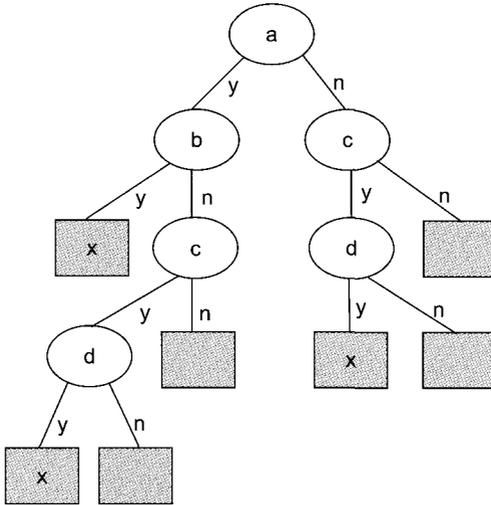


Fig. 23. Demonstration of a replicated subtree problem [156].

#### 4. Covering (Rule-based) Algorithms

One of the most well-studied methods for producing sets of classification rules from examples is covering algorithms [71]. They attempt to cover all instances of each class while excluding instances not in the class. The main point is that covering algorithms work on a specific class at a time, ignoring the rest of the classes [156]. For instance, if a rule is desired to classify the weather as warm in the previous example, then the covering algorithm attempts to replace  $x$  in the statement

If  $x$ , then class = warm,

with the condition that produces the best probability (in the training set) for the weather to be warm. Covering algorithms follow these three steps [71]:

- (i) Generate rule  $R$  on training data  $S$ ;
- (ii) Remove the training data covered by rule  $R$ ;
- (iii) Repeat the process.

This method can be visualized in the 2D space of instances illustrated in Fig. 24 (adapted from [156]). First, a rule is constructed to cover  $a$ 's by splitting the space vertically at  $x = 1.2$  and then further splitting it

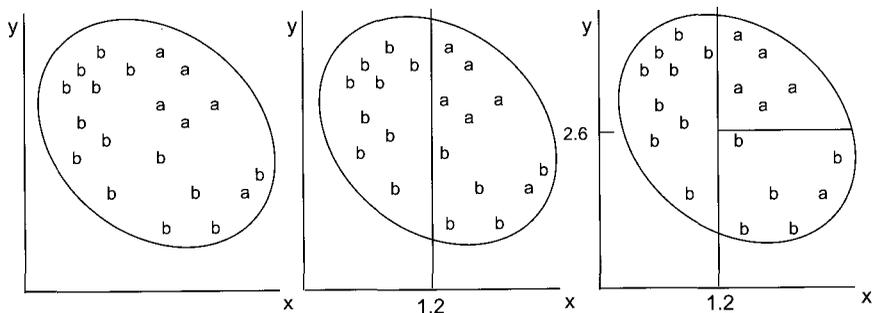


Fig. 24. Covering algorithm demonstration. Figure adapted from [156].

horizontally at  $y = 2.6$ , leading to the rule

If  $x > 1.2$  AND  $y > 2.6$ , then class =  $a$ .

Second, the following procedure is used to construct rules to cover  $b$ 's:

If  $x \leq 1.2$ , then class =  $b$ .

If  $x > 1.2$  AND  $y \leq 2.6$ , then class =  $b$ .

Note that one  $a$  is incorrectly covered by these rules, and more tests can be added to exclude that  $a$  from  $b$ 's cover and include it in the  $a$ 's cover.

#### 4.1. 1R Algorithm

One of the simple approaches used to find classification rules is called 1R, as it generates a one level decision tree. This algorithm examines the “rules that classify an object on the basis of a single attribute” [70]. A sample algorithm is provided by [42] in Algorithm 7.

The basic idea is that rules are constructed to test a single attribute and branch for every value of that attribute. For each branch, the class with the best classification is the one occurring most often in the training data. The error rate of the rules is then determined by counting the number of instances that do not have the majority class in the training data. Finally, the error rate for each attribute's rule set is evaluated, and the rule set with the minimum error rate is chosen.

A comprehensive comparative evaluation of the performance of 1R and other methods on 16 datasets (many of which were most commonly used in machine learning research) was performed by [70]. Despite its simplicity, 1R produced surprisingly accurate rules, just a few percentage points

---

**Algorithm 7** 1R Algorithm
 

---

**Input:**

$D$  //Training Data  
 $T$  //Attributes to consider for rules  
 $C$  //Classes

**Output:**

$R$  //Rules

**Algorithm:**

//1R algorithm generates rules based on one attribute  
 $R = \emptyset$ ;  
**for all**  $A$  in  $T$  **do**  
    $R_A = \emptyset$ ;  
   **for all** possible value,  $v$ , of  $A$  **do**  
     // $v$  may be a range rather than a specific value  
     **for all**  $C_j \in C$  **do**  
       //count is the number of occurrences of this class for this attribute  
       find  $count(C_j)$   
     **end for**  
     let  $C_m$  be the class with the largest count;  
      $R_A = R_A \cup ((A = v) \rightarrow (class = C_m))$ ;  
   **end for**  
    $ERR_A =$  number of tuples incorrectly classified by  $R_A$ ;  
   **end for**  
 $R = R_A$  where  $ERR_A$  is minimum;

---

lower in accuracy than the decision trees produced by the state of the art algorithm (C4). The decision trees produced by C4 were in most cases considerably larger than 1R's rules, and the rules generated by 1R were much easier to interpret. 1R therefore provides a baseline performance using a rudimentary technique to be used before progressing to more sophisticated algorithms.

**Example 6:** Consider the training data in Table 7 for deciding whether to play sports depending on four attributes: outlook, temperature, humidity and wind [156]. Table 8 shows the number of errors for each rule as well as the total number of errors for the rule set as a whole.

Table 7. The weather data [156].

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Table 8. Evaluating the attributes in the weather data [42].  
(\*) — random choice between two equally likely outcomes.

	Attribute	Rules	Errors	Total Errors
1	outlook	sunny $\rightarrow$ no	2/5	4/14
		overcast $\rightarrow$ yes	0/4	
		rainy $\rightarrow$ yes	2/5	
2	temperature	hot $\rightarrow$ no*	2/4	5/14
		mild $\rightarrow$ yes	2/6	
		cool $\rightarrow$ yes	1/4	
3	humidity	high $\rightarrow$ no	3/7	4/14
		normal $\rightarrow$ yes	1/7	
4	windy	false $\rightarrow$ yes	2/8	5/14
		true $\rightarrow$ no*	3/6	

The attributes with the smallest number of errors in the dataset are outlook and humidity. Assuming the algorithm arbitrarily chooses humidity, the resulting classification rules are:

Humidity: High  $\rightarrow$  no  
 Normal  $\rightarrow$  yes.

## 4.2. PRISM Algorithm

The inherent redundancy of decision trees requires the outcomes of irrelevant tests before a decision can be made. These tests may be very costly to perform, for instance, in medicine: irrelevant tests performed for diagnosing a patient waste time and even may require unnecessary surgery. Even though it is based on ID3, the PRISM algorithm [42] given in Algorithm 8 overcomes these and other decision tree problems [24]. PRISM identifies a rule that covers many instances in the class, separates out the covered instances, and continues the process with the rest. The algorithm induces modular rules with 100 percent accuracy and produces rules with comparable accuracy to the decision trees induced by ID3. It measures the success of a rule by the formula:  $p/t$ , where  $t$  is a total number of instances, and  $p$  is number of those that are positive.

**Example 7:** Consider an example of classifying height (here the classification to Tall is considered). Given the training set in Table 9, the probabilities that the Tall class has been assigned given attribute-value pairs are [42]:

Gender = F	0/9,
Gender = M	3/6,
Height $\leq 1.6$	0/2,
$1.6 < \text{Height} \leq 1.7$	0/2,
$1.7 < \text{Height} \leq 1.8$	0/3,
$1.8 < \text{Height} \leq 1.9$	0/4,
$1.9 < \text{Height} \leq 2.0$	1/2,
$2.0 < \text{Height}$	2/2.

Since the largest fraction is 2/2, the generated rule is

If  $2.0 < \text{Height}$ , then class = Tall.

There is no need to add any additional predicates to the rule because all predicates that satisfy  $2.0 < \text{Height}$  are Tall. To generate additional rules for the Tall class, examine the remaining tuples in the training set and recalculate the accuracy of the corresponding predicates:

Gender = F	0/9,
Gender = M	1/4,
Height $\leq 1.6$	0/2,

---

**Algorithm 8** PRISM Algorithm

---

**Input:** $D$  //Training Data $C$  //Classes**Output:** $R$  //Rules**Algorithm:**

//PRISM algorithm generates rules based on best attribute-value pairs

 $R = \emptyset$ ;**for all**  $C_j \in C$ **repeat** $T = D$ ; //All instances of class  $C_j$  will be systematically removed from  $T$  $p = \text{true}$ ; //create a new rule with empty left hand side $r = (\text{If } p, \text{ then } C_j)$ ;**repeat****for all** attribute  $A$  value  $v$  pair found in  $T$  **do**calculate  $(|(tuples \in T \text{ with } A = v) \wedge p \wedge (\in C_j)| / |(tuples \in T \text{ with } A = v) \wedge p|)$ **end for**find  $A = v$  that maximizes this value; $p = p \wedge (A = v)$ ; $T =$  tuples in  $T$  that satisfy  $A = v$ ;**until** all tuples in  $T$  belong to  $C_j$ ; $D = D - T$ ; $R = R \cup r$ ;**until** there are no tuples in  $D$  that belong to  $C_j$ ;**end for**

---

1.6 < Height  $\leq$  1.7 0/2,1.7 < Height  $\leq$  1.8 0/3,1.8 < Height  $\leq$  1.9 0/4,1.9 < Height  $\leq$  2.0 1/2.

Here, the last Height range is the most accurate, but another predicate needs to be added as the only one of the tuples that satisfies it is Tall.

Table 9. Data Height Classification [42].

Name	Gender	Height (m)	Output
Kristina	F	1.6	Short
Jim	M	2	Tall
Maggie	F	1.9	Medium
Martha	F	1.88	Medium
Stephanie	F	1.7	Short
Bob	M	1.85	Medium
Kathy	F	1.6	Short
Dave	M	1.7	Short
Worth	M	2.2	Tall
Steven	M	2.1	Tall
Debbie	F	1.8	Medium
Todd	M	1.95	Medium
Kim	F	1.9	Medium
Amy	F	1.8	Medium
Wynette	F	1.75	Medium

This problem is caused by the arbitrary range divisions performed earlier. Dividing the range into subranges produces the probabilities:

$$\begin{aligned} 1.9 < \text{Height} \leq 1.95 & \quad 0/1, \\ 1.95 < \text{Height} \leq 2.0 & \quad 1/1. \end{aligned}$$

The last predicate has the largest fraction, so it is added to the rule:

$$\text{If } 2.0 < \text{Height AND } 1.95 < \text{Height} \leq 2.0, \text{ then class} = \text{Tall},$$

which is equivalent to

$$\text{If } 1.95 < \text{Height}, \text{ then class} = \text{Tall}.$$

Thus, all Tall tuples are classified.

### 4.3. Other Algorithms

Basic covering algorithms construct rules that classify training data perfectly, that is, they tend to overfit the training set causing insufficient generalization and difficulty for processing new data. However, for applications in real world domains, methods for handling noisy data, mechanisms for avoiding overfitting even on training data, and relaxation requirements of the constraints are needed [31]. Pruning is one of the ways of dealing with these problems, and it approaches the problem of overfitting by learning a general concept from the training set “to improve the prediction of unseen

instances” [53]. The concept of Reduced Error Pruning (REP) was developed by [53], where some of the training examples were withheld as a test set and performance of the rule was measured on them. Also, Incremental Reduced Error Pruning (IREP) has proven to be efficient in handling overfitting, and it forms the basis for RIPPER [32]. SLIPPER (Simple Learner with Iterative Pruning to Produce Error Reduction) [33] uses “confidence-rated boosting to learn an ensemble of rules.”

## 5. Applications of Covering Algorithms

Covering algorithms are widely used for deriving classification rules applied in medical sciences for diagnosing illnesses, business planning, banking, government and different disciplines of science. Particularly, covering algorithms have deep roots in machine learning [156]. Within data mining, covering algorithms including SWAP-1, RIPPER, and DAIRY are used in text classification [71], adapted in gene expression programming for discovering classification rules [169].

## 6. Summary

Covering algorithms are appealing because they generate rules that are relatively simple, easy to interpret and manipulate, and each of the rules they produce seems to represent “an independent nugget of knowledge” [156]. The simple algorithms are fast and perform with comparable accuracy compared to sophisticated decision tree algorithms. Thus, they are particularly attractive for providing a baseline performance using a simple rudimentary technique to be used before progressing to more sophisticated algorithms.

On the other hand, basic covering algorithms suffer from overfitting the training data, which makes their application to independent (noisy) data sets difficult. In addition, handling missing values and ambiguous assignments to classes lead to unstable results. These problems are typically minimized with the use of more advanced tree pruning techniques (similar to the ones employed by decision trees), and with the application of specialized methods.

This page is intentionally left blank

## CHAPTER 9

### CLUSTERING: AN OVERVIEW

Matthew Aldridge

*Department of Computer Science, University of Tennessee  
203 Claxton Complex, Knoxville, TN 37996-3450, USA  
aldridge@cs.utk.edu*

#### Overview

This chapter introduces the concept of clustering, sometimes referred to as unsupervised classification. Clustering is a method employed by many disciplines to group objects into previously undefined categories with the goal of abstracting data for easier interpretation or to gain new knowledge thereof. The typical process of a clustering activity is discussed along with a brief overview of a few major clustering methodologies and related issues.

**Keywords:** Clustering, unsupervised learning, unsupervised classification, proximity measures, outliers.

#### 1. Introduction

*Clustering*, defined broadly, is the grouping of similar objects [66]. More specifically, clustering is the unsupervised classification of *patterns* into groups based upon similarity, where a pattern is a representation of features or observations made on an object. Upon first glance, the problem of clustering is quite similar to that of classification. It should be noted, however, that the key difference between the two is the *unsupervised* nature of clustering. Traditional supervised classification involves a special input training set or predefined set of classes into which objects are placed, whereas clustering attempts to derive meaningful classes solely from the data.

The problem of clustering arises in many different scientific fields, and thus, a vast amount of literature has been produced on the subject. For a more comprehensive survey of data clustering, along with an extensive list of references, the reader is directed to [74].

## 2. Motivation

Clustering is often a critical component of the data mining or knowledge discovery process. Data mining tasks frequently involve large amounts of data, and clustering methods may be employed during the data transformation stage of the knowledge discovery process [42]. Doing so effectively abstracts or compresses the data and allows the subsequent data mining algorithms to treat each data cluster as a single datum. Clustering may also be utilized to aid the practitioner in visualizing and interpreting data mining results, possibly revealing previously unknown tendencies in the data.

The primary focus of this chapter is to establish the general framework of the clustering process from initial data representation to assessment of clustering results. A brief discussion of the two major clustering algorithm approaches is made, and a handful of practical applications of clustering techniques are introduced.

## 3. The Clustering Process

Prior to discussing the specifics of the clustering process, a brief treatment on terminology and notation must be made. The notational system used in this chapter follows directly from [39].

An *object*, in the current context, could refer to a physical object (e.g., an animal), as well as to an abstract notion (e.g., a style of writing). A measurement made on such an object is called a *feature*. A feature may be either quantitative (e.g., a specific weight) or qualitative (e.g., “heavy”). A *pattern* is a set of features describing an object and is generally represented as a vector. The set of patterns for all objects in the domain of interest is represented as a *pattern matrix*. Given a set of  $n$  patterns with  $d$  features, the pattern matrix is an  $n \times d$  matrix. The  $j$ th feature of pattern  $i$  is  $x_{ij}$ . The  $i$ th pattern itself is denoted by the column vector  $\mathbf{x}_i$ , where

$$\mathbf{x}_i = [x_{i1} \ x_{i2} \ \cdots \ x_{id}]^T.$$

A *proximity matrix* is a square, symmetric matrix, with rows and columns both corresponding to patterns. The  $(i, j)$  entry represents the proximity or degree of closeness between patterns  $i$  and  $j$ . The proximity

matrix may be a *dissimilarity* matrix, in which case a large  $(i, j)$  represents a large degree of separation between  $i$  and  $j$ . Conversely, a large value in a *similarity* matrix indicates a close resemblance between the two patterns.

The pattern clustering process typically involves the following steps [74]:

- (i) Create a pattern representation.
- (ii) Define a pattern proximity measure appropriate to the data domain.
- (iii) Apply a clustering algorithm.
- (iv) Perform data abstraction, if needed.
- (v) Assess cluster output, if needed.

Each of these steps is discussed individually in the following subsections.

### 3.1. *Pattern Representation*

As noted in [74], *pattern representation* “refers to the number of classes, the number of available patterns, and the number, type, and scale of the features available to the clustering algorithm.” The generation of a pattern representation is often dependent exclusively on the data available and the clustering algorithm being used. However, it is sometimes possible and helpful for the practitioner to employ *feature selection* and/or *feature extraction* to refine the pattern representation in order to provide more appropriate clustering results.

Feature selection involves identifying the most effective features for discriminating patterns. For example, in discriminating types of vehicles, a “color” attribute would likely not be as useful as “size,” “number of wheels,” “passenger seating,” and “fuel type” attributes. In such a situation, the “color” feature could safely be excluded from consideration. Furthermore, some features selected may be weighted according to their relative importance [76].<sup>1</sup> Features may be highly variable among patterns, rare, or even redundant. Feature weightings must also be taken into account by the pattern proximity measure.

Feature extraction is the process of transforming the available features to provide new features. Given a set of images of human faces, one possible application of feature extraction is the extraction of identifiable characteristics usable by a face recognition system.

In addition to which features should be utilized, feature representation is important in creating a pattern representation. Figure 25 illustrates a

---

<sup>1</sup>In a sense, deciding to exclude certain features entirely effectively gives those features a relative weighting of zero.

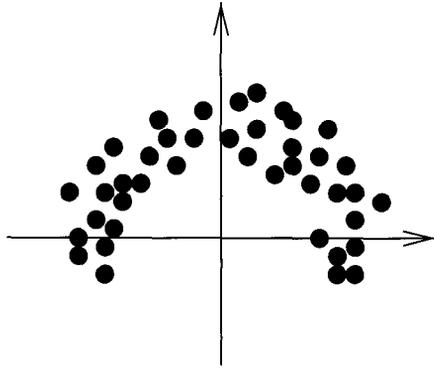


Fig. 25. Different coordinate systems in a pattern representation would yield different clustering results for these points about the origin. Figure adapted from [74].

simple example where different coordinate system choices in a pattern representation would yield different clustering results. Each point in the figure is approximately equidistant from the origin. If one chose to represent the points in Cartesian space, a clustering algorithm may likely produce multiple clusters. If the points were instead represented using polar coordinates, the similar radius components among the points could influence the creation of a single cluster.

### 3.2. Pattern Proximity Measures

Indices of proximity between all pairs of patterns are required by clustering algorithms [39]. A well-known index of dissimilarity between two patterns, say  $q$  and  $r$  in a pattern matrix, with continuous features is the Minkowski metric:

$$d(q, r) = \left( \sum_{j=1}^d |x_{qj} - x_{rj}|^m \right)^{(1/m)} .$$

For  $m = 2$ ,  $d(q, r)$  produces the Euclidean distance between the two patterns, while  $m = 1$  results in the Manhattan or city-block distance. It is often necessary to normalize the results of Minkowski metrics, because features of large degree may otherwise dominate the others. Normalization may be accomplished by dividing the distance for each feature by the feature's range. This effectively maps distances to the range  $[0, 1]$ . If the distribution of the data is known *a priori*, outliers may be avoided by "trimming"

the range by the upper and lower few percent and then mapping values outside of this range to the new minimum and maximum values [155]. For many real world clustering applications, though, the distribution of the data may not be known, and other methods must be employed to handle outliers.

Common indices of similarity for binary or nominal (unordered) features include the Jaccard coefficient (see Chap. 3) and the simple matching coefficient [39]. However, a problem arises when patterns include both continuous and nominal features, since no single proximity measure can handle such diverse pattern comparisons. In [155], several methods are proposed to handle heterogeneous patterns but are beyond the scope of this chapter.

Once the indices of dissimilarity or similarity have been computed, the proximity matrix is populated and may be examined to identify outliers based upon high distances from other patterns. Depending upon the data domain and the intended goal of the clustering process, such outliers may be removed prior to applying a clustering algorithm. However, outliers sometimes provide useful information that would otherwise be lost if removed prematurely. For example, consider a credit agency that uses clustering to model its customers' typical credit card purchasing habits based upon geographic locality. Outliers in this case would represent purchases that occur far outside of a customer's usual shopping areas, and may indicate possible credit fraud.

### 3.3. Clustering Algorithms

Clustering approaches may be broadly categorized into two methods: hierarchical and partitional. Although the focus of this chapter is not to provide individual algorithmic details, a brief discussion of both major methods follows.

#### 3.3.1. Hierarchical Algorithms

Hierarchical clustering algorithms create nested sets of clusters, producing a binary tree structure known as a *dendrogram*. Each node in the dendrogram represents a cluster. The root node is a cluster that includes every individual pattern, and each child thereof contains a subcluster of its parent node. The height of each node is proportional to the measure of similarity or dissimilarity between its two subclusters. Typically, the leaf nodes in the dendrogram represent individual patterns.

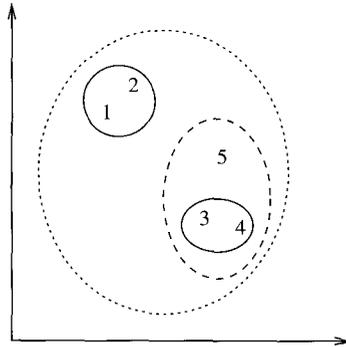


Fig. 26. A possible set of nested clusters for five patterns. Note that singleton clusters are not indicated.

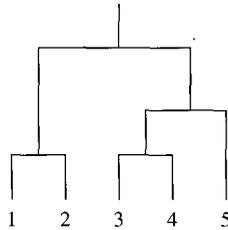


Fig. 27. The dendrogram corresponding to the sample clustering given in Fig. 26.

Figure 26 illustrates a possible set of nested clusters for five patterns, where each line style represents a different level in the cluster hierarchy. The corresponding dendrogram is shown in Fig. 27.

A hierarchical clustering may be produced by either an *agglomerative* or a *divisive* algorithm. An agglomerative algorithm begins with each pattern in its own individual cluster (leaf node in the dendrogram) and proceeds to pair clusters until the all-inclusive cluster (root node in the dendrogram) is created. A divisive algorithm works in the reverse order, by beginning with the root node and recursively splitting each node until either each pattern is in a singleton cluster or some stopping criterion is reached (e.g., the requested number,  $k$ , of clusters is reached).

A major advantage of hierarchical clustering is the embedded flexibility regarding the level of granularity. However, the creation of a hierarchical clustering may involve excessive time and space constraints, because the algorithm used must make an iteration for each level in the hierarchy [42].

### 3.3.2. *Partitional Algorithms*

Unlike hierarchical methods, partitional clustering algorithms produce a single partition of the patterns. Partitional clustering requires less time and space than hierarchical clustering, a very desirable quality when working on large data sets. However, partitional methods require the user to choose a specific number,  $k$ , of clusters to be created. Additionally, while a partitional method produces only one final set of clusters, the algorithm may create the final set iteratively, beginning each time with a different starting configuration and choosing the best result from all the runs [74].

### 3.4. *Data Abstraction*

The data abstraction process involves extracting a simple and compact representation of the clustering results [74]. This may be done either to allow for more automatic processing of the results or to provide the results in a human-comprehensible fashion. The abstraction process typically includes producing a description of each cluster, because the patterns are not placed into predefined classes, as with classification.

### 3.5. *Cluster Assessment*

As [74] notes, the cluster assessment process is multi-faceted. One task involved is the assessment of the data domain itself rather than the clustering algorithm. It should be verified that the data set contains reasonable clusters prior to performing further cluster analysis. During such verification, the practitioner may make the determination that the data set has a low *cluster tendency*, meaning there is actually little or no benefit in attempting to perform clustering.

Another facet of cluster assessment is the validation of algorithm results. Given a cluster,  $K_m$  of  $N$  points  $\{t_{m1}, t_{m2}, \dots, t_{mN}\}$ , three typical measurements of  $K_m$  are [42]:

$$\text{centroid} = C_m = \frac{\sum_{i=1}^N t_{mi}}{N},$$

$$\text{radius} = R_m = \sqrt{\frac{\sum_{i=1}^N (t_{mi} - C_m)^2}{N}},$$

and

$$\text{diameter} = D_m = \sqrt{\frac{\sum_{i=1}^N \sum_{j=1}^N (t_{mi} - t_{mj})^2}{(N)(N-1)}}.$$

The centroid may be viewed as the “middle” of the cluster, but may not actually be a point itself. Alternatively, a *medoid*, or a centrally located point, may be identified. The radius represents the average mean squared distance from any point to the cluster’s centroid. The diameter is the square root of the average mean squared distance between all pairs of points in the cluster. The radius and density measurements may be used in determining the density of a cluster. Another common clustering result evaluation involves determining the distance between clusters, either by utilizing the centroids of clusters or individual points themselves. Although such objective characteristics are used to measure and describe clusters, the relative importance of each characteristic must be judged subjectively, depending upon the data domain and desired results.

In [39], three types of validation studies are explored: external, internal, and relative. An *external* assessment of validity compares the clustering result to an *a priori* structure and attempts to quantify the match between the two. An *internal* assessment of validity attempts to determine if the clustering result is intrinsically appropriate for the data. Such an assessment may consider, for example, whether a given cluster is unusually compact or isolated compared to other clusters of the same size in random data. Finally, a *relative* assessment compares two clustering results and weighs their relative merit. Figure 28 illustrates a simple example where the same

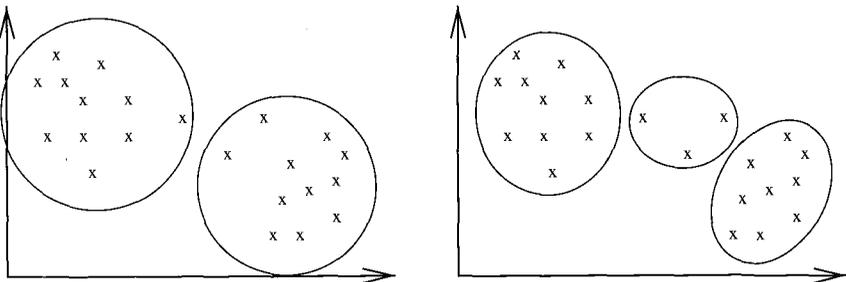


Fig. 28. An illustration of the differences between a 2-cluster and a 3-cluster of the same data.

data set is partitioned into both two and three clusters. A relative assessment would involve analyzing both cluster sets and determining which is more appropriate for the data domain.

#### **4. Current Applications**

Traditional classification, and by extension clustering, owes its origins in part to work in biological taxonomy [76]. Clustering today, however, is a vastly multi-disciplinary practice. Researchers in image analysis and the related practice of image segmentation use clustering techniques to partition images into regions, “each of which is considered to be homogeneous with respect to some image property of interest” [74]. Clustering may be used by data miners in the business world to identify potential marketing targets or to discover consumer trends. [142] presents a novel application of clustering in the field of investigative psychology for the classification of deviant and criminal behavior with respect to psychological disorders.

#### **5. Summary**

Data clustering is often an essential component of a larger knowledge discovery task, both in terms of preprocessing or transforming data and visualizing data mining results. Due to the unsupervised nature of clustering, previously unrealized data groupings may be discovered. Though there are no predefined classes for the clustering process, an expert for the data domain is still required to ensure that a reasonable pattern representation is used, choose an appropriate proximity measure and clustering algorithm, and perform the final cluster analysis.

This page is intentionally left blank

## CHAPTER 10

### CLUSTERING: HIERARCHICAL ALGORITHMS

Huádong Liu

*Department of Computer Science, University of Tennessee  
203 Claxton Complex, Knoxville, TN 37996-3450, USA  
hliu@cs.utk.edu*

#### Overview

Hierarchical clustering is an iterative method of clustering data objects. The agglomerative and divisive hierarchical algorithms are discussed in this chapter. Because the most important part of hierarchical clustering is the definition of distance between two clusters, several basic methods of calculating the distance are introduced. The chapter concludes with a comparison of the agglomerative and divisive algorithms.

**Keywords:** Agglomerative, average linkage method, centroid method, complete linkage method, DIANA algorithm, divisive, hierarchical algorithm, median method, single linkage method, Ward method.

#### 1. Introduction

Clustering is the unsupervised classification of data objects into different groups or clusters. Data objects are assumed to be similar to one another within the same cluster. However, data objects from different clusters should not be alike. In supervised classification, a collection of labelled (pre-classified) data objects is provided. The challenge is to label newly encountered data objects. In unsupervised classification, the problem is to group a collection of unlabelled data objects into meaningful clusters. Clustering combines techniques from different disciplines such as mathematics, statistics, artificial intelligence and databases. It can be used as a stand-alone tool to obtain insight into data objects or as a preprocessing step for

more complex algorithms. Clustering is particularly useful for the exploration of relationships among the data objects to make an assessment of their structure [73].

Clustering can be divided into two basic types: *hierarchical* and *partitional*. Within each of the types there exist many subtypes and different algorithms for finding clusters. Hierarchical clustering proceeds successively by either merging smaller clusters into larger ones, or by splitting larger clusters. The algorithms differ in how to decide which two small clusters are merged or which large cluster is split. Partitional clustering, on the other hand, attempts to directly decompose the data set into a set of disjoint clusters. Such algorithms use the desired number of clusters to drive how the final set is created [42]. Hierarchical clustering produces different clusters at different iterations. It provides more structural information about data objects at the expense of more computation. This is especially useful when no prior knowledge (e.g., the number of clusters) of data objects is known. The hierarchical clustering algorithms will be discussed in this chapter. However, readers interested in further details of partitional algorithms will benefit from reading [74].

## 2. Motivation

The goal of clustering is to reduce the amount of data by categorizing or grouping similar data objects together. Such grouping is pervasive in the way humans process information, and one of the motivations for using clustering algorithms is to provide automated tools for constructing categories or taxonomies [79]. Hierarchical clustering is natural in this sense and can provide more information about data objects. The hierarchical clustering algorithm and its derivatives have been implemented in a number of commercial data mining software packages, such as the Daylight<sup>TM</sup> Chemical Information System.

Hierarchical clustering is either *agglomerative* or *divisive*. Agglomerative algorithms start with all data objects as individual clusters. These clusters are then merged according to their distances, until all data objects are grouped into a single cluster. Divisive algorithms start with one cluster containing all data objects. The initial cluster is successively separated into smaller clusters until the number of clusters is equal to the number of data objects. The results of hierarchical clustering are often represented in the form of *dendrograms*. The dendrogram is a tree-like diagram that can depict sets of different clusters that have been produced at successive iterations of

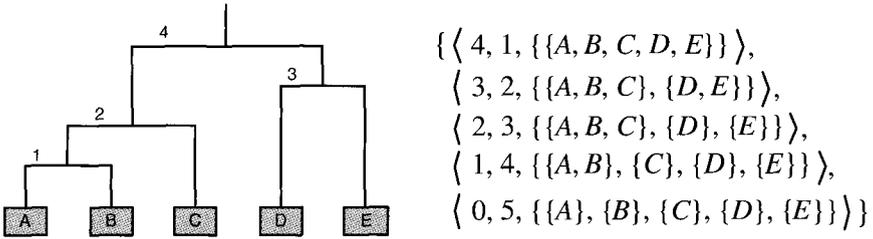


Fig. 29. A sample dendrogram.

the hierarchical algorithm. It can be represented by a set of ordered triples  $\langle d, k, S \rangle$ , where  $d$  is the threshold distance,  $k$  is the number of clusters, and  $S$  is the set of clusters. Figure 29 is an example of the representations of a dendrogram.

An important issue in hierarchical clustering is the distance between two data objects so that clusters can be formed. Distance functions such as City Block, Euclidean and Tanimoto are used to determine the distance between two data objects. A distance function usually yields a higher value for pairs of data objects that are less similar to one another. The Euclidean distance is used to calculate the distance between two data objects in this chapter. This distance is defined as

$$\text{dist}(X, Y) = \sqrt{\sum_{i=1}^m (X_i - Y_i)^2},$$

where  $m$  is the dimensionality of the data object  $X$  and  $Y$ .

In this chapter, Sec. 3 discusses agglomerative hierarchical algorithms. Methods of calculating distance between clusters — the single linkage method, the complete linkage method, the average linkage method, the centroid method and the Ward method — are introduced. The divisive hierarchical algorithm, particularly the DIANA algorithm is discussed in Sec. 4. Finally, a summary of the chapter is provided in Sec. 5.

### 3. Agglomerative Hierarchical Algorithms

Agglomerative hierarchical clustering is a bottom-up clustering method. The main goal of agglomerative algorithms is to iteratively merge a set of clusters based on some distance measure. An agglomerative algorithm starts with each data object in a cluster. In each successive iteration, the algorithm

agglomerates the closest pair of clusters by satisfying some distance criteria, until all data objects are in one cluster.

The most important part of agglomerative algorithms is the calculation of distance between clusters. Before going into detail about different methods of calculating the distance, a general agglomerative algorithm will be introduced. The algorithm is adapted from [42]. It assumes that a set of data objects  $E = \{E_1, E_2, \dots, E_n\}$  is given as input. The output of the algorithm is a dendrogram  $DE$ , which is represented as a set of ordered triples  $\langle d, k, S \rangle$ , where  $d$  is the threshold distance,  $k$  is the number of clusters, and  $S$  is the set of clusters, as defined earlier. Initially, each data object  $E_i$  is considered to be in one cluster  $C_i$ . A  $k \times k$  distance matrix  $Dist$  is used to maintain the distances between clusters.

Algorithm 9 starts with each individual data object in its own cluster and iteratively merges two clusters with the minimum distance until all data objects belong to one cluster. Some variations of the algorithm might allow multiple clusters with identical distance to be merged at one iteration. The *ComputeDistance* function calculates distances between clusters using the current set of clusters  $S$ . Agglomerative algorithms differ in how the distance matrix is calculated, and hence how the clusters are merged at each level. The *single linkage method*, the *complete linkage method*, and the *average linkage method* are the simplest methods for obtaining distances between clusters. These methods will be introduced with examples in the following sections.

### 3.1. The Single Linkage Method

The *single linkage method* is also called the nearest neighbor method or minimum distance method. The distance between clusters  $C_i$  and  $C_j$  is defined by

$$Dist(C_i, C_j) = \min_{X \in C_i, Y \in C_j} dist(X, Y).$$

The single linkage method builds clusters hierarchically starting from the most similar cluster pairs. The distance between two clusters is defined as the distance between the two closest data objects, one from each cluster. Unfortunately, the single linkage method suffers from a chaining effect [110], in that it has a tendency to produce straggly and elongated clusters [74]. One pair of close data objects can cause two clusters to be merged although other data objects are far away from each other.

**Algorithm 9** Agglomerative algorithm**Input:**  $E = \{E_1, E_2, \dots, E_n\}$ **Output:**  $DE$ 


---

```

for  $i = 1$  to  $n$  do
   $C_i \leftarrow \{E_i\}$ ;
end for
 $d \leftarrow 0$ ;
 $k \leftarrow n$ ;
 $S \leftarrow \{C_1, \dots, C_n\}$ ;
 $DE \leftarrow \langle d, k, S \rangle$ ;
repeat
   $Dist \leftarrow \text{ComputeDistance}(S)$ ;
   $d \leftarrow \infty$ ;
  for  $i = 1$  to  $k - 1$  do
    for  $j = i + 1$  to  $k$  do
      if  $Dist(i, j) < d$  then
         $d \leftarrow Dist(i, j)$ ;
         $u \leftarrow i$ ;
         $v \leftarrow j$ ;
      end if
    end for
  end for
   $k \leftarrow k - 1$ ;
   $C_{new} \leftarrow C_u \cup C_v$ ;
   $S \leftarrow S \cup C_{new} - C_u - C_v$ ;
   $DE \leftarrow DE \cup \langle d, k, S \rangle$ ;
until  $k = 1$ ;

```

---

**Example 8:** Suppose a market consulting company wishes to group customers based on their age and loyalty to grocery brands. Table 10 shows the sample data from the customer survey.

Table 10. Sample data for Example 8.

ID	E1	E2	E3	E4	E5
Age	20	30	40	50	60
Brand Loyalty (%)	40	64	55	74	85

Cluster ID	C1 = {E1}	C2 = {E2}	C3 = {E3}	C4 = {E4}
C2 = {E2}	26.0			
C3 = {E3}	25.0	13.5		
C4 = {E4}	45.3	22.4	21.5	
C5 = {E5}	60.2	36.6	36.1	14.9

**Iteration 1**  
 $d = 13.5, k = 4, K = \{\{E1\}, \{E2, E3\}, \{E4\}, \{E5\}\}$

Cluster ID	C1 = {E1}	C2 = {E2, E3}	C3 = {E4}
C2 = {E2, E3}	25.0		
C3 = {E4}	45.3	21.5	
C4 = {E5}	60.2	36.1	14.9

**Iteration 2**  
 $Dist(C1, C2) = \min\{26.0, 25.0\} = 25.0$   
 $Dist(C2, C3) = \min\{22.4, 21.5\} = 21.5$   
 $Dist(C2, C4) = \min\{36.6, 36.1\} = 36.1$   
 $d = 14.9, k = 3, K = \{\{E1\}, \{E2, E3\}, \{E4, E5\}\}$

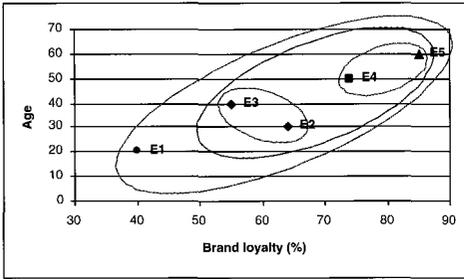
Cluster ID	C1 = {E1}	C2 = {E2, E3}
C2 = {E2, E3}	25.0	
C3 = {E4, E5}	45.3	21.5

**Iteration 3**  
 $Dist(C1, C3) = \min\{45.3, 60.2\} = 45.3$   
 $Dist(C2, C3) = \min\{21.5, 26.1\} = 21.5$   
 $d = 21.5, k = 2, K = \{\{E1\}, \{E2, E3, E4, E5\}\}$

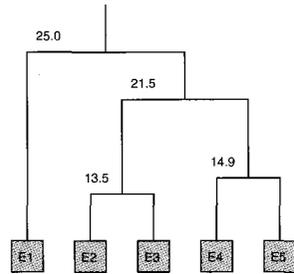
Cluster ID	C1 = {E1}
C2 = {E2, E3, E4, E5}	25.0

**Iteration 4**  
 $Dist(C1, C2) = \min\{25.0, 45.3\} = 25.0$   
 $d = 25.0, k = 1, K = \{\{E1, E2, E3, E4, E5\}\}$

(a) Iterations



(b) Clusters



(c) Dendrogram

Fig. 30. An example for the single linkage method.

Figure 30(a) illustrates iterations of the agglomerative algorithm using the single linkage method to obtain the dendrogram for the sample data given in Table 10. Figure 30(b) shows the clusters formed in each iteration, and Figure 30(c) gives the resulting dendrogram.

### 3.2. The Complete Linkage Method

The *complete linkage* method is also called the farthest neighbor or maximum distance method. The distance  $Dist(C_i, C_j)$  between two clusters  $C_i$  and  $C_j$  is defined as the longest distance from any data object of one cluster

to any data object of the other cluster:

$$Dist(C_i, C_j) = \max_{X \in C_i, Y \in C_j} dist(X, Y).$$

The complete linkage method works in the opposite manner to the single linkage method. As a result, the complete linkage method produces tightly bounded clusters [11].

**Example 9:** For the same sample data given in Example 8, Fig. 31(a) illustrates iterations of the agglomerative algorithm using the complete linkage method to obtain the dendrogram. Figure 31(b) shows the clusters formed in each iteration, and Fig. 31(c) gives the resulting dendrogram.

Cluster ID	C1 = {E1}	C2 = {E2}	C3 = {E3}	C4 = {E4}
C2 = {E2}	26.0			
C3 = {E3}	25.0	13.5		
C4 = {E4}	45.3	22.4	21.5	
C5 = {E5}	60.2	36.6	36.1	14.9

**Iteration 1**  
 $d = 13.5, k = 4, K = \{\{E1\}, \{E2, E3\}, \{E4\}, \{E5\}\}$

Cluster ID	C1 = {E1}	C2 = {E2, E3}	C3 = {E4}
C2 = {E2, E3}	26.0		
C3 = {E4}	45.3	22.4	
C4 = {E5}	60.2	36.6	14.9

**Iteration 2**  
 $Dist(C1, C2) = \max\{26.0, 25.0\} = 26.0$   
 $Dist(C2, C3) = \max\{22.4, 21.5\} = 22.4$   
 $Dist(C2, C4) = \max\{36.6, 36.1\} = 36.6$   
 $d = 14.9, k = 3, K = \{\{E1\}, \{E2, E3\}, \{E4, E5\}\}$

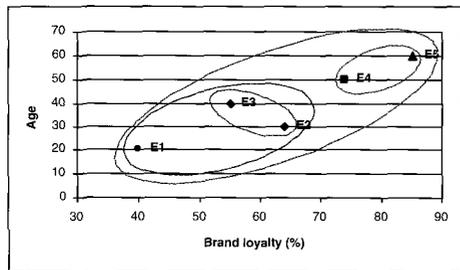
Cluster ID	C1 = {E1}	C2 = {E2, E3}
C2 = {E2, E3}	26.0	
C3 = {E4, E5}	60.2	36.6

**Iteration 3**  
 $Dist(C1, C3) = \max\{45.3, 60.2\} = 60.2$   
 $Dist(C2, C3) = \max\{22.4, 36.6\} = 36.6$   
 $d = 26.0, k = 2, K = \{\{E1, E2, E3\}, \{E4, E5\}\}$

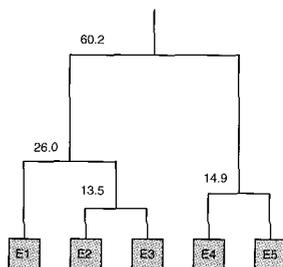
Cluster ID	C1 = {E1, E2, E3}
C2 = {E4, E5}	60.2

**Iteration 4**  
 $Dist(C1, C2) = \max\{60.2, 45.3\} = 60.2$   
 $d = 60.2, k = 1, K = \{\{E1, E2, E3, E4, E5\}\}$

(a) Iterations



(b) Clusters



(c) Dendrogram

Fig. 31. An example for the complete linkage method.

### 3.3. The Average Linkage Method

The *average linkage* method is a compromise between the single linkage method and the complete linkage method. It avoids the extremes of large or compact clusters. The distance  $Dist(C_i, C_j)$  between clusters  $C_i$  and  $C_j$  is defined by

$$Dist(C_i, C_j) = \frac{\sum_{X \in C_i} \sum_{Y \in C_j} dist(X, Y)}{|C_i| \times |C_j|},$$

where  $|C_k|$  is the number of data objects in cluster  $C_k$ .

The average linkage method computes the distance between two clusters as the arithmetic average of distances between all the data objects in one cluster and all the data objects in the other. An illustration of this method is given in Fig. 32(a).

The average linkage method is unweighted in that all data objects receive equal weight in the computation of distances between clusters. In some cases, it may occur that different groups of data objects are unequally sampled in the population. Thus, the proportion of each group in the sample does not reflect its proportion in the population. In this situation, the *weighted arithmetic average clustering*, WPGMA, is often used. For more information about WPGMA algorithms, refer to [139].

### 3.4. The Centroid Method

The centroid linkage method is also a compromise between the single linkage method and the complete linkage method. In this method, the mean cluster position, *centroid*, is determined by averaging the positions of all data objects within the cluster. A cluster centroid is the typical data object of the cluster, whether it exists or not. Given a cluster  $C_i$  with  $n$  data objects  $X_1, \dots, X_n$ . Let  $m$  be the number of dimensions of each data object. The

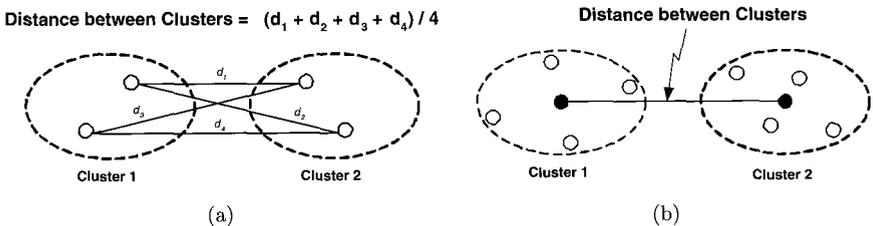


Fig. 32. (a) Average linkage distance; (b) centroid distance.

centroid of cluster  $C_i$  is defined as  $X_c = (c_1, \dots, c_m)$ , with

$$c_j = \frac{1}{n} \sum_{k=1}^n X_{kj},$$

where  $X_{kj}$  is the  $j$ th dimension of the  $k$ th data object in the cluster  $C_i$ .

With the centroid method, the distance between two clusters is defined as the distance between the pair of cluster centroids. An illustration of the centroid linkage method is given in Fig. 32(b). The main difference between the average linkage method and the centroid linkage method is that in the former, the distance is an arithmetic average, while in the latter the distance is a geometric average.

If the sizes of two clusters to be merged are very different, the centroid of the newly formed cluster will be close to that of the larger cluster and may remain within that cluster. This is a disadvantage of the centroid method. Gower suggested an alternative strategy, called the *median method*. The median method could be made suitable for distance measuring in such a situation [57].

### 3.5. The Ward Method

Ward proposed a clustering method to form clusters in a manner that minimizes the information loss associated with each grouping [148]. The information loss is defined in terms of the *Error Sum of Squares (ESS)*. Let  $X$  be a set of data objects,  $X = \{X_1, \dots, X_n\}$ . Each data object has  $m$  dimensions,  $X_i = (X_{i1}, \dots, X_{im})$ . For a set of clusters  $C = \{C_1, \dots, C_K\}$  of  $X$ , where  $K$  is the number of clusters, the *ESS* for each cluster  $C_k$  is defined as

$$ESS(C_k) = \sum_{X_i \in C_k} \sum_{j=1}^m (X_{ij} - \bar{X}_{kj})^2.$$

$\bar{X}_{kj}$  is the cluster mean of the  $j$ th dimension, defined as

$$\bar{X}_{kj} = \frac{1}{|C_k|} \sum_{X_i \in C_k} X_{ij},$$

where  $|C_k|$  is the number of data objects in cluster  $C_k$ .

The *ESS* for the set of clusters  $C$  is defined as the sum of *ESS* of all clusters:

$$ESS(C) = \sum_{k=1}^K ESS(C_k) = \sum_{k=1}^K \sum_{X_i \in C_k} \sum_{j=1}^m (X_{ij} - \bar{X}_{kj})^2.$$

The clustering procedure using the Ward method has a similar structure with agglomerative algorithms using other methods (e.g., the single linkage method and the complete linkage method) discussed above. However, the Ward procedure computes the *ESS* instead of distances between clusters in each iteration as the criteria for merging clusters.

#### 4. Divisive Hierarchical Algorithms

The divisive hierarchical clustering is a less common top-down clustering process. It works in a similar way as agglomerative clustering but in the opposite direction. Divisive hierarchical clustering starts with a single cluster containing all data objects. The initial cluster is divided into two clusters such that the data objects in one cluster are far from data objects in the other. The method then successively splits resulting clusters until each data object is in its own cluster.

Two types of divisive hierarchical clustering are *monothetic* methods and *polythetic* methods. The monothetic methods divide data objects based on the processing of a single specified attribute, while the polythetic methods are based on values of several attributes. Monothetic methods are generally more efficient than the corresponding polythetic methods, but tend to give poor results.

The DIANA algorithm is one of the most well-known divisive hierarchical clustering algorithms. It was introduced by Kaufmann and Rousseeuw [81] and has been implemented in many statistical analysis packages. The DIANA algorithm is based on the distance between an object  $X_i$  and a cluster  $C_k$ . The distance is defined as

$$\bar{D}(X_i, C_k) = \begin{cases} \frac{1}{|C_k| - 1} \sum_{X_j \in C_k, j \neq i} dist(X_i, X_j), & X_i \in C_k, \\ \frac{1}{|C_k|} \sum_{X_j \in C_k} dist(X_i, X_j), & X_i \notin C_k. \end{cases}$$

Again,  $|C_k|$  is the number of data objects in cluster  $C_k$ . The DIANA algorithm can be summarized as follows.

---

**Algorithm 10** DIANA algorithm

---

**Input:**  $E = \{E_1, E_2, \dots, E_n\}$ **Output:**  $DE$  $d \leftarrow \infty;$  $k \leftarrow 1;$  $S \leftarrow \{\{E_1, E_2, \dots, E_n\}\};$  $DE \leftarrow \langle d, k, S \rangle;$ **repeat** $A \leftarrow$  the cluster containing two data objects with the longest distance  $dist;$  $B \leftarrow \emptyset;$  $S \leftarrow S - A;$  $X_i \leftarrow$  data object in  $A$  with maximum  $\overline{D}(X_i, A);$  $A \leftarrow A - \{X_i\};$  $B \leftarrow B \cup \{X_i\};$ **repeat****for** all data object  $X_j$  in  $A$  **do** $e(j) \leftarrow \overline{D}(X_j, A) - \overline{D}(X_j, B);$ **end for****if**  $\exists e(j) > 0$  **then** $X_k \leftarrow$  data object in  $A$  with maximum  $e(j);$  $A \leftarrow A - \{X_k\};$  $B \leftarrow B \cup \{X_k\};$  $split \leftarrow TRUE;$ **else** $split \leftarrow FALSE;$ **end if****until**  $split == FALSE;$  $d \leftarrow dist;$  $k \leftarrow k + 1;$  $S \leftarrow S \cup A \cup B;$  $DE \leftarrow DE \cup \langle d, k, S \rangle;$ **until**  $k = n;$ 

---

Algorithm 10 starts with all data objects in one cluster and iteratively divides one cluster into two smaller clusters until each data object is in its own cluster. In each iteration, the algorithm selects one cluster containing two data objects with the longest distance among the current set of clusters.

Then the algorithm moves as many data objects as possible to a new cluster from the selected cluster, thus one cluster is divided into two clusters.

## 5. Summary

Most implementations of the hierarchical algorithms are agglomerative for efficiency reasons. Let  $n$  be the number of data objects in a clustering task. In the agglomerative algorithm, the first step considers all possible pairs of clusters. Each cluster has only one data object initially, so the time complexity is  $\frac{n(n-1)}{2} = O(n^2)$ . In the divisive algorithm, the first step must consider all possible divisions of the entire data objects into two nonempty sets. The time complexity is  $2^{n-1} - 1 = O(2^n)$ . The agglomerative hierarchical clustering can also produce an ordering of the data objects, which may be useful for data display and information discovery.

In recent years, hierarchical clustering has been widely used in pattern recognition [161], image processing [136], document classification [154], and many other areas. Although hierarchical clustering is a useful way of exploring data, it is still very *ad hoc*. Good results are often dependent on choosing the appropriate data representation and distance matrix. As shown in previous examples, the use of different distance metrics may yield different dendrograms. Performing multiple experiments and comparing the results is recommended for achieving better clustering results.

## CHAPTER 11

### CLUSTERING: PARTITIONAL ALGORITHMS

Kevin E. Heinrich

*Department of Computer Science, University of Tennessee  
203 Claxton Complex, Knoxville, TN 37996-3450, USA  
heinrich@cs.utk.edu*

#### Overview

This chapter introduces partitional clustering and briefly outlines several partitional algorithms including squared error clustering, nearest neighbor algorithms, partitioning around medoids, and self-organizing maps. Some advantages and shortcomings of each method are mentioned, and improvements to each method are discussed when applicable. An overview of related applications is presented.

**Keywords:**  $K$ -means,  $k$ -medoids, nearest neighbor clustering, partitional clustering, pattern recognition, squared error criterion, unsupervised learning.

#### 1. Introduction

*Partitional clustering algorithms* group data into one set of clusters. Clusters are formed in an unsupervised manner based on the similarity of patterns within a data set with respect to some criterion function. Often, the aim of partitional clustering algorithms is to minimize the average variance between intra-cluster patterns while maximizing the average inter-cluster distance. Once an acceptable clustering has been produced, the data can be abstracted to reveal tendencies within the original data set.

## 2. Motivation

The *curse of dimensionality* is a common problem in data mining. As the size and complexity of data increases, algorithms intended to handle that data perform ineffectively. Such is also the case with partitional algorithms. Many popular algorithms have been developed to cluster data based on different criteria; however, as data size and complexity increases, trade-offs must be made between solution quality and computation time.

This chapter aims to introduce the reader to a small subset of partitional clustering algorithms. Namely, squared error algorithms, nearest neighbor algorithms, partitioning around medoids, and self-organizing maps will be briefly examined. Other techniques such as minimum spanning trees [66], evolutionary algorithms [1], bond energy algorithms [42], stochastic connectionist approaches [10], and fuzzy clustering methods [25] will not be covered. For general information about these and other clustering methods, the reader is directed to [66].

## 3. Partitional Clustering Algorithms

Following the terminology introduced in Sec. 3, partitional clustering algorithms attempt to cluster patterns based on some distance metric. As such, the clustering problem becomes an optimization problem with respect to that metric. In general, let  $C$  denote a partition where  $C_i$  denotes the  $i$ th cluster within that partition. The  $i$ th pattern  $x_i$  is a member of the entire data set  $X$ . Most partitional clustering algorithms assume that the number of desired clusters,  $k$ , is known *a priori*.

### 3.1. Squared Error Clustering

Given an initial partition of  $k$  clusters, *squared error clustering* algorithms attempt to repartition the data to minimize the *squared error* associated with that partition. The squared error for a clustering  $C$  of size  $k$  on a pattern set  $X$  is given by

$$e^2(X, C) = \sum_{j=1}^k \sum_{i=1}^{n_j} \|x_i^{(j)} - \mathbf{c}_j\|^2,$$

where  $j$  indicates the cluster number and  $n_j$ ,  $x_i^{(j)}$ , and  $\mathbf{c}_j$  denote the number of elements, the  $i$ th pattern in cluster  $j$ , and the centroid for cluster  $j$ , respectively [74].

Squared error methods are iterative methods that terminate based on some stopping criterion. A common stopping criterion is met when the squared error between successive iterations is below some predefined threshold. Cluster membership convergence is another common criterion, while limiting the number of maximum allowable iterations is a practical approach. Other stopping criteria or a hybrid of the aforementioned approaches are all feasible.

---

**Algorithm 11** Squared Error Clustering Algorithm

---

**Input:** Number of clusters  $k$

**Output:** Clustering  $C$

Choose an initial partition of the data into  $k$  pattern clusters;

Calculate initial  $k$  cluster centers;

**while** some stopping criterion is not met **do**

    Assign each pattern to the cluster that has the closest center;

    Calculate new cluster centers based on the new cluster assignments;

    Calculate squared error;

**end while**

---

Squared error clustering methods follow the general procedure outlined in Algorithm 11 (see also [42]). The most common squared error clustering method is the  $k$ -means clustering algorithm. The general  $k$ -means algorithm differs from the squared error clustering method only in the initialization phase;  $k$  cluster means are chosen and patterns are assigned to its closest mean. The standard cluster mean is defined as

$$c_j = \frac{1}{n_j} \sum_{i=1}^{n_j} x_i^{(j)}.$$

Since each pattern in the data set is compared against each mean, the time complexity is  $O(kn)$  per iteration, where  $n$  is the total number of patterns in the data set.  $K$ -means may converge to a locally but not globally optimal solution dependent upon the choice of the initial cluster means as demonstrated in Fig. 33.

The calculated means at each step are denoted by the black points if they do not coincide with patterns. If  $k = 3$  and the initial cluster means are chosen to be A, B, and C, then the final partition will be  $\{\{A\}, \{B, C\}, \{D, E, F, G\}\}$  (denoted by the ovals). Changing the initial means

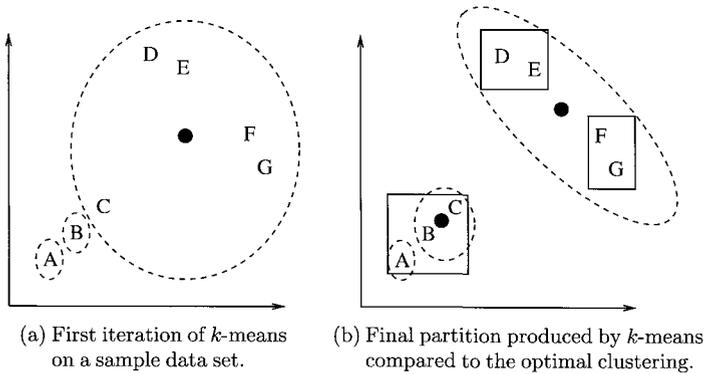


Fig. 33. Two iterations of  $k$ -means converging to a locally optimal solution for an arbitrary dataset of seven patterns.

to B, D, and F will yield the optimal partition of  $\{\{A, B, C\}, \{D, E\}, \{F, G\}\}$  (denoted by the rectangles).

Since  $k$ -means is based on centroids, it is sensitive to noise and outliers. Filtering is an obvious although impractical approach to handle this problem. [108] proposes the use of feature weighting and distortion measures to produce a partition that will minimize average within-cluster variance while simultaneously maximizing the average between-cluster distance. This method is shown to be effective but dependent upon the choice of appropriate feature weights.

There are variants of the  $k$ -means algorithm that utilize distance functions other than the Euclidean distance. For example, [158] proposes the distance function

$$d(x, y) = 1 - e^{-\beta \|x-y\|^2},$$

where  $\beta$  is a positive constant. This distance measure is shown to be robust to the outlier and noise problem. In addition to the new distance metric, alternative methods are proposed that weight patterns based on the pattern distance to its centroid [157]. In effect, outliers are downweighted with respect to the centroid, making the overall algorithm more robust to noise and outliers.

As its name implies,  $k$ -means assumes that the number of clusters  $k$  is known beforehand and that a mean can be computed. In many real world applications, the optimal value of  $k$  is not known. To overcome this, many algorithms cluster several times with several different values of  $k$ . Since means are not defined over categorical data, most  $k$ -means algorithms

assume data is not categorical. Another variation,  $k$ -modes, can be used to accommodate categorical data.

As previously shown in Fig. 33,  $k$ -means is sensitive to the initial partition choice and may converge to a locally optimal solution instead of a globally optimal one. To help alleviate this, many initialization methods have been proposed. The *Random Approach (RA)* is the most common initialization method, where the data set is partitioned into  $k$  clusters chosen at random. The *Forgy Approach (FA)* chooses  $k$  patterns from the database at random to represent cluster means and assigns the remaining patterns to the cluster represented by the closest mean. Similar to the FA, the *Macqueen Approach (MA)* assigns the remaining patterns in some predefined order. After each pattern is assigned to a cluster, that cluster centroid is recalculated. A more complex initialization procedure is the *Kaufman Approach (KA)*, which successively selects  $k$  representative instances of the data set. The complete KA algorithm can be found in [121]. The same authors performed an empirical study on these popular initialization methods and concluded that KA and RA are the two most effective methods for  $k$ -means in terms of both robustness and convergence speed, with KA slightly outperforming RA.

[98] proposes a globally optimal  $k$ -means algorithm and several variants that are invariant to the initial partition. This algorithm finds the optimal centroid for one cluster. Using this information, the “optimal” partition for  $k = m$  is computed by performing  $n$  runs of  $k$ -means with initial seeds  $x_i$  for  $i = 1, \dots, n$  and the centroids of the previous iteration. This iterative method exhaustively finds partitions of size less than  $k$  to find a globally optimal solution to the  $k$ -means problem. This result has been shown to yield experimentally optimal results, but it has not yet been proven to be theoretically optimal.

[74] proposes another improvement upon  $k$ -means called the *ISODATA algorithm*. The ISODATA algorithm permits the merging and splitting of clusters depending on the distance between cluster means and the variance within a cluster. Two clusters can be merged if their centroids are sufficiently close, and a cluster can be split if the variance of the patterns within it are over a predefined threshold. For example, given the poor choice of initial partition, the final cluster {B, C} in Fig. 33(b) would be merged with {A} while {D, E, F, G} would be split to produce the optimal partition. The ISODATA algorithm is more likely than  $k$ -means to find a globally optimal solution regardless of the choice of initial partition; however, that likelihood is dependent upon the choice of threshold values.

### 3.2. Nearest Neighbor Clustering

Nearest neighbor clustering algorithms use the properties of distance metrics introduced in Sec. 3 to form clusters. A *nearest neighbor* of  $x \in X$  is a pattern  $a \in X$  such that for all  $b \in X$  where  $a \neq b$ ,

$$d(x, a) \leq d(x, b),$$

where  $d(a, b)$  denotes the distance between patterns  $a$  and  $b$ . Likewise, a *u-nearest-neighbor set*  $N_u(x) = \{a_1, a_2, \dots, a_u\}$  must satisfy

$$d(x, a_i) \leq d(x, b),$$

for all  $b \in X - N_u(x)$ . Nearest neighbor clustering occurs in the steps outlined in Algorithm 12 [42].

---

#### Algorithm 12 Nearest Neighbor Clustering Algorithm

---

**Input:** Threshold  $t$

**Output:** Clustering  $C$

$k \leftarrow 1$ ;

Select  $x \in X$  and assign  $x$  to cluster  $C_1$ ;

**while** all patterns have not been clustered **do**

  Select an unassigned pattern  $a \in X$ ;

  Find a clustered pattern  $x \in C_j$  where  $d(a, x)$  is the smallest;

**if**  $d(a, x) \leq t$  **then**

    Assign  $a$  to  $C_j$ ;

**else**

    Increment  $k$  and assign  $a$  to the new cluster  $C_k$ ;

**end if**

**end while**

---

One variation to this algorithm ensures that a cluster remains a *u-nearest neighbor set* when adding a new pattern to it. That is, ensure that a new pattern is within the threshold distance to all elements within the cluster rather than just one pattern. Such a variation, however, may be subject to the order in which patterns are considered, as demonstrated in Fig. 34.

If A is the first pattern clustered, then the final clusters produced can be either  $\{\{A, B\}, \{C, D\}\}$  (denoted by solid ovals) or  $\{\{A, D\}, \{B, C\}\}$  (denoted by dashed ovals) depending on which pattern is considered next.

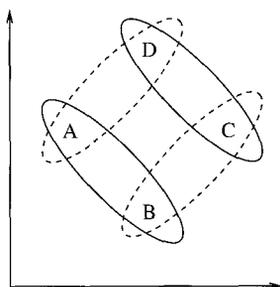


Fig. 34. Nearest neighbor may be sensitive to the order in which patterns are considered. Assume that  $d(A, B) = d(A, D) = d(B, C) = d(C, D) < t < d(A, C) = d(B, D)$ .

If the aforementioned nearest neighbor algorithm that does not consider  $u$ -nearest neighbors is followed, only one final cluster will be produced containing all four patterns.

Many storage structures have been proposed for nearest neighbor clustering. Quad trees,  $kd$ -trees,  $R$ -trees,  $R^*$ -trees,  $X$ -trees, and  $M$ -trees are several common structures that address the nearest neighbor clustering problem. The reader is directed to [1] for further discussion about these techniques. Assuming the data can be represented in high-dimensional vector spaces, most storage structures partition the pattern space into regions. If there is a possibility that a desired pattern is in a particular region, then the entire region must be searched. As a result, searching through these tree structures can be inefficient. Unfortunately, no universally efficient storage solution has been found yet, since all current methods degenerate as the number of dimensions increases.

### 3.3. Partitioning Around Medoids

Similar to the  $k$ -means algorithm, the *PAM* (*partitioning around medoids*) or *k-medoids* algorithm iteratively forms clusters around predefined medoids. A *medoid* can be thought of a centermost point within a given cluster, or as a median in higher-dimensional space. Rather than computing new means, PAM attempts to improve the clustering by replacing existing medoids with other patterns based on a cost function. The *total cost* of swapping medoid  $x_i$  with non-medoid  $x_h$  is given by

$$\mathcal{TC}_{ih} = \sum_{j=1}^n \mathcal{C}_{jih},$$

where  $\mathcal{C}_{jih}$  denotes the cost change for pattern  $x_j$  associated with the medoid swap. [42] defines *cost* as “the change to the sum of all distances from items to their cluster medoids” and further suggests the PAM algorithm requires the steps given in Algorithm 13.

---

**Algorithm 13** Partitioning Around Medoids

---

**Input:** Number of clusters  $k$

**Output:** Clustering  $\mathcal{C}$

Select  $k$  medoids from  $X$ ;

**repeat**

    Calculate  $\mathcal{TC}_{ih}$  for each non-medoid  $x_h$  and each medoid  $x_i$ ;

    Choose  $i, h$  where  $\mathcal{TC}_{ih}$  is smallest;

**if**  $\mathcal{TC}_{ih} < 0$  **then**

        Replace medoid  $x_i$  with  $x_h$ ;

**end if**

**until** no swap has occurred in the current iteration;

Assign each non-medoid  $x_i$  to its closest medoid;

---

To demonstrate the application of PAM, an example from [42] will be examined. The data in Table 11 represents a distance matrix on a simple data set. If the initial medoids chosen are  $A$  and  $B$ , then the clusters formed about them are  $\{A, C, D\}$  and  $\{B, E\}$ , where  $C$  and  $E$  are assigned arbitrarily since they are equidistant to both medoids.

The cost associated with changing a medoid with every other pattern in the data set must be examined. For example, to find  $\mathcal{TC}_{BE}$ , the total cost of swapping medoid  $B$  with medoid  $E$ , clusters must be formed around hypothetical medoids  $A$  and  $E$ . The resulting clusters are  $\{A, B, C, D\}$  and  $\{E\}$ . The cost change for each pattern must be examined based on the

Table 11. Distances between patterns in a sample data set.

Item	$A$	$B$	$C$	$D$	$E$
$A$	0	1	2	2	3
$B$	1	0	2	4	3
$C$	2	2	0	1	5
$D$	2	4	1	0	3
$E$	3	3	5	3	0

old clustering and the hypothetical clustering.  $A$  remains a medoid, so no change occurs.  $B$  loses its medoid assignment and incurs the 1 unit cost to  $A$ .  $C$  and  $D$  remain the same distance to the medoid  $A$ , so no cost change is incurred.  $E$  becomes a medoid, so its former cost of 3 is now set to zero. Thus, the total cost of swapping medoid  $B$  for medoid  $E$  is

$$\begin{aligned} \mathcal{TC}_{BE} &= \mathcal{C}_{ABE} + \mathcal{C}_{BBE} + \mathcal{C}_{CBE} + \mathcal{C}_{DBE} + \mathcal{C}_{EBE} \\ &= 0 + 1 + 0 + 0 + (-3) = -2. \end{aligned}$$

Continuing the same computation for all one-swap medoid combinations, one finds that  $\mathcal{TC}_{AC} = \mathcal{TC}_{AD} = \mathcal{TC}_{BC} = \mathcal{TC}_{BD} = \mathcal{TC}_{BE} = -2$ . Since these values are all minimum with respect to the original medoids, the choice of new cluster medoids is arbitrary. In fact, the clusters produced by PAM are dependent upon the arbitrary choice of medoids since, in this case, all subsequent iterations of PAM will produce non-negative total cost change values.

If the initial choice of medoids is  $D$  and  $E$ , PAM will converge to similar clusters in one iteration. The number of possible medoid swaps, however, is limited to swapping  $E$  with either  $A$  or  $B$ . This behavior suggests that PAM is not as sensitive as  $k$ -means to the initial partition choice.

Although PAM handles outliers well, the computation of  $\mathcal{TC}$  is a quadratic cost in the data set size that makes PAM an infeasible algorithm on large data sets. Two PAM-based methods have been developed for use specifically with large data sets — CLARA and CLARANS.

*CLARA (Clustering LARge Applications)* samples the data set and applies PAM to the smaller data set to find optimal medoids for that sample. CLARA then clusters the entire data set based on the sample medoids. As with other approaches, cluster quality depends greatly upon the initial sample chosen. To help eliminate sample bias, some CLARA algorithms will extract several samples and cluster the data set based on the medoids generated by the sample that minimizes the cost function

$$\text{Cost}(M, X) = \frac{\sum_{i=1}^n d(x_i, \text{rep}(M, x_i))}{n}, \quad (36)$$

where  $M$  is the set of medoids and  $\text{rep}(M, x_i)$  returns the medoid in  $M$  closest to  $x_i$ . In general, smaller sampling sizes will result in greater efficiency at the cost of clustering quality.

*CLARANS (Clustering LARge Applications based upon raNdomized Search)* finds  $k$ -medoids by searching a graph [113]. Let  $G = (V, E)$  be

a graph. A vertex  $v \in V$  represents a set of  $k$  patterns from  $X$ , where the patterns are the selected medoids. An edge  $(u, v) \in E$  is drawn if the sets represented by  $u$  and  $v$  differ by only one object. Equivalently, an edge is drawn if  $|M_u \cap M_v| = k - 1$ , where  $M_u$  and  $M_v$  represent the sets of  $k$  elements associated with nodes  $u$  and  $v$ . By construction, each node has  $k(n - k)$  neighbors, representing a possible clustering of  $k$ -medoids, and has a cost associated with it as defined in Eq. (36). The CLARANS algorithm is shown in Algorithm 14.

CLARANS uses a serial randomized search — it searches random neighbors of a node  $M$  until a node with a lower cost has been found. If no such node exists (i.e., the current node  $M$  is better than all its neighbors), then the set of medoids represented by  $M$  is a locally optimal solution. For many applications, CLARANS outperforms CLARA in both cluster quality and execution time.

---

**Algorithm 14** CLARANS Algorithm
 

---

**Input:** Number of clusters  $k$

**Output:** *bestnode*, which represents  $k$ -medoids (from which a clustering  $C$  can be produced)

*mincost*  $\leftarrow \infty$ ;

**for**  $i = 1$  to *numlocal*

    Randomly select a node in the graph to be the current node  $M$ ;

$j \leftarrow 1$ ;

**while**  $j > \textit{maxneighbor}$

        Randomly select a neighbor  $N$  of  $M$ ;

**if**  $\textit{Cost}(N, X) < \textit{Cost}(M, X)$  **then**

$M \leftarrow N$ ; (Assign  $N$  as the current node  $M$ )

$j \leftarrow 1$ ;

**else**

$j \leftarrow j + 1$ ;

**end if**

**end while**

**if**  $\textit{Cost}(M, X) < \textit{mincost}$  **then**

$\textit{mincost} \leftarrow \textit{Cost}(M, X)$ ;

$\textit{bestnode} \leftarrow M$ ;

**end if**

**end for**

**return** *bestnode*;

---

The two common parameters to CLARANS are *numlocal* and *maxneighbor*. CLARANS will find locally optimal solutions to *numlocal* samples in an attempt to discover the globally optimal solution. On the other hand, the parameter *maxneighbor* indicates the number of neighbors to examine from a given node. If *maxneighbor* is  $k(n-k)$ , then CLARANS will perform an exhaustive search and will mimic PAM.<sup>j</sup> Similar to CLARA, if *maxneighbor* and *numlocal* increase, solutions will tend to converge to the optimum at the cost of computation time. Unfortunately, due to the random search, an optimal clustering cannot be guaranteed. For more information regarding CLARA, CLARANS and other partitional algorithms designed for large data sets, the reader is directed to [150].

### 3.4. Self-Organizing Maps

A *self-organizing map (SOM)* or *self-organizing feature map (SOFM)* is a competitive unsupervised learning approach based on artificial neural networks. An *artificial neural network (ANN)* consists of neural processing elements (neurons) that are connected through directed links. Assuming patterns consist of  $n$  features, each neuron  $i$  is assigned an  $n$ -dimensional weight vector  $m_i$ . Neurons in an ANN are adaptive in that the output of any given neuron depends on the input parameters. In general, ANNs learn and adapt to data through a training phase. The benefits of ANNs as well as more detailed information on their function is given in [78].

Since SOMs are a competitive approach, the *winner*,  $c$ , of the training iteration is defined as having the weight

$$m_c(t) = \min_i \|x(t) - m_i(t)\|,$$

where  $c$  denotes the neuron with the lowest activation. A neuron's *activation* is typically the Euclidean distance between the weight vector and the input pattern. The adaptation of the winner is achieved by

$$m_i(t+1) = m_i(t) + \alpha(t) \times h_{ci}(t) \times [x(t) - m_i(t)], \quad (37)$$

where  $\alpha$  denotes the learning rate and  $h_{ci}(t)$  is a neighborhood function. Both the learning rate and neighborhood function decrease over time. Equation (37) effectively pulls the winner and the neurons in its vicinity closer to the input pattern to achieve adaptation and eventual clustering.

---

<sup>j</sup>This assumes that the implementation guarantees that a visited node will not be revisited by the same current node.

The *Kohonen self-organizing map* is one of the most common examples of a SOM. A Kohonen map consists of one input layer, one output layer, and a two-dimensional array of nodes called the *competitive layer*. Each node in the input layer is connected to each node in the competitive layer, and all competitive nodes produce an output. After training and adaptation occurs, nodes from the competitive layer cluster toward particular patterns in the training data so that they will produce the best clusters for the actual data set with respect to the training data. For more information regarding SOMs, the reader is directed to [7, 87, 116].

#### 4. Current Applications

Partitional clustering has roots in machine learning, pattern recognition, and image analysis. Given the informative nature of the resulting clusters, partitioning has practical applications in diverse fields. [75] suggests that partitional clustering can help with medical diagnosis and prognosis from medical imaging data as well as from genomic microarray data. Other applications exist in the fields of investigative psychology and financial decision making. Studies have been performed that examine the effect of company background statistics on revenue. Numerous applications in textual clustering and information retrieval are presented by [158].

#### 5. Summary

Grouping data based on some metric has applications in many fields. Unfortunately, this problem is non-trivial when applied to large data sets with many attributes. Although partitional clustering is considered an unsupervised method, clustering large data sets requires expert knowledge about the data in order to choose the correct clustering algorithm, distance metric, and algorithm parameters. Even with a good choice of algorithm and parameters, many algorithms sacrifice solution quality for speed or *vice versa*, and the need for tractable, robust clustering algorithms becomes apparent.

## CHAPTER 12

### CLUSTERING: LARGE DATABASES

Farial Shahnaz

*Information International Associates, Inc.*  
*P.O. Box 16221, Knoxville, TN 37996, USA*  
*shahnaz@cs.utk.edu*

#### Overview

This chapter describes the application of clustering algorithms to large databases. The basic requirements for efficient and scalable clustering algorithms and the three possible approaches to fulfilling these requirements are provided. BIRCH, DBSCAN, and CURE — three major algorithms that implement the different approaches to scalable clustering — are discussed in detail.

**Keywords:** Scalability, divide-and-conquer, order-dependent, order-independent, density-reachable, BIRCH, DBSCAN, CURE.

#### 1. Introduction

Clustering is regarded as a form of classification except unlike classification, no prior knowledge of the data (i.e., training set) is available. The most commonly used methods for clustering have computational complexities of  $O(n^2)$  and are severely constrained by memory size. In other words, clustering methods require  $n$  scans for a data set of  $n$  elements, and in order to efficiently perform the passes with minimum I/O time, the entire data set needs to be present in main memory. Although this memory requirement does not pose a problem for small databases, in the case of large databases that contain millions of records, this requirement is a major problem. Such large databases require linear approaches to clustering that do not involve loading entire datasets into the main memory of a computer.

## 2. Motivation

Clustering provides a general overview of a database by constructing a measurement of similarities (or dissimilarities) for the records stored in the database. In the absence of any such similarity measures, abstracting features or common traits from the database becomes a truly daunting task; however, it is a task that is essential to the management and manipulation of any data collection. Since the traditional clustering algorithms described in the previous chapter are inadequately equipped to perform this task on large databases (i.e., data elements numbering in the millions), the need for scalable clustering techniques capable of overcoming the impasse over memory and computational complexity has become imperative.

The following sections in this chapter describe the basic requirements for a scalable clustering algorithm, the three main approaches to implementing the requirements, and the three most widely used algorithms, namely BIRCH, DBSCAN, and CURE, for clustering large databases.

## 3. Requirements for Scalable Clustering

The primary requirement for methods that can cluster large databases (i.e., scalable clustering methods), is to have linear computational complexity. It has been proposed that a successful clustering algorithm for large databases should [42]:

- (i) Require no more (preferably less) than one scan of the database.
- (ii) Have the ability to provide status and “best” answer so far during the algorithm execution. This is sometimes referred to as the ability to be online.
- (iii) Be suspendable, stoppable, and resumable.
- (iv) Be able to update the results incremental as data are added to or removed from the database.
- (v) Work with limited memory.
- (vi) Be capable of performing different techniques for scanning the database. This may include sampling.
- (vii) Process each [record] only once.

Different clustering algorithms apply different techniques to achieve a successful implementation of the mentioned requirements. These algorithms can be categorized under three major approaches to scalable clustering that are described in the following section.

## 4. Major Approaches to Scalable Clustering

Currently there are three general approaches to clustering large databases [74] — divide-and-conquer, incremental, and parallelization. For the divide-and-conquer approach, the entire data set is stored in secondary memory and subsets of the data are clustered independently using traditional algorithms. Clustering of the subsets is followed by a merging step that constructs a clustering of the entire data set. In the incremental approach to clustering, the data set is stored in secondary memory, cluster representations are stored in main memory, and each record in the data set is transferred one at a time to the main memory for clustering. The parallel method to clustering is a variation on the divide-and-conquer approach and is only used when parallelism provides faster results [77].

### 4.1. *The Divide-and-Conquer Approach*

Given a large data set, the divide-and-conquer approach creates several smaller subsets and performs one of the classical clustering techniques on each of those subsets. Assume a data set  $D$  has  $n$  records, and is stored in secondary memory such as a disk drive. The data set is then divided into  $p$  subsets, where the optimal value of  $p$  is chosen according to the clustering algorithm used. The cardinality of each of the subsets  $p_i$  is  $n/p$ . Each  $p_i$  is sequentially transferred into main memory and clustered into  $k$  groups, and a representation or sample of each of the  $k$  clusters for  $p_i$  is retained in main memory. After clustering all  $p$  subsets, there are  $kp$  sample records stored in main memory. Clustering is again performed on these  $kp$  records to create  $k$  clusters, the labels of which are then used to cluster the original data set. Figure 35 provides a visual representation of this method [74].

This clustering method assumes a somewhat high degree of homogeneity among the records of each of the  $p_i$  subsets. If the sample records used in the final merging step are not reasonably representative of the subsets, clusters generated by the divide-and-conquer approach could contain significant inaccuracies.

### 4.2. *Incremental Clustering Approach*

The underlying premise of the incremental approach is that records in a database can be assigned a cluster label on an individual basis, and once a record has been classified, no subsequent operation (performed on other records) will change its cluster label. Incremental algorithms involve the

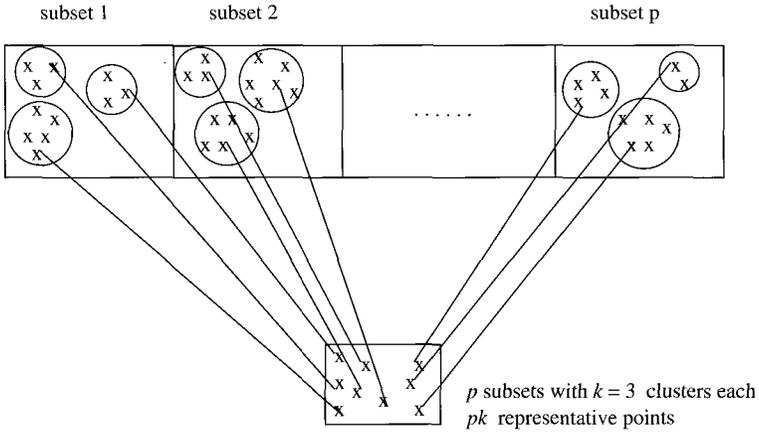


Fig. 35. Divide-and-conquer approach to clustering.

following steps [74]:

- (i) Assign the first record to a cluster.
- (ii) Read in the next record and calculate the similarity between the two records using some form of similarity measure. If the similarity measurement is within a threshold value set by the user, assign the second record to the first cluster. Otherwise, create a new cluster.
- (iii) For each subsequent record, compute the similarity between that record and the existing clusters. If the similarity measurement is within a threshold value of any of the clusters, assign it to that cluster. Otherwise, create a new cluster.

The most desirable aspect of incremental clustering algorithms is the very limited memory usage. However, this approach generates a clustering that is order-dependent, which violates one of the important requirements for a successful clustering algorithm. A clustering algorithm is *order-independent* if the same clusters are formed regardless of the order of how the database is accessed. In case of incremental algorithms, the order of fetching records from the data set significantly affects the resulting data clusters, thereby imposing an additional constraint on the database by requiring a rigid ordering of data retrieval.

### 4.3. Parallel Approach to Clustering

Parallel clustering methods take advantage of the inherently parallel nature of the divide-and-conquer approach. Given the same dataset  $D$  from the

divide-and-conquer algorithm (Fig. 35), a parallel algorithm sends each of the  $p$  subsets,  $p_i$  to a separate machine, which generates  $k$  clusters for  $p_i$  by the use of some clustering algorithm. Then the representatives of each of those  $k$  clusters are sent to a machine that performs the merging of the  $pk$  ( $p$  subsets with  $k$  clusters each) sample records and generates the final cluster labels. With the aid of enhanced clustering algorithms that create highly homogeneous clusters, the parallel clustering approach can become a very powerful method for clustering large databases.

In the following sections, the three most commonly used clustering algorithms BIRCH, DBSCAN, and CURE, are described; BIRCH and DBSCAN can be considered incremental methods of clustering, while CURE is an implementation of the divide-and-conquer approach.

## 5. BIRCH

BIRCH (Balanced Iterative Reducing and Clustering Using Hierarchies) is an incremental and hierarchical algorithm widely used to cluster large data sets that contain metric attributes. A *metric attribute* is an attribute with values that satisfy the requirements of Euclidean space, i.e., self identity and triangle inequality. This implies that BIRCH is only applicable to numeric data sets.

In [168], the clustering problem is formalized as follows. Given the desired number of clusters  $k$ , a data set  $D$  of  $n$  records with metric attributes, and a distance-based measurement function for pairs of records, partition the data set into clusters that minimize the value of the measurement function. [168] also introduces the concepts of clustering feature (CF) and clustering feature trees, both of which are very crucial components of BIRCH.

**Definition 1.** A *clustering feature* or CF is a feature summarizing the information about a cluster. Given  $n$  records or data points in a cluster, the CF is defined as a triple:  $CF = (n, LS, SS)$ ,  $LS$  is the linear sum of the  $n$  data points or the centroid of the cluster, and  $SS$  is the square sum of the  $n$  points or radius of the cluster [77].

**Definition 2.** A *CF tree* is a height-balanced tree with two parameters  $B$  (branching factor) and  $T$  (threshold value for radius). Every internal node of the tree contains at most  $B$  entries, each of which has the format  $[CF_i, child_i]$ . Here  $i = 1, 2, \dots, B$ ,  $child_i$  is a pointer to the  $i$ th child of the node, and  $CF_i$  is the cluster feature of the subcluster represented by the child. An external node, or leaf node contains at most  $L$  entries of the form  $[CF_i]$

( $i = 1, 2, \dots, L$ ), and two pointers “*prev*” and “*next*” used to chain all the leaf nodes together for effective scans. Like the internal nodes, a leaf node is also a representation of a cluster consisting of all the subclusters represented by its entries. But the entries in a leaf node must satisfy the threshold requirement.

Algorithm 15 provides the basic steps used in BIRCH to construct the CF tree [42]. The size of the tree is a function of  $T$ , and the two are inversely proportional, meaning the higher the threshold, the smaller the tree.  $T$  is determined by the page size of the computer as each tree node is required to fit into a page [168]. If this requirement is not met, the tree may have to be constructed many times in order to provide nodes that can be memory-resident, resulting in the worst-case time complexity of  $O(n^2)$ . In all other circumstances, however, BIRCH is linear in both space and I/O time [42]. BIRCH also handles the problem of outliers by removing data points found in sparsely populated areas of the problem space.

BIRCH performs well on spherical clusters of uniform size, but it does not produce good results when the clusters are of unequal size or of non-spherical shape (e.g., spiral, cylindrical, ellipsoid) [60]. To overcome these issues, a density based approach to clustering is taken in DBSCAN, which is discussed in the following section.

---

#### Algorithm 15 BIRCH

---

**Input:**

$D = \{t_1, t_2, \dots, t_n\}$  // set of elements  
 $T$  // Threshold for CF tree construction

**Output:**

$K$  // Set of clusters  
**for** each  $t_i$  in  $D$  **do**  
  determine correct leaf node for  $t_i$  intersection  
  **if** threshold condition is not violated **then**  
    add  $t_i$  to cluster and update CF triples  
  **end if**  
  **if** room to insert  $t_i$  **then**  
    insert  $t_i$  as single cluster and update CF triples  
  **end if**  
  **else** split leaf node and redistribute CF features  
**end for**

---

## 6. DBSCAN

The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) method for clustering entails creating clusters with a minimum size and density [42]. As the name suggests, this approach relies on a density-based notion of clustering and discerns clusters of arbitrary shape in spatial databases with noise. The algorithm has two input parameters — *MinPts* and *Eps*. *MinPts* is the minimum number of data points in any cluster, while *Eps* is the threshold value or maximum radius of a cluster.  $N(p)$ , the *Eps-neighborhood* or *neighborhood* of a point  $p$ , consists of all the points within *Eps* distance of  $p$ . Point  $p$  is directly *density-reachable* from another data point  $q$  if  $p$  is in  $N(q)$ , and  $|N(q)| \geq \text{MinPts}$ .  $p$  is in  $N(q)$  implies that  $q$  is “close enough” to  $p$ , and  $|N(q)| \geq \text{MinPts}$  implies that there is an adequate number of core points close enough to each other [42]. Data points that form the main portion of a cluster (i.e., those that are close to each other) are referred to as *core* points, and points that are density reachable from a core point are considered *border* points. In DBSCAN, the desired number of clusters is determined by the algorithm itself rather than being set by the user. Algorithm 16 describes DBSCAN.

---

### Algorithm 16 DBSCAN

---

**Input:**

$D = \{t_1, t_2, \dots, t_n\}$  // Set of elements

*MinPts* // Number of points in cluster

*Eps* // Maximum distance for density measure

**Output:**

$K = K_1, K_2, \dots, K_k$  // Set of clusters

$k = 0$  // Initially there are no clusters

**for**  $i = 1$  to  $n$  **do**

**if**  $t_i$  is not in a cluster **then**

$X = \{t_i | t_j \text{ is density-reachable from } t_i\}$

**end if**

**if**  $X$  is a valid cluster **then**

$k = k + 1$

$K_k = X$

**end if**

**end for**

---

The average time complexity of DBSCAN is  $O(n \lg n)$ . However, as there are no preclustering step in DBSCAN and the algorithm is performed directly on the entire data set, for large data sets DBSCAN could incur significant I/O costs. Also, in cases where two clusters are connected by a dense string of points, DBSCAN might merge the two clusters and thus reduce accuracy of the clustering. The following section includes discussion of CURE, which successfully handles arbitrary shaped clusters and has linear storage requirements and  $O(n^2)$  time complexity [60].

### 7. CURE

CURE (Clustering Using REpresentatives) is a form of the divide-and-conquer clustering algorithm that has both a hierarchical and a partitioning component [42]. The algorithm is initially performed on a sample set of the database. The basic steps for CURE include obtaining a sample set  $D$  of size  $n$  and partitioning  $D$  into  $p$  subsets, each of which is then partially clustered into  $k$  groups for some constant  $k$ . This clustering is done using a hierarchical algorithm and provides a first impression of what the clusters should be. Afterwards, outliers are eliminated by the use of two different techniques. If a cluster contains no more than one or two data points or records, that cluster is removed from the cluster list. The second method for outlier detection and removal is used near the very end of the algorithm on small clusters. Elimination of outliers is followed by application of the earlier hierarchical algorithm to  $pk$  records that represent each of the  $pk$  subsets derived from the initial clustering. The representative points are then clustered into  $k$  subsets, which are used to cluster the entire database on disk. A record in the database is placed in the cluster that has a representative point most similar to it. Figure 36 provides a visual representation of this method [60].

The time and space complexity of CURE can be reduced to  $O(n)$  with the use of heaps and  $k$ -D trees in the actual hierarchical algorithm. This algorithm is presented in Algorithm 17, but the exact algorithms for

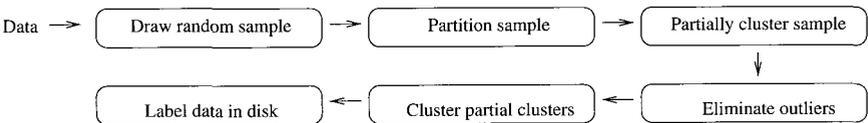


Fig. 36. Overview of CURE [60].

---

**Algorithm 17** CURE

---

**Input:** $D = \{t_1, t_2, \dots, t_n\}$  // Set of elements $k$  // desired number of clusters**Output:** $Q$  // Heap containing one entry for each cluster $T = build(D)$  $Q = heapify(D)$  // Initially build heap with one entry per item**Repeat:** $u = min(Q)$  $delete(Q, u.close)$  $w = merge(u, v)$  $delete(T, u)$  $delete(T, v)$  $insert(T, w)$ **for** each  $x$  in  $Q$  **do**     $x.close =$  find closest cluster to  $x$     **if**  $x$  is closest to  $w$  **then**         $w.close = x$          $insert(Q, w)$     **end if****end for****until** number of nodes in  $Q$  is  $k$ 

---

obtaining the sample data set or for merging the initial clusters are beyond the scope of this chapter.

## 8. Summary

Clustering as a means of unsupervised classification and knowledge abstraction is becoming an essential feature of database management systems. The pressing need for clustering extremely large databases has initiated the transformation of traditional clustering algorithms into highly efficient algorithms capable of successfully overcoming the memory constraints. Besides the three algorithms described in this chapter, several other methods such as ScaleKM/EM, MAFIA, CHAMEL, ROCK, and WaveCluster have been proposed in recent years to cluster large databases, but these methods are beyond the scope of this chapter.

This page is intentionally left blank

## CHAPTER 13

### CLUSTERING: CATEGORICAL ATTRIBUTES

Olufemi A. Omitaomu

*Department of Industrial & Information Engineering, University of Tennessee  
411 East Stadium Hall, Knoxville, TN 37996-0700, USA  
femi@utk.edu*

#### Overview

This chapter describes clustering techniques for categorical data. Clustering categorical attributes is an important task in data mining; however, it has not received much attention. Some of the popular algorithms, such as ROCK, COOLCAT, and CACTUS, are described. An example of the application of the ROCK algorithm is presented, and the results are compared with the results of a traditional algorithm for clustering numeric data.

**Keywords:** Categorical clustering, CACTUS, COOLCAT, ROCK.

#### 1. Introduction

*Clustering* is a widely used technique in which data are partitioned into groups (called clusters) based on their similarities or differences, such that data points in the same cluster are more similar among themselves than those in other clusters. An introduction to clustering techniques is presented in Chap. 9. Clustering has been used extensively for grouping numeric data. Traditional algorithms for clustering numeric data can be classified into hierarchical clustering (see Chap. 10) and partitional clustering (see Chap. 11). More recently, variants of either partitional or centroid-based hierarchical clustering algorithms have been proposed for mining large databases. A review of these algorithms is presented in Chap. 12. A review of algorithms for clustering categorical attributes is the focus of this chapter.

*Categorical attributes* are objects whose domain is not numeric, and many fields including statistics, psychology, biology, business, and engineering deal with them. An example of a categorical attribute is color, whose domain includes values such as brown, white, yellow, blue, and so on. Another example is student grades, whose domain includes A, B+, B, and so on. These values can be viewed as records with boolean attributes, each attribute corresponding to a single item. In the record for a transaction, the attribute corresponding to an item is true if and only if the transaction contains the item; otherwise, it is false. Boolean attributes are a special case of categorical attributes. The domain of categorical attributes is not limited to simply true and false values but could be any arbitrary finite set of values. It is usually more difficult to deal with attributes with categorical domains; therefore, clustering of categorical attributes has not received as much attention as its numerical counterpart.

## 2. Motivation

Categorical attributes have unique features; therefore, traditional algorithms do not always do a good job of clustering these attributes as they do with numerical data [42]. For example, hierarchical clustering algorithms may be unstable when used to cluster categorical data because “the distance between the centroid of clusters for categorical data is a poor estimate of the similarity between these data” [60]. Partitional clustering algorithms may also be unsuitable because the set of items that defines clusters may not have the same sizes, since the cluster may contain only some subset of the possible number of items. For example, a cluster involving all common items such as baby food, milk, sugar, diapers, and toys will typically involve a large number of items and customer transactions, while the cluster defined by imported items such as French beer, Swiss cheese, Italian pasta sauce, Belgian wine, and so on will be much smaller. Every transaction in the cluster may not contain all of the above items, but some subset of them. Thus, it is possible that a pair of transactions in a cluster have few items in common [61]. Therefore, there is a need for algorithms that take these features into consideration during data clustering tasks.

Three of the popular algorithms for clustering categorical attributes are reviewed in this chapter. The next section discusses the *ROCK* algorithm. The *COOLCAT* algorithm is discussed in Sec. 4, and the *CACTUS* algorithm is discussed in Sec. 5. A summary of the chapter is presented in Sec. 6.

### 3. ROCK Clustering Algorithm

The *RObust Clustering using linKs* (ROCK) clustering algorithm is a form of agglomerative hierarchical clustering algorithm. This section is based on the original paper on ROCK by Guha, Rastogi, and Shim [61]. This algorithm is based on *links* between data points, instead of distances between data points. The notion of links between data helps overcome the problems with distance based coefficients. The link between point  $i$  ( $p_i$ ) and point  $j$  ( $p_j$ ), defined as  $link(p_i, p_j)$ , is the number of common *neighbors* between  $p_i$  and  $p_j$ . A pair of points are neighbors if their similarity exceeds a certain threshold. The similarity value for pairs of points can be based on  $L_p$  distances, the Jaccard coefficient or any other non-metric similarity function. It follows that if  $link(p_i, p_j)$  is large, then it is more probable that  $p_i$  and  $p_j$  belong to the same cluster. The link-based approach adopts a global approach to the clustering problem, since it captures the global knowledge of neighboring data points into the relationship between individual pairs of points. The steps involved in clustering categorical attributes using the ROCK algorithm are [42, 61]:

- (i) Obtain a random sample from the database.
- (ii) Perform clustering on the data using the link approach (see Sec. 3.1).  
A *goodness measure*, discussed in Sec. 3.2, is used to determine which pair of points is merged at each step.
- (iii) Use these clusters to assign the remaining data points not selected in Step (i).

ROCK's hierarchical clustering algorithm accepts as input the set  $S$  of  $n$  sampled points (drawn randomly from the original data set) to be clustered, and the number of desired clusters  $k$ . The procedure begins by computing the number of links (see Sec. 3.1) between pairs of points. The number of links is then used in Algorithm 18 to cluster the dataset [61]. The first step in implementing the algorithm is to create a boolean matrix with entry 1 and 0 from the adjacency matrix. The entry is 1 if the two corresponding points are neighbors or 0 if otherwise. This is an  $O(n^2)$  step since the size of the adjacency matrix is  $n^2$ . The next step converts this boolean matrix into another matrix indicating the links. This is achieved in  $O(n^{2.37})$ , that is, calculating  $S \times S$ . The hierarchical clustering part of the ROCK algorithm starts by placing each point in the sample in a separate cluster. It successively merges clusters until  $k$  clusters are found. To enhance

---

**Algorithm 18** ROCK Algorithm

---

```

procedure cluster( $S, k$ )
  begin
   $link := compute\_links(S)$ 

  for each  $s \in S$  do
     $q[s] := build\_local\_heap(link, s)$ 
     $Q := build\_global\_heap(S, q)$ 

    while  $size(Q) > k$  do
       $u := extract\_max(Q)$ 
       $v := \max(q[u])$ 
       $delete(Q, v)$ 
       $w := merge(u, v)$ 

      for each  $x \in q[u] \cup q[v]$  do
         $link[x, w] := link[x, u] + link[x, v]$ 
         $delete(q[x], u); delete(q[x], v)$ 
      end for
       $insert(q[x], w, q(x, w)); insert(q[w], x, q(x, w))$ 
       $update(Q, x, q[x])$ 
       $insert(Q, w, q[w])$ 
       $deallocate(q[u]); deallocate(q[v])$ 
    end while
  end for
end

```

---

this processing, both local and global heaps are used. A local heap,  $q$ , is created to represent each cluster. Hence,  $q$  contains every cluster that has a nonzero link to the other cluster. The global heap,  $Q$ , contains information about each cluster. All information in this heap is ordered based on the goodness measure discussed in Sec. 3.2.

### 3.1. Computation of Links

The algorithm for computing links between every pair of points is given in Algorithm 19 [61]. After computing a list of neighbors for every point, the algorithm considers all pairs of neighbors and each point contributes one link for each pair. The process is repeated for every point while the link

---

**Algorithm 19** Links Algorithm

---

```

procedure compute.links( $S$ )
begin
  Compute  $nbrlist[i]$  for every point  $i$  in  $S$ 
  Set  $link[i, j]$  to be zero for all  $i, j$ 

  for  $i := 1$  to  $n$  do
     $N := nbrlist[i]$ 

    for  $j := 1$  to  $|N| - 1$  do

      for  $l := j + 1$  to  $|N|$  do
         $link[N(j), N(l)] := link[N(j), N(l)] + 1$ 
      end for
    end for
  end for
end

```

---

count is increased for each pair of neighbors to obtain the total link counts for all points.

**3.2. Goodness Measure**

To determine the best pair of clusters, the ROCK algorithm uses a measure in which the best clusters are those that result in the highest values for the *criterion function*. The criterion function in this case is called *goodness measure*. For a pair of clusters  $C_i, C_j$ , let  $link[C_i, C_j]$  store the number of cross links between clusters  $C_i$  and  $C_j$ , and  $n_i$  and  $n_j$  are the number of points in each cluster. Then, the goodness measure  $g(C_i, C_j)$  for merging clusters  $C_i, C_j$  is defined as [61]

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}},$$

where  $n_i^{1+2f(\theta)}$  is an estimate for the number of links between pairs of points in  $C_i$  and the threshold used for the measure is  $\theta$ . The function  $f(\theta)$  is a function of the data, and it satisfies the property that each item in  $C_i$  has approximately  $n_i^{f(\theta)}$  neighbors in the cluster. Therefore, if all points in the cluster are connected  $f(\theta) = 1$  and  $n_i^3$  is the number of links between points in  $C_i$ . The pair of clusters for which the above goodness

measure is maximum is the best pair of clusters to be merged at any given step. A large number of cross links is an indication of good candidates to merge [42, 61].

### 3.3. *Miscellaneous Issues*

Some of the issues of concern when implementing the ROCK algorithm include random sampling and outliers.

In the case of a large database, random sampling enables ROCK to reduce the number of points to be considered and ensures that the input data set fits in the main memory. Random sampling can aid clustering by filtering outliers [61] using techniques such as the one proposed by [145]. In addition, determining the appropriate random sample size enhances the quality of the clustering. One method of achieving this appropriate size is proposed in [60].

The ROCK algorithm handles outliers in two parts. The most significant part is choosing the right  $\theta$  value. This relatively discards outliers from the rest of the points and immediately helps to isolate points with very few or no neighbors from participating in the clustering. However, in some situations, outliers may be present as small groups that are loosely connected to the rest of the dataset. In this situation, these data will participate in the clustering because they are closer to the major clusters; however, they can be prevented by stopping the clustering and weeding out the clusters that have very little support [61].

### 3.4. *Example*

This example compares *Traditional Hierarchical Clustering Algorithm* (THCA) to the ROCK clustering algorithm. This example is adopted from [61]. The dataset used, according to [61], was obtained from the UCI Machine Learning Repository. It contains the United States Congressional Voting Records for the year 1984. Each record contains a Congressman's votes on 16 issues. All attributes are boolean ("yes" or "no"), with a few of the votes containing missing values. Some congressmen "crossed" parties to vote. There are 435 records in the set (267 Democrats and 168 Republicans).

The results obtained for centroid-based THCA and ROCK with the threshold ( $\theta$ ) set to 0.73 are summarized in Table 12 [61]. Both algorithms identify two clusters, each with a large number of either Republicans or Democrats. Nevertheless, there is about 25 percent Democrat votes in

Table 12. Clustering result for congressional voting data [61].

Algorithm Type	Cluster No.	No. of Republicans	No. of Democrats
THCA	1	157	52
THCA	2	11	215
ROCK	1	144	22
ROCK	2	5	201

the Republican cluster using the traditional algorithm, while this is only about 12 percent using ROCK. Records with missing data were not used in the clustering task. The improvement in the quality of clustering can be attributed to the usage of links by ROCK [61]. These results indicate that the ROCK algorithm performs better than the traditional hierarchical algorithm when used to cluster categorical data.

#### 4. COOLCAT Clustering Algorithm

*COOLCAT* is an entropy-based algorithm for clustering categorical attributes. Named because the entropy of the clusters is reduced, thereby cooling them, this algorithm uses the notion of entropy to group records and, more importantly, does not rely on arbitrary distance metrics. Entropy has been defined as the degree of disorder in a system. For example, a classroom with disorganized chairs and tables has more entropy than a classroom in which chairs and tables are well arranged. If  $X$  is a random variable,  $S(X)$  the set of values that  $X$  can take, and  $p(X)$  the probability function of  $X$ , then the entropy  $E(X)$  is defined as:

$$E(X) = - \sum_{X \in S(X)} p(X) \log(p(X)).$$

This section is based on the original paper about this algorithm by Barbará, Li, and Couto [35]. The COOLCAT algorithm consists of two steps: initialization and the incremental step. The initialization step *bootstraps* the algorithm by finding a suitable set of clusters out of a sample  $S$ , taken from the data set ( $|S| \ll N$ ), where  $N$  is the size of the entire data set. The first step is to find  $k$ , defined as the most “dissimilar” records from the sample set. This is achieved by maximizing the minimum pairwise entropy of the chosen points. To do this, two points  $p_{s_1}, p_{s_2}$  that maximize  $E(p_{s_1}, p_{s_2})$  are found and placed in two separate clusters ( $C_1, C_2$ ). The algorithm proceeds incrementally to find the record to place in  $j$ th cluster by

choosing an unmarked point  $p_{s_j}$  that maximizes

$$\min_{i=1, \dots, j-1} (E(p_{s_i}, p_{s_j})).$$

The remaining unmarked sample points ( $|S| - k$ ), and the points outside the sample, are then placed in the clusters using the incremental step.

The incremental step finds a suitable cluster for each record by computing the expected entropy that results from placing the point in each of the clusters and selecting the cluster for which that expected entropy is the minimum. The first batch of points are those that were not selected during the initialization step. The order in which the points are processed is of importance because a point that appears to be a good fit for a cluster using a particular order of process may become a poor fit as more points are clustered using another order of process. To minimize this problem, a re-processing capability of a fraction of the points in the batch is introduced into the algorithm. After each batch of points is clustered, a fraction  $m$  of worst-fit points for each cluster is selected and re-clustered. The number of occurrences for each of attributes' values in a particular cluster is used to determine the goodness of the fit. For example, if each record in the batch is  $i$ , each attribute is  $j$ , and  $q_{ij}$  represent the number of times that the value  $V_{ij}$  appears in the cluster where  $i$  was placed then  $p_{ij}$  is  $q_{ij}$  divided by the cluster size. Hence, for each record a fitting probability,  $p_i$ , is computed, where  $p_i = \prod_j (q_{ij})$ . The lower the fitting probability, the less likely the record would be a good fit. The computed fitting probabilities are sorted and  $m$  records in the batch with the lowest  $p_i$  are selected for processing. Each reprocessed record is placed in the cluster that minimizes the expected entropy. The incremental step algorithm is presented in Algorithm 20 [35].

---

**Algorithm 20** COOLCAT — Incremental Step Algorithm

---

**Given** an initial set of clusters  $\check{C} = C_1, \dots, C_k$

**Bring** points to memory from disk and for each point  $p$  do

**for**  $i = 1, \dots, k$  **do**

**Place**  $p$  in  $C_i$  and compute  $\bar{E}(\check{C}^i)$

    where  $\check{C}^i$  denotes the clustering obtained by placing  $p$  in cluster  $C - i$

**Let**  $j = \text{argmin}_i(\bar{E}(\check{C}^i))$

**Place**  $p$  in  $C_j$

**end for**

Until all points have been placed in some cluster

---

## 5. CACTUS Clustering Algorithm

*Categorical Clustering Using Summaries* (CACTUS) is a summarization-based algorithm for clustering categorical data. This section is based on the paper by Ganti, Gehrke, and Ramakrishnan [55]. The central idea behind CACTUS is that “a summary of the entire data set is sufficient to compute a set of candidate clusters that can then be validated to determine the actual set of clusters.” CACTUS consists of three phases: *summarization*, *clustering*, and *validation*. In the summarization phase, summary information is computed from the data set. In the clustering phase, the summary information is used to discover a set of candidate clusters. In the validation phase, the actual set of clusters is determined from the set of candidate clusters.

The summary information used to describe the CACTUS algorithm is of two types: *inter-attribute* summaries and *intra-attribute* summaries. The inter-attribute summaries consist of all strongly-connected attribute value pairs where each pair has attribute values from different attributes; the intra-attribute summaries consist of similarities between attribute values of the same attribute.

Once the summary information is obtained, the output is used by the clustering algorithm to compute candidate clusters in the data. In the first step of the two-step phase, each attribute is analyzed to compute all cluster-projections ( $DS_i^j$ ) on it. In the second step, candidate clusters on sets of attributes from the cluster-projections on individual attributes are synthesized in a level-wise manner. That is, candidate clusters are determined by a pair of attributes. This is then extended to a set of three attributes, four attributes, and so on. The pseudocode for the computation of cluster-projections is shown in Algorithm 21.

After the cluster-projections have been determined and candidate clusters identified, the next phase is to compute the set of actual clusters. Some of the candidate clusters may not have enough support because the clusters that combine to form a candidate cluster may be due to different sets of tuples. To recognize such false candidates, the support of each candidate cluster is compared against the required threshold. Only clusters whose support on  $D$  passes the threshold requirement are retained. Once the support of all candidate cluster has been set to zero, the scanning of the data set  $D$  begins. At the end of the scan, all candidate clusters whose support in the data set is less than the required threshold are deleted.

---

**Algorithm 21** CACTUS Algorithm —  $\text{Extend}(DS_i^j, \sum_{ij})$ 


---

**Output:**  $DS_i^j$ **Initialization**

$$DS_i^j = 0$$

Reset the subset flags and the participation counts of all distinguishing sets in  $DS_i^j$  to zero

**for each**  $S_i \in DS_i^j$ **if** the subset flag of  $S_i$  is not set **then**Extend  $S_i$  to  $C_i^S$ 

Set the subset flags and increment by the sibling strength of  $S_i$  the participation counts of all subsets of  $C_i^S$  in  $DS_i^j$ .

**end if****end for****Identify** and add small cluster-projections (of size  $\leq k$ ) to  $DS_i^j$ 

## 6. Summary

The ROCK algorithm employs links rather than distances for merging clusters. This feature demonstrates that ROCK can be used in situations where a domain expert or similarity table is the only source of knowledge. This feature has also been demonstrated to be useful for clustering time-series data [61]. However, ROCK is difficult to fine-tune (that is, finding the right  $\theta$ ). Therefore, ROCK may be more suitable for smaller data sets. The COOLCAT algorithm clusters categorical data by minimizing the expected entropy of the clusters. COOLCAT's only parameter ( $m$ ) is usually stable: "small values of  $m$  are sufficient to obtain a good result" [35]. This feature makes COOLCAT useful for larger data sets. CACTUS is a summarization-based algorithm. It finds intuitively meaningful clusters from the data set using summary information incrementally. However, for this method to perform better, the inter-attribute and intra-attribute summaries must fit in the main memory [55]. Like ROCK, this algorithm may be more suitable for smaller data sets.

## CHAPTER 14

### ASSOCIATION RULES: AN OVERVIEW

Geoffrey Mazeroff

*Department of Computer Science, University of Tennessee  
203 Claxton Complex, Knoxville, TN 37996-3450, USA  
mazeroff@cs.utk.edu*

#### Overview

This chapter introduces the concept of association rules, a form of local-pattern discovery in an unsupervised learning system. Association rules are used to uncover relationships between inherently unrelated data items. The terminology, notation, and the processes used with association rules are discussed, and a brief overview of three basic rule inference algorithms is given.

**Keywords:** *Apriori*, association rule, confidence, large itemset, market basket, negative border, partitioning, support.

#### 1. Introduction

Association rules were first introduced by Agrawal *et al.* in 1993 as a means of determining relationships among a set of items in a database [3]. Association rules, like clustering, are a form of unsupervised learning and have been applied to many fields such as retail business, Web mining, and text mining [77]. Although numerous applications of association rules involve categorical data, the rule-mining methods available to an analyst can be applied to numerical data as well.

The most challenging part of association rule inference involves finding sets of items which satisfy specific criteria, and in turn are used to infer the rules themselves. The reader should note that the literature on association rules often contains algorithms that do not infer the actual rules, but instead

creatively find the desired itemsets upon which the rules are based. This chapter presents an application of three such algorithms in the context of market baskets used in retail businesses. A *market basket* is a collection of items purchased during a single transaction [77]. The rules inferred from market baskets can allow a market analyst to design advertising campaigns that effectively match the purchasing trends indicated by the data.

## 2. Motivation

Data mining tasks typically involve the analysis of data whose inherent relationships may be obscured by the quantity of data or high dimensionality. Association rules, as a form of unsupervised learning, can be used to extract these relationships so that an analyst can make informed decisions based on the available data. Given that databases can be quite large, efficient algorithms for mining association rules are required to maximize the quality of inferred information and at the same time minimize the computation time.

This chapter covers the general framework for association rules, beginning with an introduction to terminology, notation, and the process of rule inference. A discussion of three basic algorithms — *Apriori*, sampling, and partitioning — will be presented with the aid of supplementary examples. The reader should note that formal statements are given for the algorithms discussed in this chapter.<sup>k</sup> However, textual descriptions of a given algorithm's functionality are also provided.

## 3. Association Rule Process

Association rules are used to detect common relationships between items, thus making market basket analysis a practical application of rule inference and usage. The example market basket used throughout this chapter consists of small sets of items purchased at a grocery store. A sample database of transactions will be used supplement the introduction of terminology, notation, and the process of creating association rules.

### 3.1. Terminology and Notation

The source of information for association rule algorithms is often a database viewed as a set of tuples, where each tuple contains a set of items.

---

<sup>k</sup>All algorithms presented in the chapter are adapted from [42].

Table 13. Sample database of purchase transactions.

Transaction	Items
$t_1$	Peanuts, Popcorn, Chips
$t_2$	Peanuts, Chips
$t_3$	Peanuts, Dip, Chips
$t_4$	Pretzels, Chips
$t_5$	Dip, Pretzels

Table 13 shows the contents of a sample database containing five purchase transactions.

Throughout this chapter the items listed in the examples appear in alphabetical order within a transaction because several algorithms assume the data is sorted in a preprocessing step [42]. An *association rule* is an implication that one item (or set of items) is associated with another item (or set of items). Formally, given a set of  $m$  items  $I = \{I_1, I_2, \dots, I_m\}$  and a database of  $n$  transactions  $D = \{t_1, t_2, \dots, t_n\}$ , where a given transaction contains  $k$  items  $t_i = \{I_{i1}, I_{i2}, \dots, I_{ik}\}$  and  $I_{ij} \in I$ , an association rule is an implication of the form  $X \Rightarrow Y$ , where  $X, Y \subset I$  are sets of items called *itemsets* and  $X \cap Y = \emptyset$  [77].

The *support* (denoted by variable  $s$ ) of an item or set of items is the percentage of transactions in which the item can be found. For example, {Chips} has 80 percent support because it appears in all but the last transaction, and {Pretzels, Peanuts} has no support because those two items never appear in the same transaction. Formally, the support for an association rule  $X \Rightarrow Y$  is the percentage of transactions in the database that contain  $X \cup Y$  [42]. Another numerical measure that is found with association rules is *confidence* (denoted by variable  $\alpha$ ), which essentially describes the strength of co-occurrence between two items or sets of items. The confidence for an association rule  $X \Rightarrow Y$  is the ratio of the number of transactions that contain  $X \cup Y$  to the number of transactions that contain  $X$  [42].

Support and confidence can also be represented using probabilistic notation. The support of an item or itemset  $X$  can be written as

$$P(X) = \frac{\text{number of transactions in which } X \text{ occurs}}{\text{number of transactions}}.$$

For example, **Chips** occurs in four transactions, therefore  $P(\text{Chips}) = 80\%$ . The confidence of a rule  $X \Rightarrow Y$  can be written as

$$P(X|Y) = \frac{P(X \cup Y)}{P(Y)}.$$

For example, the confidence for the association rule **Peanuts**  $\Rightarrow$  **Chips** can be expressed as

$$\begin{aligned} P(\text{Chips}|\text{Peanuts}) &= \frac{P(\{\text{Peanuts}, \text{Chips}\})}{P(\text{Chips})} \\ &= \frac{3}{4} = 75\%. \end{aligned}$$

Association rules are inferred based on support and confidence. Support refers to the percentage of occurrence of the rule in the database and confidence reflects the strength of the rule. Typically, high confidence values are used for choosing rules because the values indicate that the given rule occurs frequently in the database. For example, **Peanuts**  $\Rightarrow$  **Chips** has a confidence value of  $\alpha = 75\%$  because  $\{\text{Chips}, \text{Peanuts}\}$  occurs in three transactions ( $t_1, t_2, t_3$ ) and **Chips** occurs in four transactions ( $t_1$  through  $t_4$ ), and confidence is the ratio of the occurrences of  $X \cup Y$  to the occurrences of  $X$ . The support for this rule is  $s = 60\%$  because **Chips** and **Peanuts** occur together in three out of the five transactions. The rule **Peanuts**  $\Rightarrow$  **Chips** is much stronger than **Dip**  $\Rightarrow$  **Popcorn** because **Popcorn** and **Dip** never occur within the same transaction, giving a confidence of  $\alpha = 0\%$ . Table 14 gives some other example association rules and their corresponding support and confidence values.

From an application perspective, a market analyst would base an advertising campaign on rules with high confidence and high support. For example, placing peanuts on sale might increase the sales for chips as well. Conversely, having a sale on dip will most likely not result in increased popcorn sales.

Table 14. Support and confidence for some association rules.

$X \Rightarrow Y$	$s$	$\alpha$
<b>Peanuts</b> $\Rightarrow$ <b>Chips</b>	60%	75%
<b>Chips</b> $\Rightarrow$ <b>Peanuts</b>	60%	100%
<b>Chips</b> $\Rightarrow$ <b>Pretzels</b>	20%	50%
<b>Popcorn</b> $\Rightarrow$ <b>Peanuts</b>	20%	33.3%
<b>Peanuts</b> $\Rightarrow$ <b>Popcorn</b>	20%	100%
<b>Dip</b> $\Rightarrow$ <b>Popcorn</b>	0%	0%

### 3.2. From Data to Association Rules

The problem of mining association rules can be separated into two phases: (i) discover large itemsets, i.e., sets of items that have support  $s$  above a predetermined minimum threshold, and (ii) use large itemsets to generate the association rules for the database that have confidence  $\alpha$  above a predetermined minimum threshold [2]. This chapter uses the following notation to describe large itemsets:  $L$  indicates the complete set of large itemsets,  $l$  indicates a specific large itemset, and  $l_k$  indicates a large itemset of size  $k$ .

Determining the members of large itemsets is generally easy when using a naive approach, such as generating all possible itemsets that could appear in any transaction. However, given a set of  $m$  items, there are  $2^m - 1$  potential subsets, disregarding the empty set. The number of itemsets to generate quickly becomes impractical from a computational perspective. Association rule inference algorithms aim to reduce the number of itemsets to examine by generating *candidate itemsets*. The performance of these algorithms is often based on the number of candidate itemsets generated (denoted by the variable  $C$ ).

Association rule inference is straightforward once the large itemsets are found. Algorithm 22, adapted originally from [6], describes the steps needed to transform itemsets into association rules. To summarize in words, Algorithm 22 examines each large itemset and calculates the support value by finding the ratio of large itemset member support to individual item support. If the calculated ratio is at or above the threshold  $\alpha$ , a rule is constructed.

To illustrate rule inference, the transactions from Table 13 will be used. Suppose that the support threshold is  $s = 30\%$  and the confidence threshold is  $\alpha = 50\%$ . One can calculate the support of each possible subset of items (omitted here) to obtain a large itemset of

$$L = \{\{\text{Pretzels}\}, \{\text{Chips}\}, \{\text{Dip}\}, \{\text{Peanuts}\}, \{\text{Peanuts, Chips}\}\}.$$

Next, the ratio of large itemset support to individual item support is calculated. Suppose  $l = \{\text{Chips, Peanuts}\}$ , which contains two nonempty subsets,  $\{\text{Chips}\}$  and  $\{\text{Peanuts}\}$ . Calculating the two ratios gives

$$\frac{\text{support}(\{\text{Peanuts, Chips}\})}{\text{support}(\{\text{Chips}\})} = \frac{60}{80} = 75\%,$$

and

$$\frac{\text{support}(\{\text{Peanuts, Chips}\})}{\text{support}(\{\text{Peanuts}\})} = \frac{60}{60} = 100\%.$$

---

**Algorithm 22** Association Rule Generation
 

---

**Input:** $L$  {Large itemsets} $s$  {Support} $\alpha$  {Confidence}**Output:** $R$  {Association rules satisfying  $s$  and  $\alpha$ } $R = \emptyset$ for  $l \in L$   for  $x \subset l$  such that  $x \neq \emptyset$  do    if  $\text{support}(l)/\text{support}(x) \geq \alpha$  then       $R = R \cup \{x \Rightarrow (l - x)\}$ 

end if

end for

end for
 

---

These ratios indicate the rule **Peanuts**  $\Rightarrow$  **Chips** has 75% confidence and the rule **Chips**  $\Rightarrow$  **Peanuts** has 100% confidence. Because both of these rules have confidence values exceeding  $\alpha = 50\%$ , both rules will be added to the set of inferred rules  $R$ .

## 4. Large Itemset Discovery Algorithms

Association rule inference depends on the efficient discovery of large itemsets. Three such itemset-mining algorithms are presented in this section: *Apriori*, sampling, and partitioning. The *Apriori* algorithm generates candidate large itemsets based on inference rather than brute-force itemset creation. The sampling algorithm uses portions of the database instead of all the transactions to infer large itemsets. Lastly, the partitioning algorithm divides the database into partitions that can be mined individually to infer candidate itemsets that correspond to the entire database.

### 4.1. *Apriori*

The *Apriori* algorithm is one of the most well-known association rule algorithms [6, 42]. This algorithm uses the property that any subset of a large itemset is also large. In other words, if an itemset satisfies the minimum support requirements, so do all of its subsets. The contrapositive of the

property also applies: small itemsets have supersets that are also small. The *Apriori* algorithm works by generating candidate itemsets of a particular size. The support for the candidates is then calculated by counting the number of occurrences of a given candidate in the database. Only those candidates that are considered large are used to generate candidates for the next iteration; small candidates can be discarded because no superset containing them can be large. Candidates for the next pass ( $C_{i+1}$ ) are the result of joining large itemsets from the current pass ( $L_i$ ).

*Apriori* can be divided into two steps: (i) generate candidate itemsets, and (ii) count and select candidates. The first step is known as Apriori-Gen and is formally described by Algorithm 23.

---

**Algorithm 23** Apriori-Gen
 

---

**Input:**
 $L_{i-1}$  {Large item sets of size  $i - 1$ }
**Output:**
 $C_i$  {Candidates of size  $i$ }

 $C_i = \emptyset$ 
**for**  $I \in L_{i-1}$  **do**

   **for**  $J \neq I \in L_{i-1}$  **do**

      **if**  $i - 2$  of the elements in  $I$  and  $J$  are equal **then**

        $C_i = C_i \cup \{I \cup J\}$ 

       **end if**

     **end for**
**end for**


---

Apriori-Gen generates the candidate itemsets for each iteration of the *Apriori* algorithm except for the first pass, where the initial candidates are the singleton itemsets. The set of large itemsets from the previous pass ( $L_{i-1}$ ) is joined with itself to determine the next set of candidates.

The *Apriori* algorithm counts the occurrences of a given itemset in the database transactions and selects those candidates who meet the minimum support requirement as members of the large itemset  $L$ . See Sec. 5, Algorithm 24. *Apriori* uses Apriori-Gen to generate the candidates for the next iteration. These two steps, generating and counting/selecting, occur until no additional large itemsets can be created.

Table 15. Using *Apriori* with transactions in Table 13.

Pass	Candidates	Large Itemsets
1	{Pretzels}, {Chips}, {Popcorn}, {Dip}, {Peanuts}	{Pretzels}, {Chips}, {Dip}, {Peanuts}
2	{Pretzels, Chips}, {Pretzels, Dip}, {Pretzels, Peanuts} {Chips, Dip}, {Chips, Peanuts}, {Dip, Peanuts}	{Chips, Peanuts,}

To illustrate two passes of the *Apriori* algorithm, the transactions from the market basket given in Table 13 will be used. The candidates and large itemsets for two passes of the algorithm are given in Table 15. Suppose that  $s = 30\%$  and  $\alpha = 50\%$ . The first pass has a candidate set containing all singletons, in this case {Pretzels}, {Chips}, {Popcorn}, {Dip}, and {Peanuts}. Because {Popcorn} has a support of 20% (only occurring in one of the five transactions), it is not included as a candidate for the next pass. The second pass uses the four large itemsets to generate the next set of candidates by joining each itemset with every other itemset. Of these candidates for the second pass, only one large itemset remains, {Chips, Peanuts}, which will eventually become the associate rule  $\text{Chips} \Rightarrow \text{Peanuts}$ .

#### 4.2. Sampling

The efficiency of association rule inference can be measured by the number of database scans made during a given algorithm's execution. The number of scans can be reduced by *sampling*, where only a few transactions are selected for mining rather than the entire database [143]. The performance of sampling yields one scan in the best case and two scans in the worst case. A sample that can fit in main memory is selected from the database, then given as input to any algorithm that finds large itemsets (e.g., *Apriori*). The large itemsets returned by the algorithm are potential large itemsets (denoted as *PL*) and are used to generate candidates that are to be verified against the entire database.

Candidates are generated by applying a *negative border* function (where  $B_n(X)$  denotes finding the negative border of itemset  $X$ )<sup>1</sup> to the potentially large itemsets  $PL$ . The negative border of a set of itemsets is the minimal set of itemsets that are not in  $PL$  but have subsets that are in  $PL$ . The candidate set is the negative border combined with  $PL$ :

$$C = B_n(PL) \cup PL.$$

For example, suppose there exists a set of items  $\{S_a, S_b, S_c, S_d\}$  that has a representative set of large itemsets  $PL = \{S_a, S_c, S_d, S_{cd}\}$ .  $S_{ac}$  is in the negative border because its subsets,  $S_a$  and  $S_c$ , are in  $PL$ .  $S_{acd}$  cannot be in the negative border because neither  $S_{ac}$  nor  $S_{ad}$  is in  $PL$ . The negative border acts as a buffer area between those itemsets known to be large and the remaining itemsets. It represents the smallest possible set of itemsets that could be in  $L$ . The large itemset property mentioned previously guarantees that if there are no large itemsets in the negative border, there can be no large itemsets in the rest of the set. The sampling method combined with the *Apriori* algorithm as a means of finding large itemsets is given in Algorithm 25 (see Sec. 5, Algorithm 25). The small support value given as input to *Apriori* is used to discover more of the true large itemsets by preventing the exclusion of some potential candidates.

In practice, any sampling method and any algorithm for finding large itemsets can be used. The sampling algorithm works by drawing transactions from the database to serve as a sample or representative of the entire database. In the first scan, these transactions are used as input to an itemset-mining algorithm (*Apriori* in this case) to generate  $PL$ . Next,  $PL$  is combined with its negative border  $B_n(PL)$  to produce the list of large itemset candidates. In the second scan, additional candidates are generated and counted if there are additional itemsets in  $L$  that do not exist in  $PL$ . The sampling algorithm repeatedly applies the negative border function until no further candidates can be generated. Although the resulting set of candidates may be large, the algorithm guarantees that only one additional scan of the database is required.

---

<sup>1</sup>This notation is used here for clarity. The conventional notation for the negative border is  $BD^-(X)$ [2].

To illustrate sampling using the market basket example, suppose a sample of two transactions is drawn from the database:

$$D_S = \{t_1 = \{\text{Chips, Popcorn, Peanuts}\}, \\ t_2 = \{\text{Chips, Peanuts}\}\}.$$

Applying the *Apriori* algorithm to  $D_S$  gives

$$PL = \{\{\text{Chips}\}, \{\text{Popcorn}\}, \{\text{Peanuts}\}, \{\text{Chips, Popcorn}\}, \\ \{\text{Chips, Peanuts}\}, \{\text{Popcorn, Peanuts}\}, \\ \{\text{Chips, Popcorn, Peanuts}\}\}.$$

Applying the negative border yields

$$B_n(PL) = \{\{\text{Pretzels}\}, \{\text{Dip}\}\}$$

because **Pretzels** and **Dip** are large in the entire database (both having  $s = 40\%$ ) and are not accounted for in  $PL$ . From the initial scan, the following large itemsets are obtained:

$$L = \{\{\text{Chips}\}, \{\text{Peanuts}\}, \{\text{Chips, Peanuts}\}, \\ \{\text{Pretzels}\}, \{\text{Dip}\}\}.$$

Because **Pretzels** and **Dip** are missing from  $PL$ , a second scan of the database is needed. After three applications of the negative border function, the resultant candidate set is

$$C = \{\{\text{Pretzels, Chips, Dip, Peanuts}\}\}.$$

If a smaller support value (e.g.,  $s = 10\%$ ) were used,  $PL$  would be much larger for the first pass, thus yielding more potentially large itemsets [42].

### 4.3. Partitioning

Another approach to generating large itemsets introduced by [135] called *partitioning* divides the database into smaller subsets of transactions. Partitioning aims to improve the performance of mining large itemsets in four ways [42]. First, partitioning algorithms can better adapt to limited main memory size such that each partition fits into memory. Second, the large itemset property can be exploited because a large itemset must be large in at least one of the partitions. Third, the partitions can serve as states of the database, thus allowing incremental generation of association rules. Finally, partitioning lends itself well to parallel and distributed design to increase overall computational efficiency.

Table 16. Partitions of the transactions in Table 13.

Partition	Transaction
$D^1$	$t_1 = \{\text{Chips, Popcorn, Peanuts}\}$
	$t_2 = \{\text{Chips, Peanuts}\}$
$D^2$	$t_3 = \{\text{Chips, Dip, Peanuts}\}$
	$t_4 = \{\text{Pretzels, Chips}\}$
	$t_5 = \{\text{Pretzels, Dip}\}$

The partition algorithm works by splitting the database into  $p$  partitions,  $D^1, D^2, \dots, D^p$  such that each partition can reside in main memory. During the first scan of the database, the algorithm determines the large itemsets for each partition using any given itemset-mining algorithm (e.g., *Apriori*). In the second scan, the itemsets deemed large from the first scan are counted to determine if they are large across the entire database. The formal description of the partition algorithm (using the *Apriori* method for itemset-mining) is presented in Algorithm 26 (see Sec. 6, Algorithm 26).

Returning to the market basket example, the database can be arbitrarily divided into two partitions as shown in Table 16. The *Apriori* algorithm yields the following large itemsets for both partitions using  $s = 10\%$ :

$$L^1 = \{\{\text{Chips}\}, \{\text{Popcorn}\}, \{\text{Peanuts}\}, \{\text{Chips, Popcorn}\}, \\ \{\text{Chips, Peanuts}\}, \{\text{Popcorn, Peanuts}\}, \\ \{\text{Chips, Popcorn, Peanuts}\}\},$$

$$L^2 = \{\{\text{Pretzels}\}, \{\text{Chips}\}, \{\text{Dip}\}, \{\text{Peanuts}\}, \{\text{Pretzels, Chips}\}, \\ \{\text{Pretzels, Dip}\}, \{\text{Chips, Dip}\}, \{\text{Chips, Peanuts}\}, \\ \{\text{Dip, Peanuts}\}, \{\text{Chips, Dip, Peanuts}\}\}.$$

Because the two large itemsets differ in content and in cardinality, a market analyst may choose two different advertising campaigns based on the rules inferred for each partition. In other words, the first partition may correspond to a certain demographic while the second partition may correspond to another.

## 5. Summary

The itemset-mining algorithms presented in this chapter attempt to minimize the number of itemsets analyzed or the number of database scans needed to find applicable large itemsets; however, each algorithm can perform sub-optimally. First, the *Apriori* algorithm assumes that the entire

database is memory-resident, which may be problematic for situations involving large amounts of data. Also, the maximum number of database scans is one more than the size of the largest itemset, resulting in a large number of scans and slower performance. Second, the sampling algorithm can potentially generate a very large set of candidates during the second scan of the database to ensure that all large itemsets are found. It is possible that this larger set may not include the entire set of itemsets, meaning that certain items in the database are never considered as they would be by other algorithms. Third, the partitioning algorithm results depend entirely on the manner in which the database is divided. If the data is not uniformly spread throughout the partitions (e.g., seasonal data), the resultant association rules may not apply to the entire database, also known as *false candidates* [42]. Lastly, each of these algorithms typically expect the data to be alphabetically sorted. This preprocessing may affect the order in which itemsets are analyzed and in turn what rules are actually derived.

Many literature sources provide enhancements to the introductory algorithms presented here, such as [4]. The two chapters that follow discuss the application of association rule algorithms to parallel and distributed architectures, additional rule-mining algorithms, and methods to determine rule quality.

## Appendix

---

### Algorithm 24 Apriori

---

**Input:** $I$  {Itemsets} $D$  {Database of transactions} $s$  {Support}**Output:** $L$  {Large itemsets} $k = 0$  { $k$  is used as the scan number} $L = \emptyset$  $C_1 = I$ 

---

---

**Algorithm 24** (*Continued*)

---

```

repeat
   $k = k + 1$ 
   $L_k = \emptyset$ 
  for  $I_i \in C_k$  do
     $c_i = 0$  {Initial counts for each itemset are zero}
  end for
  for  $t_j \in D$  do
    for  $I_i \in C_k$  do
      if  $I_i \in t_j$  then
         $c_i = c_i + 1$ 
      end if
    end for
  end for
  for  $I_i \in C_k$  do
    if  $C_i \geq (s \times |D|)$  then
       $L_k = L_k \cup I_i$ 
    end if
  end for
   $L = L \cup L_k$ 
   $C_{k+1} = \text{Apriori-Gen}(L_k)$ 
until  $C_{k+1} = \emptyset$ 

```

---



---

**Algorithm 25** Sampling

---

**Input:**

$I$  {Itemsets}  
 $D$  {Database of transactions}  
 $s$  {Support}

**Output:**

$L$  {Large itemsets}

$D_S = \text{sample drawn from } D$   
 $PL = \text{Apriori}(I, D_S, \text{small } s)$   
 $C = PL \cup B_n(PL)$   
 $L = \emptyset$

---

---

**Algorithm 25** (*Continued*)
 

---

```

for  $I_i \in C$ 
   $c_i = 0$  {Initial counts for each itemset are zero}
  end for
  for  $t_j \in D$  do
    for  $I_i \in C$  do
      if  $I_i \in t_j$  then
         $c_i = c_i + 1$ 
      end if
    end for
  end for
  for  $I_i \in C$  do
    if  $c_i \geq (s \times |D|)$  then
       $L = L \cup I_i$ 
    end if
  end for
   $ML = \{x \mid x \in B_n(PL) \text{ and } x \in L\}$  {Missing large itemsets}
  if  $ML \neq \emptyset$ 
     $C = L$  {Set candidates to be the large itemsets}
  repeat
     $C = C \cup B_n(C)$  {Expand candidate sets using negative border}
  until no new itemsets are added to  $C$ 
  for  $I_i \in C$  do
     $c_i = 0$  {Initial counts for each itemset are zero}
  end for
  for  $t_j \in D$  do
    for  $I_i \in C$  do
      if  $I_i \in t_j$  then
         $c_i = c_i + 1$ 
      end if
    end for
  end for
  if  $c_i \geq (s \times |D|)$  then
     $L = L \cup I_i$ 
  end if
end if

```

---

---

**Algorithm 26** Partition

---

**Input:** $I$  {Itemsets} $D = \{D^1, D^2, \dots, D^p\}$  {Database transactions divided into partitions} $s$  {Support}**Output:** $L$  {Large itemsets} $C = \emptyset$ 

{Find large itemsets in each partition}

**for**  $i = 1$  to  $p$  **do**     $L^i = \text{Apriori}(I, D^i, s)$      $C = C \cup L^i$ **end for**     $L = \emptyset$     **for**  $I_i \in C$  **do**         $c_i = 0$  {Initial counts for each itemset are zero}    **end for**    **for**  $t_j \in D$  **do**        **for**  $I_i \in C$  **do**            **if**  $I_i \in t_j$  **then**                 $c_i = c_i + 1$             **end if**        **end for**    **end for**    **for**  $I_i \in C$     **if**  $c_i \geq (s \times |D|)$  **then**         $L = L \cup I_i$     **end if****end for**

---

This page is intentionally left blank

## CHAPTER 15

### ASSOCIATION RULES: PARALLEL AND DISTRIBUTED ALGORITHMS

Shakhina Pulatova

*National Instruments*

*11500 N. Mopac Expressway, Austin, TX 78759-3504, USA*

*shakhina@gmail.com*

#### Overview

Mining association rules from databases with extremely large numbers of transactions requires massive amount of computation. Efficient parallelization of association rule mining is particularly important for scalability. Some of the data and task parallel algorithms for both distributed and shared memory systems are reviewed in this chapter. A discussion and comparison of these algorithms in terms of computation, communication, synchronization, and memory usage is provided.

**Keywords:** Association rules, parallel algorithms, parallel data mining, parallel association mining, distributed association mining, scalability.

#### 1. Introduction

As one of the most important problems in data mining and knowledge discovery, association rule mining attempts to discover strong associations among items from databases having a large number of transaction records. Besides market basket analysis, association rules have been successfully applied to domains such as decision support, financial forecasting, telecommunication alarm prediction, medical diagnosis, customer segmentation and catalog design [26, 120, 165].

The task of finding association rules can be decomposed into two sub-problems [28, 119, 165]. First, all *large itemsets* that are contained in a sufficient number of transactions above the *minimum support* requirement

must be identified. Second, strong association rules having *minimum confidence* are generated from large itemsets.<sup>m</sup> Efficiently determining large itemsets is often viewed as a challenging problem and a significant amount of research has been conducted to devise algorithms to accomplish it. Basic algorithms including *Apriori* were introduced in Chap. 6.

Computing large itemsets is both computationally and I/O intensive since it involves scanning the database and identifying all candidate itemsets for each transaction [28]. Moreover, scalability becomes an issue as the number of items and transactions increases in the database. Since sequential algorithms are not scalable, high performance parallel and distributed computing becomes essential for efficient generation of association rules [165]. Efficient methods for distributed mining of association rules have been proposed in the literature and some of the parallel algorithms are presented in this chapter. For a more comprehensive survey of parallel and distributed association mining algorithms, the reader is directed to [43] and [165].

## 2. Motivation

Association rule mining has attracted a growing amount of attention due to its wide applicability in the retail industry by improving marketing strategy [119]. Databases in these areas often contain very large numbers of transactions, thus requiring massive computational power. In addition, many databases are physically distributed. Efficient distributed and parallel algorithms thus become particularly important for mining association rules. Distributed systems in these situations can provide an ideal platform with their high scalability, flexibility, ease of connectivity and low cost performance ratio. Furthermore, advances in this field may benefit other areas of parallel data mining as well [119].

The primary focus of this chapter is to introduce and discuss the parallel algorithms that are currently available. Some of the data and task parallel algorithms on both distributed and shared memory systems are reviewed in the next section. A discussion of these algorithms in terms of communication, computation, memory usage, and synchronization is presented in Sec. 4, followed by a summary in Sec. 5.

---

<sup>m</sup>Refer to Chap. 6 for definitions of large itemset, support, and confidence.

### 3. Parallel and Distributed Algorithms

Parallel and distributed algorithms for mining association rules can be classified in terms of three main components [165]: (i) distributed versus shared memory systems,<sup>n</sup> (ii) data versus task parallelism, and (iii) static versus dynamic load balancing.

In a *distributed (shared-nothing) memory* architecture each processor has a private memory and a message passing mechanism (such as MPI<sup>o</sup>) needs to be employed for exchanging data between processors. In a *shared memory* architecture, in contrast, all processors can access a common memory [165]. In a distributed memory system, communication between the processors is crucial to accomplish tasks. Parallel algorithms geared toward these systems are often concerned with partitioning the candidate sets to processor memories, reducing communication, and pruning candidate sets [28]. In a shared memory system, however, communication cost is no longer an issue, since processors communicate through shared variables. Instead, performance is determined by I/O and computation costs. Unfortunately, I/O can become a bottleneck due to the access of different processes via the same I/O channel. Also I/O contention may result due to a synchronized access of processors. On the other hand, with its large aggregated memory, a shared memory processor is good for mining association rules that need large storage for storing intermediate values [28].

*Data and task parallelism* paradigms split according to the distribution of candidate sets across the processors. While each processor counts the same set of candidates in a data parallel algorithm, each processor in a task parallel algorithm counts a different set of distributed candidates [43]. With data parallelism, the database is partitioned among the processors and with task parallelism, each processor has or needs access to the entire database [165].

Parallel algorithms can further be classified as having a *static or dynamic load balancing*. Static load balancing refers to the initial and final partitioning of the database among processors according to some heuristic cost function. In contrast, dynamic load balancing refers to the environment where data is continuously moved from heavily loaded processors to less busy ones. Current association rule mining algorithms all employ

---

<sup>n</sup>Hierarchical systems that have both distributed and shared memory components (e.g., a cluster of SMP workstations) are not discussed here. The reader is directed to [165] for a survey of algorithms suited for these systems.

<sup>o</sup>MPI is a library specification for message passing. Refer to [9] for more information.

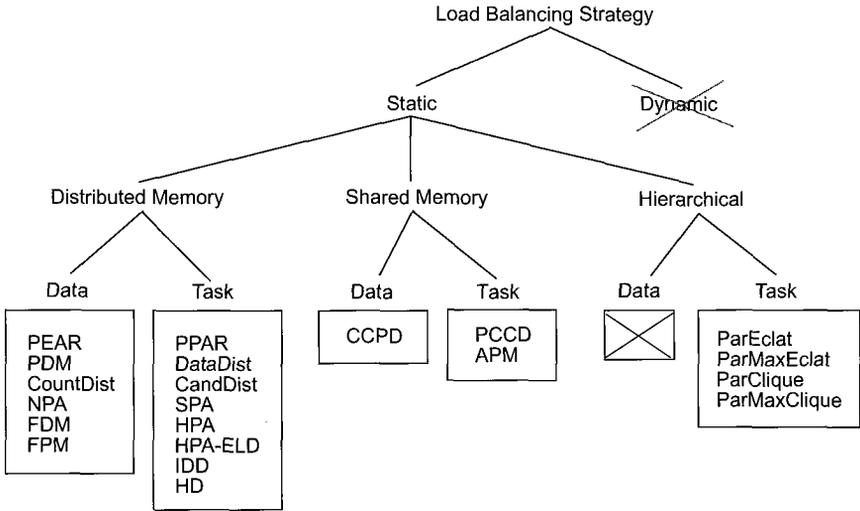


Fig. 37. Taxonomy of parallel association rule mining algorithms (figure adapted from [165]).

static load balancing, since they partition the database among processors and assume a homogeneous environment [165]. The taxonomy of different parallel and distributed algorithms according to these three components is given in Fig. 37.

**3.1. Data Parallel Algorithms on Distributed Memory Systems**

The algorithms that adopt the data parallelism paradigm on a distributed memory architecture include *Count Distribution* (CD) proposed by [5], *Parallel Data Mining* (PDM) [119], and *Distributed Mining of Association Rules* (DMA) [26]. The representative algorithm CD is described here in detail.

**3.1.1. Count Distribution (CD)**

CD algorithm is a simple parallelization of the Partitioning algorithm presented in Chap. 6. The main focus of the algorithm is to minimize communication, but it does so at the expense of performing redundant computation in parallel [5]. More specifically, the database is divided into  $p$  partitions, where  $p$  is the number of processors and the candidates are duplicated on all processors. Each processor then determines local counts for all candidates

using the support counts in its database partition. Then, by exchanging the local counts with other processors, each processor calculates the global counts that are used to determine large itemsets. Candidates for the next scan are then generated, and the whole process is repeated until no further candidates are found [42, 43].

The main advantage of CD is that no data tuples are sent from processor to processor. Instead only the counts are exchanged allowing processors to operate independently and asynchronously while reading the data. The disadvantage of the algorithm includes its inability to use aggregate memory effectively.

The algorithm is provided in Algorithm 27 in Sec. 6 and an example of its use on grocery store data (see Table 17 [42]) is illustrated in Fig. 38. There are three processors in the example, P1, P2 and P3. The first two transactions (t1 and t2) are counted at P1, the next two (t3 and t4) are counted at P2, and the last one (t5) at P3. After the local counts are obtained, they are broadcast to other processors for generation of global counts [42].

Table 17. Grocery store data [42].

Transaction	Items
t1	Bread, Jelly, Peanut Butter
t2	Bread, Peanut Butter
t3	Bread, Milk, Peanut Butter
t4	Beer, Bread
t5	Beer, Milk

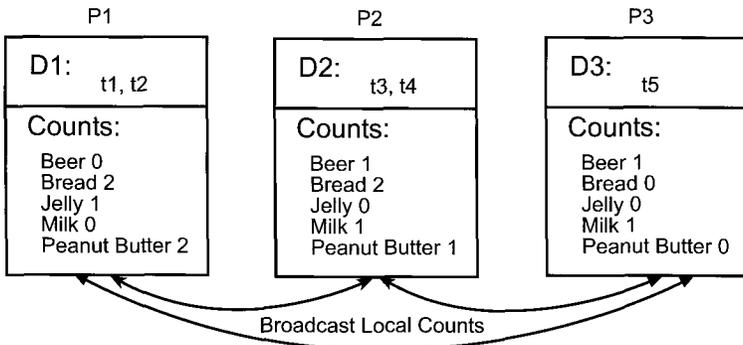


Fig. 38. Example database partitions for CD (figure adapted from [42]).

### 3.2. Task Parallel Algorithms on Distributed Memory Systems

Task parallel algorithms that are more suitable for distributed memory systems include *Data Distribution* (DD), *Candidate Distribution* (CaD) [5] and *Intelligent Data Distribution* (IDD) [62].

#### 3.2.1. Data Distribution (DD)

In contrast to the CD algorithm, the DD algorithm is designed to exploit the total aggregate memory of a machine. It distributes and partitions the candidate sets and the database to each processor, thus each processor counts mutually exclusive candidates. The disadvantage of DD is that every processor must broadcast its database partition to other processors in order to obtain global support counts for the entire database. As a result, the algorithm suffers from high communication overhead.

The details of the algorithm for DD is provided in Algorithm 28, Sec. 6 and its application for the sample data from Table 17 is illustrated in Fig. 39 [42]. In the example, processor P1 counts Beer and Bread, P2 counts Jelly and Milk, while P3 counts Peanut Butter. Initially, the first two transactions are counted at P1, the next two at P2 and the last one is counted at P3. Then, local counts are obtained at each processor and database partitions are broadcast to the other processors to obtain a global count.

#### 3.2.2. Candidate Distribution (CaD)

With CD and DD algorithms each transaction is compared against the entire candidate set. This is accomplished in CD by duplicating the entire

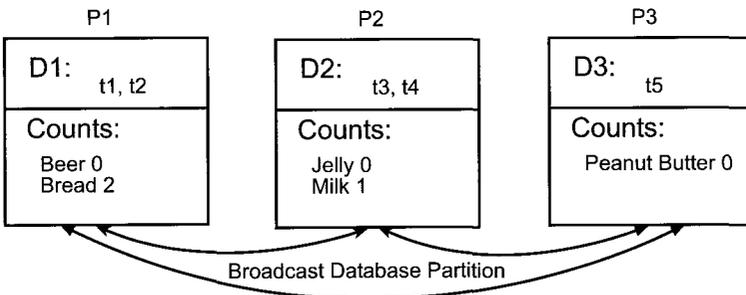


Fig. 39. Example database partitions for DD (figure adapted from [42]).

candidate set on every processor, and by broadcasting each database transaction between the processors in DD. Moreover, in both algorithms processors need to synchronize at the end of each pass through the data in order to exchange counts (CD) and candidates (DD). This requirement may lead to long wait times if the workload between the processors is not balanced [5].

The CaD algorithm in [5] attempts to solve these problems by partitioning both the data and the candidates so that processors can proceed independently. During some pass  $l$  ( $l$  is heuristically determined), the algorithm divides the large itemsets between the processors in such a way that each processor can generate a unique set of candidates. At the same time, the database is repartitioned so that each processor can count the candidates it generated independently. However, some parts of the database may have to be replicated on several processors depending upon the quality of the itemset partitioning. The itemset partitioning choice depends on the trade-off between decoupling processor dependence as soon as possible, and waiting until the itemsets become more easily partitionable.

After the candidates are partitioned, each processor proceeds with counting candidates independently, that is, no communication of counts or portions of the database is required. The only dependence between the processors involves pruning the local candidate set. This is handled by having each processor broadcast local pruning information asynchronously. If this information arrives in time, it is used by the processors; otherwise, it is saved for subsequent passes.

Thus, by incorporating domain knowledge to partition both the data and the candidates, CaD algorithm prevents the dataset from being repeatedly broadcasted. This maximizes the use of aggregate memory, limits heavy communication between processors, and eliminates the synchronization problems of CD and DD [5].

### 3.2.3. *Intelligent Data Distribution (IDD)*

The IDD algorithm is proposed by [62] as an improvement upon the DD algorithm. Specifically, it minimizes communication overhead and idle processor time, while eliminating redundant computation [62].

With IDD, redundant communication is minimized by adopting a *ring* architecture, where asynchronous point-to-point communication is used between neighbors in the ring, instead of each processor broadcasting data to all others as in DD. In order to assure a load-balanced distribution of the candidates among processors, IDD employs a partitioning algorithm based

on *bin-backing*, which assigns the items to candidate partitions. The number of candidates in each partition is roughly equal. Moreover, a candidate set is partitioned across the processors more intelligently, where candidates with the same first item are placed into the same partition, and each processor gets itemsets that begin only with the items assigned to it. This reduces the redundant computation inherent in DD, where each processor needs to consider all subsets of each transaction [43, 62, 165].

### 3.3. Data Parallel Algorithms on Shared Memory Systems

Shared memory multiprocessors are useful for association rule mining, since they have large aggregate memory. The algorithms developed for machines with this type of architecture are not concerned with communication cost due to the communication of processors via shared variables. Hence, the main objectives of algorithms targeting this architecture are to reduce computation and I/O contention caused by synchronization [28]. One of the first data parallel algorithms for shared memory multiprocessors is the *Common Candidate Partitioned Database (CCPD)* and it is described in the following subsection.

#### 3.3.1. Common Candidate Partitioned Database (CCPD)

The CCPD algorithm is an improvement of CD, which was implemented in shared memory in a 12-node SGI Power Challenge [166]. It generates candidate itemsets in parallel and stores them in a common candidate hash tree<sup>P</sup> shared by all the processors. The database is logically split among processors into equal-sized chunks. Each processor traverses its local database and generates a disjoint subset of the whole candidate set. A separate lock is associated with each leaf node of the hash tree and the processor has to acquire the lock in order to insert a candidate. Thus, different processors can insert itemsets in different parts of the hash tree in parallel. CCPD also introduces a short-circuited join method to count the candidates for each transaction efficiently and a hash tree balancing technique to speed up processing [137, 165–167].

---

<sup>P</sup>Hash trees are the most common data structure used to store the candidate itemsets and their counts. For more information about them, refer to [42].

### 3.4. Task Parallel Algorithms on Shared Memory Systems

Task parallel algorithms designed for shared memory systems include *Partitioned Candidate Common Database* (PCCD) [166] and *Asynchronous Parallel Mining* (APM) [28]. PCCD can be viewed as the opposite of the CCPD algorithm described in Sec. 3.3.1, in that it has a partitioned candidate hash tree, but a common database. The description of APM algorithm will be presented in the next section.

#### 3.4.1. Asynchronous Parallel Mining (APM)

The APM algorithm is an asynchronous and adaptive algorithm for shared memory systems developed by [28]. Compared to CD and its variants, it requires significantly fewer database scans, and generates a much smaller set of candidates. Moreover, APM produces less I/O contention than other algorithms [28]. Its candidate set generation is based on the dynamic candidate generation method in an algorithm called *Dynamic Itemset Counting* (DIC). The database is divided into intervals, and the candidates are counted on these intervals. Dynamic candidate generation is appealing because it has a potential to generate less database scanning and processors can perform counting asynchronously over their partitions. APM uses a dynamic technique together with two other optimization techniques — Adaptive interval configuration and Virtual partition pruning. Instead of a hash tree, APM uses a trie<sup>9</sup> to store the supports so that candidates of different sizes can be stored. The algorithm is presented in Algorithm 29, Sec. 6 [28].

## 4. Discussion of Parallel Algorithms

The main challenges parallel and distributed algorithms face today include communication overhead minimization, synchronization, workload balancing, data decomposition, efficient memory usage, and disk I/O minimization [165]. Various algorithms have been proposed that explore trade-offs between these issues. While the CD algorithm (based on sequential *Apriori*) minimizes communication at the expense of ignoring aggregate memory, DD algorithm exploits aggregate memory at the cost of heavy communication. The CaD algorithm attempts to reduce the synchronization and

---

<sup>9</sup>A trie is a data structure, where information about the contents of each node is stored in the path from the root to the node.

communication overhead of CD and DD by forming the hybrid of the two and by incorporating detailed problem knowledge. According to the performance evaluation on these three algorithms on a 32-node IBM SP2 in [5], CD outperformed DD and CaD. *Scaleup* (database size increased in direct proportion to the number of nodes in the system), *speedup* (number of processors increased, database kept constant) and *sizeup* (multiprocessor size fixed, database size per node increased) characteristics of the CD algorithm were examined, and CD was shown to have linear scaleup and excellent speedup and sizeup behaviors. Many subsequent algorithms use CD as a baseline algorithm to measure performance.

Further extensive analysis in [62] leads to the following results about the scalability issues of algorithms. CD does not scale well with respect to the increasing number of candidates: if the number of candidates is too large to fit in the main memory, transaction sets need to be read from the disk. Since DD performs redundant computation and has an extra cost of data movement, it does not scale well with respect to the increasing number of transactions in the database. However, DD is scalable with respect to the increasing number of candidates. Even though IDD performs no redundant work, for it to have a good load balance, the number of processors needs to be relatively small. IDD is not scalable with respect to the number of transactions, but since it does not have redundant computations, it scales better than DD. Like DD, IDD also scales with respect to the increasing number of candidates.

Moreover, plenty of experiments were run in [120] on a 12-node SGI Power Challenge (shared memory machine) on different databases that have been used as benchmarks for many association rule algorithms. The results indicated that CCPD achieved a good speedup, but serial I/O inhibited good performance.

APM was compared to CD as well as on the five series of databases in terms of response time, total number of candidate itemsets generated, CPU, and I/O costs and performance. For experiments run on a 12-node Sun Enterprise 4000 shared memory multiprocessor, APM outperformed CD in all of these areas. In the experiments conducted, APM was two to five times faster than CD, had about the half of I/O cost, and had a six to thirty times smaller candidate set. Furthermore, APM was shown to have a better scaleup (close to the ideal scaleup) and sizeup. APM was demonstrated to be consistently superior to CD despite the inherent I/O contention of shared memory architecture [28].

## 5. Summary

Discovery of association rules is an important task in data mining. Since computing large itemsets sequentially is costly in terms of I/O and CPU resources, there is a practical need for scalable parallel algorithms especially when databases are enormous and distributed. Various parallel and distributed methods have been developed that attempt to minimize communication, employ efficient computation, and synchronization techniques, and make a better usage of memory on both distributed and shared memory systems. Nevertheless, there is still room for plenty of improvement for solving issues including high dimensionality, large database sizes, data location (logically and physically distributed data), data skew, and dynamic load balancing.

In addition, most, if not all current parallel algorithms are mainly concerned with frequent itemset discovery and ignore the rule-generation phase. Although rule generation may seem as a straightforward task, it is not a trivial problem when millions of large itemsets are generated. Parallel methods are thus needed to “efficiently enumerate all strong rules” [165].

## Appendix

---

**Algorithm 27** Count Distribution [42]

---

**Input:** $I$  //Itemsets $P^1, P^2, \dots, P^p$  //Processors $D^1, D^2, \dots, D^p$  //Database divided into partitions $s$  //Support**Output:** $L$  //Large itemsets**Algorithm:**perform in parallel at each processor  $p^i$ ; //Count in parallel. $k = 0$  //  $k$  is used as the scan number. $L = \emptyset$ ; $C_1 = I$ ; //Initial candidates are set to be the items.**repeat** $k = k + 1$ ; $L_k = \emptyset$ ;

---

**Algorithm 27** (*Continued*)

---

```

for all  $I_i \in C_k$  do
   $c_i^l = 0$ ; //Initial counts for each itemset are 0
end for
for all  $t_j \in D^l$  do
  for all  $I_i \in C_k$  do
    if  $I_i \in t_j$  then
       $c_i^l = c_i^l + 1$ ;
    end if
  end for
end for
broadcast  $c_i^l$  to all other processors;
for all  $I_i \in C_k$  do
  //Determine global counts
   $c_i = \sum_{l=1}^p c_i^l$ ;
end for
for all  $I_i \in C_k$  do
  if  $c_i \geq (s \times |D^1 \cup D^2 \cup \dots \cup D^p|)$  then
     $L_k = L_k \cup I_i$ ;
  end if
end for
 $L = L \cup L_k$ ;
 $C_{k+1} = \text{Apriori-Gen}(L_k)$ 
until  $C_{k+1} = \emptyset$ ;

```

---



---

**Algorithm 28** Data Distribution [42]

---

**Input:** $I$  //Itemsets $P^1, P^2, \dots, P^p$  //Processors $D^1, D^2, \dots, D^p$  //Database divided into partitions $s$  //Support**Output:** $L$  //Large itemsets**Algorithm:** $C_1 = I$ ;

**Algorithm 28** (*Continued*)

---

```

for all  $1 \leq l \leq p$ 
  //Distribute size 1 candidates to each processor.
  determine  $C_1^l$  and distribute to  $P^l$ ;
  perform in parallel at each processor  $P^l$ ; //Count in parallel.
   $k = 0$ ; //  $k$  is used as the scan number.
   $L = \emptyset$ ;
  repeat
     $k = k + 1$ ;
     $L_k^l = \emptyset$ ;
    for all  $I_i \in C_k^l$  do
       $c_i^l = 0$ ; //Initial counts for each itemset are 0.
    end for
    for all  $t_j \in D^l$  do
      for all  $I_i \in C_k^l$  do
        if  $I_i \in t_j$  then
           $c_i^l = c_i^l + 1$  //Determine local counts.
        end if
      end for
    end for
    broadcast  $D^l$  to all other processors;
    for every other processor  $m \neq l$  do
      for all  $t_j \in D^m$  do
        for all  $I_i \in C_k^l$  do
          if  $I_i \in t_j$  then
             $c_i^l = c_i^l + 1$  //Determine global counts.
          end if
        end for
      end for
    end for
    if  $c_i \geq (s \times |D^1 \cup D^2 \cup \dots \cup D^p|)$  then
       $L_k^l = L_k^l \cup I_i$ ;
    end if
    broadcast  $L_k^l$  to all other processors;
     $L_k = L_k^1 \cup L_k^2 \cup \dots \cup L_k^p$ ; //Global large  $k$ -itemsets.
     $C_{k+1} = \text{Apriori-gen}(L_k)$ 
     $C_{k+1}^l \subset C_{k+1}$ ;
  end for
until  $C_{k+1}^l = \emptyset$ ;
end for

```

---

---

**Algorithm 29** Asynchronous Parallel Mining [28]
 

---

$I$  //Itemsets

$P^1, P^2, \dots, P^p$  //Processors

$D^1, D^2, \dots, D^p$  //Database divided into partitions

**Preprocessing:**

- (i) All processors scan their partitions to compute local supports of size-1 itemsets in their intervals;
- (ii) Compute size-1 large itemsets  $L_1$  and generate  $C_2 = \text{Apriori-gen}(L_1)$ ;
- (iii) Perform a virtual partition pruning on  $C_2$ ;
- (iv) Initialize the shared trie with the remaining size-2 candidates;
- (v) Perform inter-partition interval configuration and intra-partition interval configuration to prepare a homogeneous distribution.

//Parallel Execution:

//Each processor  $i$  runs the following fragment on its partition  $D^i$

**while** Some processor has not finished the counting of all the itemsets on the trie on its partition **do**

**while**  $P^i$  has not finished the counting of all the itemsets on the trie on  $D^i$  **do**

    Scan the next interval on  $D^i$  and count the supports of the itemsets on the trie in the interval scanned;

    Find the locally large itemsets among the itemsets on the trie;

    Generate new candidates from these locally large itemsets;

    Perform virtual partition pruning on these candidates and insert the survivors into the trie;

    Remove globally small itemsets on the trie;

**end while**

**end while**

---

## CHAPTER 16

# ASSOCIATION RULES: ADVANCED TECHNIQUES AND MEASURES

Asim YarKhan

*Department of Computer Science, University of Tennessee  
203 Claxton Complex, Knoxville, TN 37996-3450, USA  
yarkhan@cs.utk.edu*

### Overview

Association rules are a powerful tool for discovering relationships in data sets. Nevertheless, the straightforward representation of items does not allow for rules based on item categories or on quantitative information. The support-confidence measures of association rules are often not powerful enough to separate valuable insight from junk. Presented in this work are extensions to association rules that handle incremental updates, item-categories, and quantitative information. A variant of association rules that uses correlation between items is described. Advanced measures that help determine the value of association rules are shown.

**Keywords:** Generalized association rules, quantitative association rules, correlation rules, lift metric, conviction metric, chi-squared metric.

### 1. Introduction

Many of the concepts in data mining have emerged from other fields such as statistics and artificial intelligence. On the other hand, the concept of association rules originated with data mining itself and may be considered to be one of its major successes. In earlier chapters the basics of association rules have been introduced and some methods of parallelizing them have been presented. In this chapter advanced ideas relating to association rules will be discussed.

## 2. Motivation

Very often the business value from association rules derives from information that is beyond the simple view of transactions as groups of items. The actual price of the items, the general category (e.g., a hammer is a tool), the season (e.g., warm clothes in winter) and other external information may be very important.

Basic association rules can be used to generate interesting knowledge about the relationships between items from transactions. Nevertheless, it may be desirable to extend beyond the transactions using external knowledge about the items to generate the rules. In the following fictitious example, generating association rules between categories (e.g., sports equipment and womens clothing) from specific market basket items (e.g., Spalding golf balls, fluffy Angora sweaters) might lead to useful business ideas such as “advertise sports equipment on womens clothing Web sites.”

Rules that are in some sense “surprising” and reflect rare events may provide new insights into the business process, whereas common associations (with high support) can be used to do basic tasks like make product recommendations. Some measures are presented that go beyond the support-confidence framework to determine the value of a rule.

Methods of incorporating incremental updates to the data into association rules are presented in Sec. 3. Techniques to deal with hierarchies and taxonomies of items are presented in Sec. 4. Section 5 discusses one way to add quantitative information to the items. Section 6 describes how statistical correlation information can be added to association rules. Section 7 presents some measures of association rules that go beyond support-confidence.

## 3. Incremental Rules

In many data mining tasks the underlying data cannot be assumed to be static, although most data mining algorithms, such as *Apriori* (see Chap. 6), are designed for static data sets and updates to the data are handled by running the algorithm again. This is not acceptable for large data sets, or when the runtime is slow. Algorithms are needed that can handle incremental updates to the data in a graceful manner.

Fast Update (FUP) is one approach to association rules that handles incremental updates to the data [27]. The method tries to determine the promising itemsets in the incremental update to reduce the size of the candidate set to be searched in the original large database.

Given an initial large database  $DB$  with known large itemsets  $L$ , an incremental update  $db$  is to be added to it. Some previously large  $k$ -itemsets may become small in  $DB + db$ , whereas some previously small ones may become large. An itemset will be large if it is large in both  $DB$  and  $db$ , and it will be small if it is small in both. The itemsets that are large in  $DB$ , but small in  $db$  just need to have the counts updated using the counts from  $db$ . Moreover, the itemsets that are large only in  $db$  need to be checked for sufficient support, requiring a scan of the large  $DB$  to get the count. Using these updated counts, the support and confidence for the itemsets can then be computed.

#### 4. Generalized Association Rules

Generalized association rules use a concept hierarchy, i.e., building rules at different levels of the hierarchy. One possible motivation would be when there is insufficient support for a rule involving the actual items, there might be sufficient support for the ancestors of the item. A simple approach to using taxonomies is shown by Srikant and Agrawal in [140]. In this case, all ancestors of an item are added into the transaction. This naive approach is presented here, but it is computationally expensive; several improvements and variations are discussed in [140].

In Fig. 40, a simple hierarchy is shown, and a set of transactions based on the items is presented in Table 18. In Table 19, each transaction has been extended to include the hierarchy. Standard algorithms such as *Apriori* can be applied to the extended transactions to generate the association rules. Note that support for an ancestor is not simply the sum of the support for the children because several of the children could be presented in a single transaction.

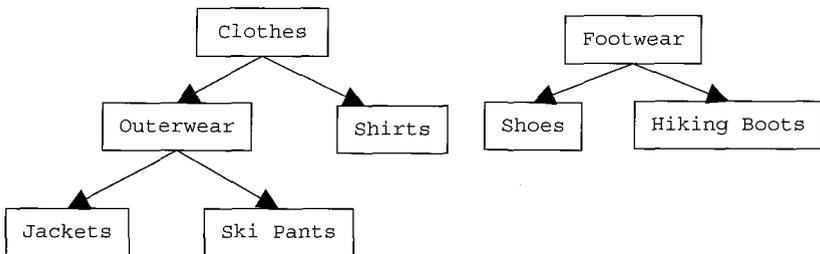


Fig. 40. Hierarchy for clothes and footwear (from [140]).

Table 18. The original transactions (from [140]).

Transaction	Items Bought
100	Shirt
200	Jacket, Hiking Boots
300	Ski Pants, Hiking Boots
400	Shoes
500	Shoes
600	Jacket

Table 19. Transactions extended with the hierarchy.

Transaction	Items Bought
100	Shirt, (Clothes)
200	Jacket, Hiking Boots, (Outerwear, Clothes, Footwear)
300	Ski Pants, Hiking Boots, (Outerwear, Clothes, Footwear)
400	Shoes, (Footwear)
500	Shoes, (Footwear)
600	Jacket, (Outerwear, Clothes)

There are many possible problems with this naive approach, for example, the support for the categories will be much higher than the support for the actual items. Thus, interesting rules based on rare items may be “drowned out” by the rules for their ancestors. Also, redundant rules may be generated (Outerwear  $\Rightarrow$  Hiking boots; Jacket  $\Rightarrow$  Hiking boots).

In [140], an *interesting rule* is defined as one whose support is other than expected. For example, consider a rule

Outerwear  $\Rightarrow$  Hiking boots (16% support, 80% confidence).

If 25% of Outerwear sales is Jackets, then the expected rule would be

Jacket  $\Rightarrow$  Hiking boots (4% support, 80% confidence).

Note that if the support for this rule is substantially different from the expected support, then this would be an interesting rule. Based on a formalization of this notion of interesting rules, the rules that are generated can be filtered to produce a more useful set.

Multiple-level association rules are a variation of generalized association rules [63, 65]. The motivation of this variation is that items at the lowest level of hierarchy (e.g., seven-grain-bread) are likely to have lower support than those at a higher level (e.g., all bread). If a minimum support value is set low to allow rules with the seven-grain-bread to emerge, then many

uninteresting rules may be generated. If the minimum support is set too high, rules involving rare items may be lost. In multiple-level association rules, different levels of the concept hierarchy can use different minimum support values.

## 5. Quantitative Association Rules

This variation on association rules involves quantitative information about the items, and was studied by Srikant and Agrawal [141]. Extending the example from Fig. 40 and Table 19, the price of a jacket could be associated with the sales of shoes or more expensive hiking boots to produce the following rules:

Jacket:[\$0 ··· \$50] ⇒ Shoes,  
 Jacket:(\$50 ··· ∞) ⇒ Hiking Boots.

This is valuable information to retailers. If they are aware that the customer is likely to buy more expensive hiking boots, they can direct the customer in that direction, rather than have the customer walk away with a cheaper pair of shoes. Table 20 shows a simple approach to including quantitative information in the rules. Break the quantity into several regions, label the items appropriately, and use the usual association rule algorithms.

Since the original item is broken into several *pseudo* items, the support for these items is less than for the original item. Therefore, the minimum support value may need to be adjusted to avoid losing useful rules. Additionally, there is no easy way to determine how to create the quantitative regions.

Table 20. Transaction hierarchy with quantitative information.

Transaction	Items Bought
100	Shirt, (Clothes)
200	Jacket:[50 ··· ∞], Hiking Boots, (Outerwear, Clothes, Footwear)
300	Ski Pants, Hiking Boots, (Outerwear, Clothes, Footwear)
400	Shoes, (Footwear)
500	Shoes, (Footwear)
600	Jacket:[50 ··· ∞], (Outerwear, Clothes)
200	Jacket:[0 ··· 50], Shoes, (Outerwear, Clothes, Footwear)

## 6. Correlation Rules

Correlation rules were proposed by Brin *et al.* [21] as a way of generalizing beyond market baskets and association rules. These mining rules use correlations, and the absence or presence of an item are both considered in generating rules. The significance of the associations is tested using the standard *chi-squared* test (presented in Sec. 7.3). Correlation rules can be used on generalized market baskets, which can be any collection of subsets of items from some space. For example, in [21], correlation rule techniques were applied to census data as well as text data from Usenet.

For random variables  $A$  and  $B$ , the correlation between  $A$  and  $B$  is defined as follows. Define the probability that  $A$  does not occur as  $P(\bar{A}) = 1 - P(A)$ , and define  $P(\bar{B})$  similarly. If  $A, B$  are independent then  $P(AB) = P(A)P(B)$ , otherwise they are dependent.  $A$  and  $B$  are said to be correlated if and only if any of  $AB, \bar{A}\bar{B}, A\bar{B}, \bar{A}B$  are dependent.

This example is adapted from [21] and shows some of the deficiencies of support and confidence. For the association rule  $A \Rightarrow B$ , define its support by  $P(A, B)$  and its confidence by  $P(B|A)$ . Suppose 90 percent of customers buy coffee, 25 percent buy tea, and 20 percent buy both tea and coffee. Then the following rule can be made:

tea  $\Rightarrow$  coffee (support = 20%, confidence =  $20/25 = 80\%$ ).

On the basis of support-confidence this seems to be a good rule; however, this ignores the knowledge that 90 percent of customers buy coffee anyway. So, knowing that the customer bought tea actually *decreases* the likelihood that they buy coffee by 10%. There is a negative relationship between buying tea and coffee that is not captured by the support-confidence framework. The correlation between tea and coffee can be calculated by  $(P(\text{tea and coffee}) / (P(\text{tea}) P(\text{coffee}))) = 0.2 / (0.25 \times 0.9) = 0.89$ , which is less than 1, so they are negatively correlated.

Correlation has a useful property for the construction of rules; it is upwardly closed. This means that if  $A, B$  are correlated, then any supersets (e.g.,  $A, B, C$ ) will also be correlated. This can intuitively be seen because adding an item to a set would not cancel out the correlations that already exist. In finding correlation rules, the minimum correlated itemsets are desired, that is, those itemsets whose subsets are not correlated.

The *upward closure* property of correlation can be contrasted with the downward closure property of support. In *downward closure*, all candidate  $(i + 1)$ -itemsets are generated from the supported  $i$ -itemsets, and then the

ones that do not have sufficient support are pruned. In upward closure, all supersets of correlated sets are known to be correlated. Support and correlation can be combined to generate *significant* itemsets, which are supported and minimally correlated.

A simplified core of the algorithm to determine significant (i.e., correlated and supported) itemsets follows. Candidate  $k$ -itemsets are constructed using the usual ideas of support. For each  $k$ -itemset, the chi-squared test (see Sec. 7.3) is used to determine if the itemset is correlated at some specified significance level. If it is correlated, then it is returned as a significant itemset. If it is not correlated, then it will be used to generate the next set of  $(k + 1)$ -itemsets.

## 7. Measuring the Quality of Association Rules

As defined in the previous section, the support and confidence for an association rule  $A \Rightarrow B$  is given by  $P(A, B)$  and  $P(B|A)$ , respectively. On the other hand, these measures for the rule leave a lot of questions, some of which have been mentioned in earlier sections. For business purposes, an obvious rule is not usually a very useful rule. So additional measures are defined that bring out other aspects of the association rules. The discussion in this section follows [42].

### 7.1. Lift

*Lift* or *interest* may be used to measure the relationship or independence of items in a rule [22]:

$$\text{lift}(A \Rightarrow B) = \frac{P(A, B)}{P(A)P(B)}.$$

Note that this measure can be seen in the definition of independence in Sec. 6. This measure is symmetric, and does not distinguish between  $A \Rightarrow B$  and  $B \Rightarrow A$  and could help judge if items are positively or negatively related. Lift would be 1 if  $A$  and  $B$  are statistically independent. This is a measure of “surprise,” in that it indicates the difference from expectations under statistical independence.

### 7.2. Conviction

From symbolic logic, it is known that  $A \rightarrow B \equiv \neg(A \wedge \neg B)$ . So,  $\frac{P(A, \neg B)}{P(A)P(\neg B)}$  would measure the negation of  $A \rightarrow B$ . To account for the negation, the

fraction is inverted to measure *conviction* [22]:

$$\text{conviction}(A \Rightarrow B) = \frac{P(A)P(\neg B)}{P(A, \neg B)}.$$

When  $A$  and  $B$  are not related, all the events are independent, and conviction will have the value of 1. Rules that are always true approach  $\infty$ . Therefore, conviction measures how strongly a rule holds.

### 7.3. Chi-Squared Test

The chi-squared test for independence takes into account both the absence and the presence of items in a set. It measures the degree to which the actual count of an itemset differs from its expected count. A contingency table for sales of coffee and tea is shown in Table 21. The table shows that 90% of the customers bought coffee, and 25% of the customers bought tea and 20% bought both tea and coffee (this continues the example in Sec. 6 taken from [22]).

The chi-squared statistic is defined as

$$\chi^2 = \sum_{X \in R} \frac{(O(X) - E[X])^2}{E[X]},$$

where  $X$  is an itemset in the rule  $R$ . For the single items  $i$  in  $X$ , the expected value  $E(i)$  is equal to the observed count  $O(i)$ . Thus  $E[t] = 25$ ,  $E[\bar{t}] = 75$ ,  $E[c] = 90$ ,  $E[\bar{c}] = 10$ . Calculate  $E[tc] = 100 \times 25/100 \times 90/100 = 22.5$ , whereas the observed  $O(tc) = 20$ . The expected value  $E[tc]$  for the itemset  $[tc]$  gives a contribution to  $\chi^2$  of  $(20 - 22.5)^2/22.5 = 0.278$ . Similarly, the contributions of  $t\bar{c}$ ,  $\bar{t}c$ , and  $\bar{t}\bar{c}$  can be calculated to get  $\chi^2 = 2.204$ . This chi-squared value can be looked up in standard statistical tables and is found to be too low at 95 percent confidence (less than 3.84). So the independent assumption cannot be rejected, and even though there appears to be a correlation, it is not statistically significant. Ultimately, the sales of coffee and tea are not correlated.

Table 21. Contingency table for coffee and tea sales.

	Coffee ( $c$ )	No Coffee ( $\bar{c}$ )	$\sum_{row}$
Tea ( $t$ )	20	5	25
No Tea ( $\bar{t}$ )	70	5	75
$\sum_{col}$	90	10	100

## 8. Summary

The techniques shown in this chapter can help overcome some of the shortcomings of the basic association rule algorithms like *Apriori* and the standard support-confidence framework. Any large, frequently-updated database should take advantage of an incremental update technique, since the cost of continually scanning the database would be too high.

Many sales databases are sparse in their counts at the UPC (Universal Product Code) or product-ID level. This is because these identifiers can change when new shipments come in, when minor updates to the product occur, etc. In these circumstances, a hierarchical or categorical technique would be vital in generating useful association rules.

One of the major problems of association rules is selecting the minimum support and confidence values. If minimum-support is too low, too many rules will be generated and they will have no value. If minimum-support is set too high, rare but interesting events might be missed. Several additional measures are presented that help select the interesting rules. Techniques for combining correlation statistics with association rules were presented that can test relationships in different ways from support-confidence.

Many other advanced techniques for association rules are not addressed in this chapter. Spatial association rules add spatial operations (e.g., near, far, next to) to describe the relationships between the data items. Temporal association rules handle the temporal variation of items, e.g., the sales of clothing varies over time (e.g., seasonal items, warm versus cool clothing). Frequent Pattern (FP) trees [64] compress the database into a tree representing frequent items, which avoids the exponential problem of candidate generation in *Apriori*.

This page is intentionally left blank

## CHAPTER 17

# SPATIAL MINING: TECHNIQUES AND ALGORITHMS

Matthew Aldridge

*Department of Computer Science, University of Tennessee  
203 Claxton Complex, Knoxville, TN 37996-3450, USA  
aldridge@cs.utk.edu*

### Overview

When attempting knowledge discovery on spatial data, certain additional constraints on and relationships among the data must be considered. These include spatially or locationally explicit attributes, as well as more implicit topological relationships. Given such additional constraints, many generalized data mining techniques and algorithms may be specially tailored for mining in spatial data. This chapter introduces several adapted techniques and algorithms that may be applied in a spatial data mining task.

**Keywords:** Spatial data mining, concept hierarchy, generalization, STING, spatial rules, ID3, two-step classification, CLARANS, GDBSCAN, DBCLASD.

### 1. Introduction and Motivation

Spatial mining is a specialized domain of data mining whose goal is to find latent or implicit knowledge in spatial data. While many general data mining techniques apply in spatial mining, additional challenges are introduced by the intrinsic nature of spatial data. Unlike other forms of data, spatial data may be viewed as objects with some location in a physical space. Spatial data also contain nonspatial attributes that may be either dependent or independent of location. Furthermore, spatial object attributes are often influenced by other nearby objects, or those contained within some notion of a neighborhood. Another challenge imposed on spatial mining is

the very large amount of spatial data often produced by remote sensing, medical equipment, geographical information systems, etc.

Given such challenges, researchers in spatial mining have adapted many general data mining techniques and algorithms for use in spatially oriented data mining tasks. This chapter discusses several such adaptations, beginning with concept hierarchies and their roles in generalizing spatial data. Rules used to describe spatial data are then introduced, followed by a hierarchical spatial mining technique based on data statistics. Finally, an overview of several spatial classification and clustering methods is given.

## 2. Concept Hierarchies and Generalization

A *concept hierarchy* is a tree structure that shows the set relationships among objects. Concept hierarchies satisfy upward consistency, in that a high-level concept is more general than but consistent with its lower-level concepts. *Generalization* is a process that employs concept hierarchies to provide descriptions of data objects at different granularities or levels of detail.

As applied to spatial data, the hierarchical levels may illustrate spatial relationships. An example spatial concept hierarchy is given in Fig. 41. Such spatial hierarchies may be generated by consolidating adjacent spatial objects. Since spatial data contain both spatial and nonspatial features, attribute hierarchies may be provided to further aid in the extraction of general knowledge from spatial databases [102]. An attribute hierarchy is given in Fig. 42, where types of agriculture production are examined. Using

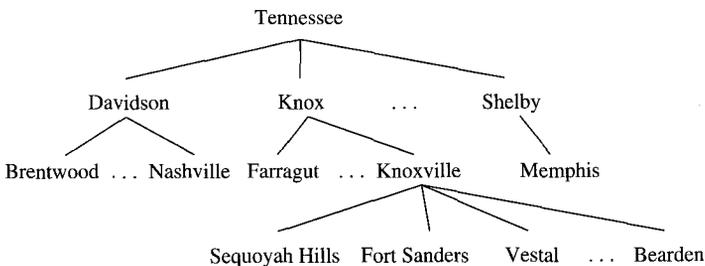


Fig. 41. A portion of a spatial concept hierarchy that describes the state of Tennessee by county, city, and community.

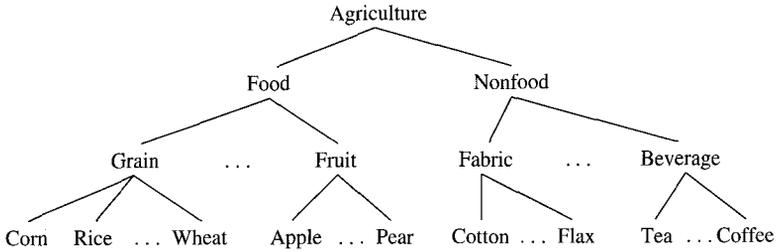


Fig. 42. An attribute concept hierarchy that partitions types of agriculture production. Figure adapted from [102].

this attribute hierarchy, a region that produces corn and wheat may be generalized to a grain-production region.

Depending upon the application, the practitioner may choose to generalize either nonspatial or spatial data first. *Nonspatial data dominant generalization* first performs generalization on nonspatial attributes using an expert-provided concept hierarchy, followed by the generalization of spatial attributes into clusters or regions. This process consists of the following steps:

- (i) Collect related nonspatial data. In the above agriculture example, the relevant nonspatial data would be the lowest-level production types for each spatial object in the spatial database.
- (ii) Perform attribute-oriented generalization on collected nonspatial data using concept hierarchy ascension. The sufficiency of a given nonspatial generalization may be determined either by the number of generalized tuples in the generalized relation or by an appropriate concept level, both of which can be provided by the user or expert explicitly.
- (iii) Perform spatial generalization. After the desired level of nonspatial generalization has been reached, spatial objects or regions are merged together with neighboring objects or regions that contain the same high-level attribute values. If an object or region is dominated by a particular attribute value, then the submissive value(s) may be dropped from the high-level description. For example, if 95 percent of a region produces corn, and 5 percent of the region produces tobacco, then the region may still be generalized to grain-producing according to the concept hierarchy in Fig. 42.

*Spatial data dominant generalization* works in a very similar manner as nonspatial data dominant generalization:

- (i) Collect related spatial data.
- (ii) Generalize spatial data by clustering spatial objects according to their regions.
- (iii) For each region, generalize nonspatial data until a small number of concepts subsume all the concepts existing in the region.

The computational complexity of both spatial and nonspatial dominant generalization on  $n$  spatial objects is  $O(n \log n)$ . Nevertheless, [102] suggests interleaved generalization between spatial and nonspatial data to achieve satisfactory results with improved performance. For example, a performance gain may be achieved if nonspatial data is generalized to a certain level before performing a potentially expensive spatial data dominant generalization.

### 3. Spatial Rules

As previously mentioned, spatial data contain both spatial and nonspatial features. For example, consider a thematic raster image depicting average yearly rainfall amounts for a region. A pixel's coordinates in the image correspond to location (spatial feature), while the pixel's color indicates rainfall amount (nonspatial feature). To better describe the nature of such data, *spatial rules* may be employed.

One type of spatial rule is the *spatial characteristic rule*, which characterizes or describes the data. This is the simplest spatial rule, as it provides no greater context for the spatial object being described. *Spatial discriminant rules* provide such missing context by describing the differences between classes or objects within the data and may be viewed as conjunctions of multiple spatial characteristic rules. Using the above rainfall raster image example, possible instances for spatial characteristic and discriminant rules are as follows:

- Characteristic rule: The average yearly rainfall at pixel  $A$  is 35 inches.
- Discriminant rule: The average yearly rainfall at pixel  $A$  is 35 inches, while the average yearly rainfall at pixel  $B$  is 39 inches.

The third type of spatial rule is the *spatial association rule*. The concept of spatial association rules is an extension of the original work on

association rules as applied to mining of large transaction databases [88].<sup>r</sup> A spatial association rule is of the form  $X \rightarrow Y(c\%)$ , where  $X$  and  $Y$  are sets of spatial or nonspatial predicates and  $(c\%)$  is the confidence of the rule. Referring again to the above rainfall example, one potential spatial association rule may be:

Eighty percent of the pixels near pixel  $C$  have average yearly rainfall levels of less than 33 inches.

Note that this example association rule contains a spatial predicate in the antecedent, while the consequent is nonspatial. One or both of the antecedent and consequent must contain a spatial predicate. A possible rule with a nonspatial antecedent and spatial consequent is:

Twenty percent of the pixels with average yearly rainfall levels above 39 inches lie south of pixel  $D$ .

As with traditional association rules, support and confidence thresholds are used to reduce the number of association rules under consideration. These thresholds may also be adjusted for different levels of a concept hierarchy. At lower levels of a hierarchy, high thresholds could prevent the discovery of interesting associations because fewer objects have a given description [89].

A two-step method for discovering spatial association rules is given in [88]. The first step involves finding the support for predicates among the objects at a coarse level of description. For example, R-trees or fast minimum bounding rectangle methods may be used to find objects that are approximately close to each other. Those predicates that pass the support threshold are given further consideration, and another round of predicate support computation is performed with finer detail and probably a lower support threshold. Finally, those predicates that pass both filtration steps are mined for possible strong association rules.

#### 4. STING

The *STatistical INformation Grid (STING)* approach to spatial data mining is introduced in [146]. STING is a hierarchical method that divides the spatial area into cells at multiple levels of granularity, much like a quad tree, with statistical information about the objects in each cell. Unlike

---

<sup>r</sup>See Chap. 14 for an introduction to association rules.

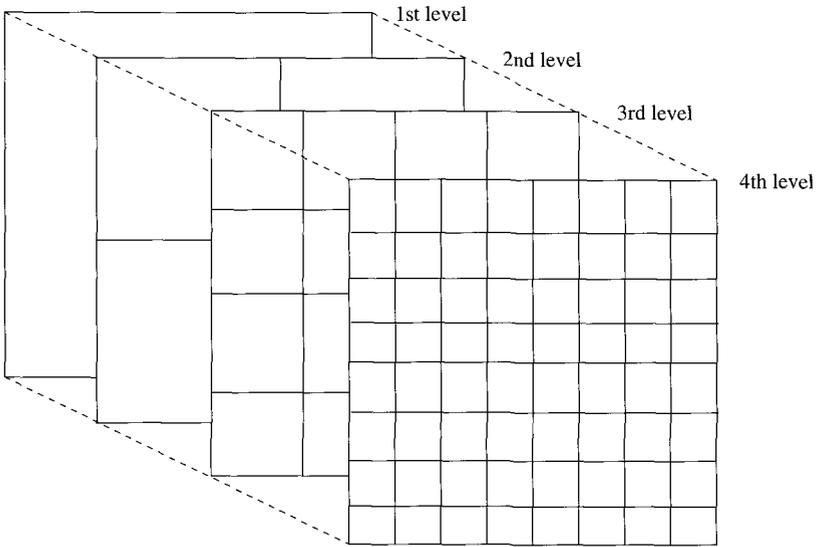


Fig. 43. An example STING grid hierarchy of four levels.

generalization techniques, STING does not require a predefined concept hierarchy. Moreover, STING is a query-independent approach since the statistical summary of the spatial region is computed only once, independently of any query.

The initial step prior to applying the STING algorithm is to create the hierarchical grid cell structure. Figure 43 illustrates this structure. The top level is a single cell containing the entire spatial region of interest. Beginning with the top cell, the region is recursively divided into quadrants until a sufficiently deep hierarchy is reached. The size of the lowest level cells is dependent upon the density of the spatial objects; [146] suggests choosing a depth such that the average number of objects in each leaf cell is in the range from several dozens to several thousands.

For each cell in the lowest level, the number of objects contained within the cell is stored, along with several object attribute-dependent parameters. It is assumed that all object attributes have numerical values. For each numerical attribute, the mean, minimum, maximum, standard deviation and distribution type of the attribute are determined.

The parameters of higher level cells may be easily computed from their child cells. This statistical summary need only be computed once for static data. Moreover, if the data is updated, only the parameters for the lowest

level cells affected and each of their ancestor cells would require recomputation, rather than the entire region. Though not discussed here, [147] proposes an extension called STING+ that handles dynamically evolving data in a more sophisticated manner by utilizing user-defined triggers.

The most common query fielded by STING is a region query, where regions satisfying certain conditions are returned. Another possible query type is one that selects a predetermined region, returning some function of the region's parameters (e.g., an attribute's range). An extension of SQL is suggested in [146] for handling STING queries, and an example query is given:

```
SELECT REGION
FROM house-map
WHERE DENSITY IN (100,  $\infty$ )
AND price RANGE (400000,  $\infty$ )
    WITH PERCENT (0.7, 1)
AND AREA (100,  $\infty$ )
AND WITH CONFIDENCE 0.9
```

This query selects with 90 percent confidence regions containing at least 100 houses per unit area, with at least 70 percent of the house prices above \$400K, and with total area at least 100 units.

The STING algorithm, shown in Algorithm 30 as adapted from [42], uses a breadth-first search of the grid hierarchy to find regions of cells that satisfy a given query. Regions of relevant cells may be generated by joining cells marked *relevant* with each neighboring cell successively and determining whether this small region satisfies the query requirements. If it does, the region is marked *relevant*, and the process of joining and testing neighboring cells of the region continues until the region cannot be increased while still maintaining relevancy. When region production is performed at multiple levels of granularity, the STING approach may be viewed as a type of hierarchical clustering. If the number of cells,  $k$ , at the lowest level of the grid hierarchy is significantly less than the number of objects in the entire region (as the STING authors suggest), then the computational complexity of a STING query is  $O(k)$ . As such, STING provides better performance than many other spatial mining approaches.

## 5. Spatial Classification

A specialization of the general classification problem introduced in previous chapters is *spatial classification*, wherein spatial objects are partitioned. To

---

**Algorithm 30** STING Algorithm
 

---

**Input:**

$T$  {Grid hierarchy tree}  
 $q$  {Query}

**Output:**

$R$  {Regions of relevant cells}

$i = 1$  {Current hierarchy level}

$c$  {Current cell}

**repeat**

**for**  $c \in T_i$  such that parent of  $c$  is *relevant* **do**

    Based on a confidence threshold, mark  $c$  as *relevant* or *irrelevant* to  
      $q$

**end for**

$i = i + 1$

**until** all levels of the tree have been visited

Populate  $R$

---

find rules for use in such partitioning, several types of characterizations of spatial objects must be considered [90]:

- Nonspatial attributes, such as “population,”
- Spatially-related attributes with nonspatial values, such as “population living within walking distance of a store,”
- Spatial predicates, such as *distance\_less\_than\_10km (source, target)*,
- Spatial functions, such as *distance\_to (source, target)*.

Each of these categories may be used both to define classes and to build the decision trees for predicting objects’ class membership. To create simpler decision trees and improve efficiency, it may be possible to aggregate some attribute values, and concept hierarchies could be utilized to generalize attributes.

### 5.1. ID3 Extension

An extension to the ID3 classification algorithm for classifying spatial data is proposed in [106].<sup>s</sup> The driving force behind this approach is that the

---

<sup>s</sup>See Chap. 7 for an introduction to the ID3 method.

implicit relationships of spatial neighborhoods impact the classification of spatial objects. To examine such relationships, the concept of *neighborhood graphs* is introduced.

The authors define a neighborhood graph  $G_{neighbor}$ , for some spatial relation or predicate *neighbor*, as a graph  $(N, E)$ , where  $N$  is a set of nodes and  $E$  a set of edges. Each node is an object in the spatial database, and two nodes  $n_1$  and  $n_2$  are connected via an edge if and only if *neighbor*( $n_1, n_2$ ) holds. The predicate *neighbor* may be based on topological, metric, or directional relationships. From the neighborhood graph, a *neighborhood path* may be defined as a list of nodes of  $G$ ,  $[n_1, n_2, \dots, n_k]$ , such that *neighbor*( $n_i, n_{i+1}$ ) holds for each  $i$ ,  $1 \leq i \leq k - 1$ .

The method then applies a modified version of ID3 for creating a decision tree that takes into consideration not only the attributes of a given object, but also the attributes of neighboring objects. Because the influence of neighboring objects and their attributes decreases with increased distance, the neighbors taken into consideration are limited to those within some maximum distance along a neighborhood path. This ID3 extension may be prone to producing overspecialized decision trees. As described in the next section, this can be avoided by using predicate relevance analysis in their two-step method [90].

## 5.2. Two-Step Method

A two-step approach for classifying spatial objects is proposed in [90]. This process is similar to the two-step method for mining association rules mentioned earlier in Sec. 3, in that rough computations are performed first and then fine computations are done only for the promising patterns.

In the first step, a training sample of the objects in the domain is chosen and examined to find coarse descriptions and relevant predicates and functions. Minimum bounding rectangles about the sample objects may be used at this stage for finding approximate distances among the objects. Then, for every sample object  $s$ , two nearest neighbors are found for  $s$ . One neighbor is the *nearest hit*, which is the closest object to  $s$  that belongs to the same class. The other is the *nearest miss*, which is the closest object to  $s$  that does not belong to the same class.

Prior to examining all sample objects and their nearest hits or misses, all predicates defined in the system are given an initial weight of zero. Then, if a nearest hit contains the same value as  $s$  for a given predicate, then the predicate's weight is increased. Otherwise, the predicate's weight

is decreased. Conversely, if a nearest miss for  $s$  contains the same predicate value, the weight decreased, while a different predicate value increases the weight. After examining all objects and their nearest hits and misses, only those predicates that have positive weights above a predetermined threshold are given further consideration. The motivation for this step is to create more efficient decision trees that do not branch on irrelevant predicates.

In the second step, the *buffers* around each sample object are examined. Determining buffer shape and size is important, as buffers represent areas that have an impact on class discrimination. In their experiments, [90] use equidistant rings around objects to define the buffers. Once the appropriate buffer shape and size have been determined, each buffer is described by aggregating the values of the most relevant predicates for objects contained within. Then, the predicates for each buffer are generalized using a predefined concept hierarchy. The information gain for each of these generalized predicates is computed in a method similar to that used in ID3. Finally, a binary decision tree is produced to classify all the spatial objects in the domain.

## 6. Spatial Clustering

As in any data mining task, clustering often plays an important role in the mining of spatial data. Nevertheless, spatial data may impose new limitations or complications not realized in other domains. Spatial databases often contain massive amounts of data, and spatial clustering algorithms must be able to handle such databases efficiently. Spatial clustering algorithms typically cluster spatial objects according to locality; for point objects, Euclidean or Manhattan metrics of dissimilarity may suffice, but for polygon objects no such intuitive notion of similarity exists. Furthermore, spatial clustering algorithms should be able to detect clusters of arbitrary shape. The following is a discussion of several clustering techniques and their ability or inability to handle some of these complications.

### 6.1. CLARANS

Introduced by [113] but revised in [114], CLARANS is a hybrid approach using both the PAM and CLARA clustering techniques.<sup>†</sup> The CLARA technique selects an initial random sample of objects in the domain before

---

<sup>†</sup>See Chap. 11 for an overview of PAM and CLARA, and for algorithmic details of CLARANS.

applying the PAM algorithm to find better cluster medoids or center points among those objects' neighbors. CLARANS, on the other hand, applies the PAM algorithm to *all* objects in the domain, but randomly samples the neighbors in the search for better cluster medoids. This approach has the added benefit of not limiting the medoid searches to a given area, as may happen with the initial CLARA random sample.

CLARANS is not limited to point objects, and the original authors propose a method for efficiently computing distance between polygon objects. An *isothetic rectangle* for an object is its minimum bounding rectangle that has edges parallel to either the  $x$ - or  $y$ -axis. By utilizing such a restricted rectangle, testing for overlap and distance among polygons is computationally much simpler. Experimental results are given in [114] that show that the isothetic approximation approach requires only about 30 to 40 percent of the time for performing exact polygon distances.

One major drawback of the CLARANS algorithm is its main-memory requirement. An entire spatial database must be loaded at once when applying CLARANS, which is not satisfactory for very large data sets. Modifications of CLARANS have been proposed to offset the effects of this problem [89]. These modifications involve the use of an R-tree variant to focus the algorithms computations on only relevant objects, thereby reducing the amount of data required to be in memory at a given point.

## 6.2. GDBSCAN

[134] introduces *GDBSCAN*, a generalized version of DBSCAN.<sup>14</sup> GDBSCAN can cluster point objects as well as polygon objects using both their spatial and nonspatial attributes. One way that GDBSCAN generalizes DBSCAN is that any notion of a neighborhood of an object may be used, as long as the definition of a neighborhood is based on a binary predicate that is symmetric and reflexive. GDBSCAN is not limited to using a simple maximum cluster radius like in DBSCAN. Further generalization is provided by the ability to use nonspatial attributes to determine the density of a neighborhood, rather than just using the number of objects in the neighborhood, as with DBSCAN.

Spatial index structures such as R-trees may be used with GDBSCAN to improve upon its memory and runtime requirements. When not using such a structure, the overall complexity of GDBSCAN applied to  $n$  objects is  $O(n^2)$ . When using an R-tree, the complexity is  $O(n \log n)$ .

---

<sup>14</sup>See Chap. 12 for an overview of DBSCAN.

Experimental results are provided by [134] for running the algorithm on data sets containing intuitive clusters of irregular shapes and sizes with obvious outliers. It is shown that GDBSCAN performs quite well in such situations. The authors also provide experimental results for running CLARANS on the same data sets. The accuracy of the clusters provided by CLARANS pale in comparison, as nearby clusters are often merged, and outliers are not handled properly.

### 6.3. DBCLASD

The creators of GDBSCAN also introduce a spatial clustering method called *DBCLASD*, or *Distribution Based Clustering of LArge Spatial Databases* [160]. The fundamental idea behind DBCLASD is the assumption that points within a given cluster are uniformly distributed. The algorithm creates clusters in a manner very similar to (G)DBSCAN. Initially, no points belong to any cluster. Each point  $p$  in the database is examined, and if  $p$  does not belong to a cluster, a new cluster containing  $p$  is created. Each neighbor  $q$  of  $p$  is also added to this cluster. Then for each  $q$ , any remaining neighbors of  $q$  not already added to the cluster are examined for the uniform distribution requirement. Those neighbors that fit the distribution of the cluster are added.

Though certainly not applicable in every clustering application, the authors provide a few examples where the uniform distribution assumption applies. One is the task of identifying minefield clusters and distinguishing those clusters from outliers, possibly representing rocks or other debris. Since mines in a minefield are typically fairly uniformly distributed, DBCLASD should perform well in this task. Experimental results are also provided for running DBCLASD on a database of California earthquakes. Though the earthquake points do not follow an exactly uniform distribution, DBCLASD provides intuitive clusters of the points.

Like GDBSCAN, DBCLASD provides accurate clusters of points arranged in irregular shapes. Empirically, the runtime of DBCLASD is shown to be roughly twice that of DBSCAN, though DBCLASD outperforms CLARANS by “a factor of at least 60” [160].

## 7. Summary

As the amount of available spatial data continues to grow, the need for efficient and effective spatial data mining techniques becomes more pressing. Though general data mining methods may be applicable in some spatially

oriented tasks, the challenges imposed by the nature of spatial data often need to be addressed in order to extract meaningful knowledge from spatial databases. Though advancements have been made on this front, spatial mining is still a young field with many available avenues for further progress.

This page is intentionally left blank

## REFERENCES

1. H. ABBASS, R. SARKER, AND C. NEWTON, *Data Mining: A Heuristic Approach*, Idea Group Publishing, London, 2002.
2. J. ADAMO, *Data Mining for Association Rules and Sequential Patterns: Sequential and Parallel Algorithms*, Springer-Verlag, New York, 2001.
3. R. AGRAWAL, T. IMIELINSKI, AND A. SWAMI, *Mining association rules between sets of items in large databases*, in Proceedings of ACM-SIGMOD Conference, Washington, DC, 1993.
4. R. AGRAWAL, H. MANNILA, R. SRIKANT, H. TOIVONEN, AND A. VERKANO, *Fast discovery of association rules*, in Advances in Knowledge Discovery and Data Mining, U. FAYYAD, G. PIATETSKY-SHAPIRO, P. SMYTH, AND R. UTHURUSAMY, eds., AAAI Press/The MIT Press, Menlo Park, CA, 1996.
5. R. AGRAWAL AND J. C. SHAFER, *Parallel mining of association rules*, IEEE Transactions on Knowledge and Data Engineering, 8 (1996), pp. 962–969.
6. R. AGRAWAL AND R. SRIKANT, *Fast algorithms for mining association rules*, in Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile, 1994.
7. N. ALLINSON, H. YIN, L. ALLINSON, AND J. SLACK, eds., *Advances in Self-Organising Maps*, Springer-Verlag, London, 2001.
8. C. APTÉ AND S. WEISS, *Data mining with decision trees and decision rules*, Future Generation Computer Systems, 13 (1997), pp. 197–210.
9. ARGONNE NATIONAL LABORATORY, *The Message Passing Interface (MPI) Standard*, 2004. [www-unix.mcs.anl.gov/mpi](http://www-unix.mcs.anl.gov/mpi), accessed May, 2004.
10. G. BABU, M. MURTY, AND S. KEERTHI, *A stochastic connectionist approach for global optimization with application to pattern clustering*, IEEE Transactions on Systems, Man, and Cybernetics — Part B: Cybernetics, 30 (2000), pp. 10–24.
11. R. BAEZA-YATES, Introduction to data structures and algorithms related to information retrieval, in *Information Retrieval: Data Structures and Algorithms*, R. BAEZA-YATES AND W. B. FRAKES, eds., Prentice Hall, Englewood, NJ, 1992.
12. R. BAEZA-YATES AND B. RIBEIRO-NETO, *Modern Information Retrieval*, Addison-Wesley, New York, 1999.
13. A. BANERJEE, S. MERUGU, I. DHILLON, AND J. GHOSH, *Clustering with Bregman divergences*, in Proceedings of the Fourth SIAM International Conference on Data Mining, 2004, pp. 234–245.

14. P. BERKHIN AND J. BECHER, *Learning simple relations: Theory and applications*, in Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, 2002, pp. 420–436.
15. M. BERRY AND M. BROWNE, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*, SIAM, Philadelphia, PA, 1999.
16. J. BILMES, *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*, International Computer Science Institute, Berkeley, CA, 1998.
17. T. BOZKAYA AND M. OZSOYOGLU, *Distance-based indexing for high-dimensional metric spaces*, in Proceedings of the 1997 ACM SIGMOD, ACM, 1997.
18. L. BREIMAN, *Bagging predictors*, Machine Learning, 24 (1996), pp. 123–140.
19. L. BREIMAN, *Arcing classifier*, Annals of Statistics, 26 (1998), pp. 801–824.
20. L. BREIMAN, J. FRIEDMAN, R. OLSHEN, AND P. STONE, *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.
21. S. BRIN, R. MOTWANI, AND C. SILVERSTEIN, *Beyond market baskets: Generalizing association rules to correlations*, in Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, vol. 26(2), ACM Press, 1997, pp. 265–276.
22. S. BRIN, R. MOTWANI, J. D. ULLMAN, AND S. TSUR, *Dynamic itemset counting and implication rules for market basket data*, in Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, vol. 26(2), 1997, pp. 255–264.
23. J. CANADA, W. SULLIVAN, AND J. WHITE, *Capital Investment Analysis for Engineering and Management*, Prentice Hall, Upper Saddle River, NJ, 2nd ed., 1996.
24. J. CENDROWSKA, *PRISM: An algorithm for inducing modular rules*, International Journal of Man-Machine Studies, 27 (1987), pp. 349–370.
25. P. CHAN AND A. RAD, *Adaptation an learning of a fuzzy system by nearest neighbor clustering*, Fuzzy Sets and Systems, 126 (2002), pp. 353–366.
26. D. CHEUNG, J. HAN, V. NG, A. FU, AND Y. FU, *A fast distributed algorithm for mining association rules*, in Proceedings of the Parallel and Distributed Information Systems Conference, 1996.
27. D. W. CHEUNG, J. HAN, V. T. NG, AND C. WONG, *Maintenance of discovered association rules in large databases: An incremental updating technique*, in Proceedings of the International Conference on Data Engineering, ICDE, 1996, pp. 106–114.
28. D. W. CHEUNG, K. HU, AND S. XIA, *An adaptive algorithm for mining association rules on shared-memory parallel machines*, Distributed and Parallel Databases, 9 (2001), pp. 99–132.
29. G. CHOWDHURY, *Introduction to Modern Information Retrieval*, Library Association, London, 1999.
30. P. CIACCIA, M. PATELLA, AND P. ZEZULA, *M-tree: An efficient access method for similarity search in metric spaces*, in Proceedings of the Twenty-Third International Conference on VLDB, Morgan Kaufmann Publishers, 1997, pp. 426–435.

31. P. CLARK AND T. NIBLETT, *The CN2 induction algorithm*, Machine Learning Journal, 3 (1988), pp. 261–283.
32. W. COHEN, *Fast effective rule induction*, in Proceedings of the Twelfth International Conference on Machine Learning, vol. 12, 1995, pp. 115–123.
33. W. COHEN AND Y. SINGER, *A simple, fast, and effective rule learner*. [www-2.cs.cmu.edu/~wcohen/postscript/aaai-99-slipper.ps](http://www-2.cs.cmu.edu/~wcohen/postscript/aaai-99-slipper.ps), accessed May, 2004, 1999.
34. T. CORMEN, C. LEISERSON, R. RIVEST, AND C. STEIN, *Introduction to Algorithms*, McGraw Hill, Cambridge, 2nd ed., 2001.
35. D. BARBARÁ, Y. LI, AND J. COUTO, *COOLCAT: An entropy-based algorithm for categorical clustering*, in Proceedings of the Eleventh International Conference on Information and Knowledge Management, McLean, VA, 2002.
36. I. DE FALCO, A. DELLA CIOPPA, AND E. TARANTINO, *Mutation-based genetic algorithm: Performance evaluation*, Applied Soft Computing, 1 (2002), pp. 285–299.
37. I. DHILLON, S. MALLELA, AND R. KUMAR, *Enhanced word clustering for hierarchical text classification*, in Proceedings of the Eighth ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, pp. 191–200.
38. J. DONGARRA, V. EIJKHOUT, E. FUENTES, J. DEMMEL, R. VUDU, AND K. YELICK, *Self adapting linear algebra algorithms and software*, in Proceedings of the IEEE, 2004. In Press.
39. R. DUBES, *Cluster analysis and related issues*, in Handbook of Pattern Recognition & Computer Vision, C. H. CHEN, L. F. PAU, AND P. S. P. WANG, eds., World Scientific Publishing, River Edge, NJ, 1993.
40. O. DUDA AND P. HART, *Pattern Recognition and Scene Analysis*, Wiley, New York, 1973.
41. R. O. DUDA, P. E. HART, AND D. G. STORK, *Pattern Classification*, Wiley, New York, 2001.
42. M. DUNHAM, *Data Mining: Introductory and Advanced Topics*, Prentice Hall, Upper Saddle River, NJ, 2003.
43. M. H. DUNHAM, Y. XIAO, L. GRUENWALD, AND Z. HOSSAIN, *A survey of association rules*, Technical Report TR 00-CSE-8, Southern Methodist University, Department of Computer Science, 2000.
44. B. EFRON AND R. TIBSHIRANI, *An Introduction to the Bootstrap*, Chapman and Hall, London, 1993.
45. L. EGGHE AND C. MICHEL, *Strong similarity measures for ordered sets of document in information retrieval*, Information Processing & Management, 38 (2002), pp. 823–848.
46. U. FAYYAD, G. PIATETSKY-SHAPIRO, AND P. SMYTH, *From data mining to knowledge discovery: An overview*, in Advances in Knowledge Discovery and Data Mining, U. FAYYAD, G. PIATETSKY-SHAPIRO, P. SMYTH, AND R. UTHURUSAMY, eds., AAAI Press/MIT Press, Menlo Park, CA, 1996.

47. V. FERNÁNDEZ, R. G. MARTÍNEZ, R. GONZÁLES, AND L. RODRIGUEZ, *Genetic algorithms applied to clustering*, in Proceedings of the IASTED International Conference: Signal and Image Processing, 1996.
48. A. FREITAS, *A survey of evolutionary algorithms for data mining and knowledge discovery*, in Advances in Evolutionary Computing, S. T. A. GHOSH, ed., Springer-Verlag, 2001.
49. A. FREITAS, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, Springer, Berlin, New York, 2002.
50. Y. FREUND AND R. SCHAPIRO, *Experiments with a new boosting algorithm*, in Proceedings of the Thirteenth International Conference on Machine Learning, 1996, pp. 148–156.
51. H. FRIEDMAN, T. HASTIE, AND R. TIBSHIRANI, *Additive logistic regression: A statistical view of boosting*, Annals of Statistics, 28 (1998), pp. 337–374.
52. T. FUNKHOUSER, P. MIN, M. KAZHDAN, J. CHEN, A. HALDERMAN, AND D. DOBKIN, *A search engine for 3D models*, ACM Transactions on Graphics, 22 (2003), pp. 83–105.
53. J. FURNKRANZ AND G. WIDMER, *Incremental reduced error pruning*, in Proceedings of the Eleventh International Conference on Machine Learning, vol. 11, 1994.
54. P. GANESAN, H. GARCIA-MOLINA, AND J. WIDOM, *Exploiting hierarchical domain structure to compute similarity*, ACM Transaction on Information Systems, 21 (2003), pp. 64–93.
55. V. GANTI, J. GEHRKE, AND R. RAMAKRISHNAN, *CACTUS: Clustering categorical data using summaries*, in Proceedings of the ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, 1999.
56. P. GIUDICI, *Applied Data Mining: Statistical Methods for Business and Industry*, John Wiley, West Sussex, England, 2003.
57. J. GOWER, *A comparison of some methods of cluster analysis*, Biometrics, 23 (1967), pp. 623–628.
58. G. GRAHNE, R. HAKLI, M. NYKÄNEN, H. TAMM, AND E. UKKONEN, *Design and implementation of a string database query language*, Information Systems, 28 (2003), pp. 311–337.
59. R. GROTH, *Data Mining: Building Competitive Advantage*, Prentice Hall, Upper Saddle River, NJ, 2000.
60. S. GUHA, R. RASTOGI, AND K. SHIM, *CURE: A clustering algorithm for large databases*, in ACM SIGMOD International Conference on Management of Data, 1998, pp. 73–84.
61. S. GUHA, R. RASTOGI, AND K. SHIM, *ROCK: A robust clustering algorithm for categorical attributes*, Information Systems, 25 (2000), pp. 345–366.
62. E. HAN, G. KARYPIS, AND V. KUMAR, *Scalable parallel data mining for association rules*, IEEE Transactions on Knowledge and Data Engineering, 12 (1996), pp. 337–352.
63. J. HAN AND Y. FU, *Mining multiple-level association rules in large databases*, Knowledge and Data Engineering, 11 (1999), pp. 798–804.

64. J. HAN, J. PEI, AND Y. YIN, *Mining frequent patterns without candidate generation*, in Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, ACM Press, 2000, pp. 1–12.
65. J. HAN AND Y. FU, *Discovery of multiple-level association rules from large databases*, in Proceedings of 1995 International Conference on Very Large Data Bases, Zürich, Switzerland, 1995.
66. J. HARTIGAN, *Clustering Algorithms*, John Wiley & Sons, New York, 1975.
67. P. HE, C.-J. XU, Y.-Z. LIANG, AND K.-T. FANG, *Improving the classification accuracy in chemistry via boosting technique*, Chemometrics and Intelligent Laboratory Systems, 70 (2004), pp. 39–46.
68. R. HOGG AND A. CRAIG, *Introduction to Mathematical Statistics*, Macmillan Publishing Co., New York, 4th ed., 1978.
69. J. HOLLAND, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.
70. R. HOLTE, *Very simple classification rules perform well on most commonly used datasets*, Machine Learning, 11 (1993), pp. 63–91.
71. D. HSU, S. SODERLAND, AND O. ETZIONI, *A redundant covering algorithm applied to text classification*. [www.cs.washington.edu/homes/hsud/papers/quals.ps](http://www.cs.washington.edu/homes/hsud/papers/quals.ps), accessed May, 2004, 1998.
72. W. HSU, H. GUO, B. PERRY, AND J. STILSON, *A permutation genetic algorithm for variable ordering in learning Bayesian networks from data*, in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), New York, 2002.
73. A. JAIN AND R. DUBES, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, NJ, 1988.
74. A. JAIN, M. MURTY, AND P. FLYNN, *Data clustering: A review*, ACM Computing Surveys, 31 (1999), pp. 264–323.
75. K. JAJUGA, A. SOKOŁOWSKI, AND H. BOCK, eds., *Classification, Clustering, and Data Analysis: Recent Advances and Applications*, Springer-Verlag, Berlin, 2002.
76. N. JARDINE AND R. SIBSON, *Mathematical Taxonomy*, John Wiley & Sons, New York, 1971.
77. M. KANTARDZIC, *Data Mining: Concepts, Models, Methods, and Algorithms*, Wiley-Interscience, Hoboken, NJ, 2003.
78. M. KANTARDZIC, *Data Mining: Concepts, Models, Methods, and Algorithms*, IEEE, Piscataway, NJ, 2003.
79. S. KASKI, *Data exploration using self-organizing maps*, PhD thesis, Acta Polytechnica Scandinavica, Mathematics, Computing and Management in Engineering, Finnish Academy of Technology, 1997.
80. G. KASS, *An exploratory technique for investigating large quantities of categorical data*, Applied Statistics, 29 (1980), pp. 119–127.
81. L. KAUFMAN AND P. ROUSSEEUW, *Finding Groups in Data: An Introduction to Cluster Analysis*, J. Wiley and Sons, New York, 1990.
82. J. KEATING, R. MASON, AND P. SEN, *Pitman's Measure of Closeness: A Comparison of Statistical Estimators*, SIAM, Philadelphia, 1993.

83. R. L. KENNEDY, Y. LEE, B. VAN ROY, C. D. REED, AND R. P. LIPPMAN, *Solving Data Mining Problems Through Pattern Recognition*, Prentice Hall PTR, Upper Saddle River, NJ, 1997.
84. M. KHAN, Q. DING, AND W. PERRIZO, *k*-nearest neighbor classification on spatial data streams using *p*-trees, in Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2002.
85. W. KLÖSGEN AND J. ZYTKOW, eds., *Handbook of Data Mining and Knowledge Discovery*, Oxford University Press, New York, 2002.
86. E. M. KNORR AND R. T. NG, *Algorithms for mining distance-based outliers in large datasets*, in Proceedings of the 24th International Conference on Very Large Data Bases, VLDB, 1998.
87. T. KOHONEN, *Self-Organizing Maps*, Springer-Verlag, Berlin, 1997.
88. K. KOPERSKI AND J. HAN, *Discovery of spatial association rules in geographic information databases*, in Proceedings of the Fourth SSD Symposium, 1995.
89. K. KOPERSKI, J. HAN, AND J. ADHIKARY, *Spatial data mining*, in Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 1996.
90. K. KOPERSKI, J. HAN, AND N. STEFANOVIC, *An efficient two-step method for classification of spatial data*, in Proceedings of the International Symposium on Spatial Data Handling, 1998.
91. M. KUBAT, I. KOPRINSKA, AND G. PFURTSCHELLER, Learning to classify biomedical signals, in *Machine Learning and Data Mining: Methods and Applications*, R. MICHALSKI, I. BRATKO, AND M. KUBAT, eds., John Wiley, West Sussex, England, 1997.
92. O.-W. KWON AND J.-H. LEE, *Web page classification based on k-nearest neighbor approach*, in Proceedings of the Fifth International Workshop on Information Retrieval with Asian Languages, Hong Kong, China, 2000.
93. M. LAST, O. MAIMON, AND E. MINKOV, *Improving stability of decision trees*, *International Journal of Pattern Recognition and Artificial Intelligence*, 16 (2002), pp. 145–159.
94. Y. LECUN, L. JACKEL, L. BOTTOU, A. BRUNOT, C. CORTES, J. DENKER, H. DRUCKER, I. GUYON, U. MULLER, E. SACKINGER, P. SIMARD, AND V. VAPNIK, *Comparison of learning algorithms for handwritten digit recognition*, in Proceedings of the International Conference on Artificial Neural Networks, 1995.
95. E. LEHMANN AND G. CASELLA, *Theory of Point Estimation*, Springer-Verlag, New York, 2nd ed., 1998.
96. R. LEWIS, *An introduction to classification and regression tree (cart) analysis*, in Annual Meeting of the Society for Academic Emergency Medicine, San Francisco, CA, 2002.
97. Y. LI, Z. BANDAR, AND D. MCLEAN, *An approach for measuring semantic similarity between words using multiple information sources*, *IEEE Transactions on Knowledge and Data Engineering*, 15(4) (2003), pp. 871–882.
98. A. LIKAS, N. VLASSIS, AND J. VERBEEK, *The global k-means clustering algorithm*, *Pattern Recognition*, 36 (2003), pp. 451–461.

99. B. LINDGREN, *Statistics Theory*, Chapman and Hall, London, 4th ed., 1993.
100. W. LOH AND Y. SHIH, *Split selection methods for classification tree*, *Statistica Sinica*, 7 (1997), pp. 815–840.
101. P. LOPEZ-DE-TERUEL, J. GARCIA, AND M. ACACIO, *The Parallel EM Algorithm and its Applications in Computer Vision*, PDPTA, University of Murcia, Spain, 1999.
102. W. LU, J. HAN, AND B. C. OOI, *Discovery of general knowledge in large spatial databases*, in Proceedings of Far East Workshop on Geographic Information Systems, 1993.
103. W. MACHEREY, D. KEYSERS, J. DAHMEN, AND H. NEY, *Improving automatic speech recognition using tangent distance*, in Eurospeech 2001, 7th European Conference on Speech Communication and Technology, Aalborg, Denmark, 2001.
104. R. MARMELSTEIN, *Application of genetic algorithms to data mining*, in Proceedings of the Eighth Midwest Artificial Intelligence and Cognitive Science Conference, 1997.
105. W. MARTINEZ AND A. MARTINEZ, *Computational Statistics Handbook with Matlab*, Chapman and Hall/CRC, New York, 2002.
106. M. ESTER, H. P. KRIEGEL, AND J. SANDER, *Spatial data mining: A database approach*, in Proceedings of the Fifth SSD Symposium, 1997.
107. B. MINAEI-BIDGOLI AND W. P. III, *Using genetic algorithms for data mining optimization in an educational web-based system*, in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), 2003.
108. D. MODHA AND W. SPANGLER, *Feature weighting in k-means clustering*, *Machine Learning*, 52 (2003), pp. 217–237.
109. A. MOOD, F. GRAYBILL, AND D. BOES, *Introduction to the Theory of Statistics*, McGraw-Hill Publishing, New York, 3rd ed., 1974.
110. G. NAGY, *State of the art in pattern recognition*, in Proceedings of IEEE, 1968, pp. 836–862.
111. M. NAKASHIMA, Y. KANEKO, AND T. ITO, *Ranking of documents by measures considering conceptual dependence between terms*, *Systems and Computers in Japan*, 34 (2003), pp. 81–91.
112. J. NEVILLE, D. JENSEN, AND B. GALLAHER, *Simple estimators for relational Bayesian classifiers*, in Proceedings of the Third IEEE International Conference on Data Mining, 2003.
113. R. NG AND J. HAN, *Efficient and effective clustering methods for spatial data mining*, in Proceedings of the International VLDB Conference, 1994.
114. R. NG AND J. HAN, *CLARANS: A method for clustering objects for spatial data mining*, *IEEE Transactions on Knowledge and Data Engineering*, 14 (2002), pp. 1003–1016.
115. R. ODDY, S. ROBERTSON, C. VAN RIJSBERGEN, AND P. WILLIAMS, eds., *Information Retrieval Research*, Butterworth, London, 1981.
116. E. OJA AND S. KASKI, eds., *Kohonen Maps*, Elsevier Science, Amsterdam, 1999.
117. C. OLARU AND L. WEHENKEL, *A complete fuzzy decision tree technique*, *Fuzzy Sets and Systems*, 138 (2003), pp. 221–254.

118. M. PAL AND P. M. MATHER, *An assessment of the effectiveness of decision tree methods for land cover classification*, *Remote Sensing of Environment*, 86 (2003), pp. 554–565.
119. J. S. PARK, M. CHEN, AND P. S. YU, *An effective hash-based algorithm for mining association rules*, in *Proceedings of the ACM International Conference on Management of Data*, 1995, pp. 175–186.
120. S. PARTHASARATHY, M. J. ZAKI, M. OGIHARA, AND W. LI, *Parallel data mining for association rules on shared-memory system*, *Knowledge and Information Systems*, 3 (2001), pp. 1–29.
121. J. PEÑA, J. LOZANO, AND P. LARRAÑAGA, *An empirical comparison of four initialization methods for the k-means algorithm*, *Pattern Recognition Letters*, 20 (1999), pp. 1027–1040.
122. H. PENG AND C. DING, *Structure search and stability enhancement of Bayesian networks*, in *Proceedings of the Third IEEE International Conference on Data Mining*, 2003.
123. W. PERRIZO, Q. DING, AND A. DENTON, *Lazy classifiers using p-trees*. [www.citeseer.nj.nec.com/perrizo02lazy.html](http://www.citeseer.nj.nec.com/perrizo02lazy.html), accessed May, 2004, 2002.
124. J. R. QUINLAN, *Induction of decision trees*, *Machine Learning*, 11 (1986), pp. 81–106.
125. J. R. QUINLAN, *Simplifying decision trees*, *International Journal of Man-Machine Studies*, 27 (1987), pp. 221–234.
126. J. R. QUINLAN, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.
127. C. REEVES AND J. ROWE, *Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory*, Kluwer Academic Publishers, Boston, 2003.
128. E. S. RISTAD AND P. N. YIANILOS, *Learning string-edit distance*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20 (1998), pp. 522–532.
129. M. RODRÍGUEZ AND M. EGENHOFER, *Determining semantic similarity among entity classes from different ontologies*, *IEEE Transactions on Knowledge and Data Engineering*, 15(2) (2003), pp. 442–456.
130. C. ROMERO, S. VENTURA, C. DE CASTRO, W. HALL, AND M. NG, *Using genetic algorithms for data mining in web-based educational hypermedia systems*, in *Proceedings of the Workshop on Adaptive Systems for Web-based Education*, 2002.
131. T. RUI AND X. GUORONG, *A classification method of diagnostic image based on tangent distance*, in *International Conference on Diagnostic Imaging and Analysis*, Shanghai, China, 2002, ICDIA'02.
132. G. SALTON, *Automatic Information Organization and Retrieval*, McGraw-Hill, New York, 1968.
133. G. SALTON AND M. MCGILL, *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983.
134. J. SANDER, M. ESTER, H.-P. KRIEGEL, AND X. XU, *Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications*, *Data Mining and Knowledge Discovery*, 2 (1998), pp. 169–194.

135. A. SAVASERE, E. OMIECINSKI, AND S. NAVATHE, *An efficient algorithm for mining association rules in large databases*, in Proceedings of the 21st International Conference on Very Large Databases, Zurich, Switzerland, 1995, pp. 432–444.
136. X. SHEN AND M. SPANN, *Segmentation of 2D and 3D images through a hierarchical clustering based on region modeling*, in International Conference on Image Processing (ICIP '97), vol. 3, Washington, DC, 1997.
137. T. SHINTANI AND M. KITSUREGAWA, *Hash based parallel algorithms for mining association rules*, in Proceedings of the Parallel and Distributed Information Systems Conference, 1996.
138. P. SIMARD, Y. L. CUN, AND J. DENKER, *An efficient pattern recognition using a new transformation distance*, in Advances in Neural Information Processing Systems 5, S. HANSON, J. COWAN, AND C. GILES, eds., Morgan Kaufmann, San Mateo, CA, 1993, pp. 50–58.
139. R. SOKAL AND C. D. MICHENER, *A statistical method for evaluating systematic relationships*, University of Kansas Science Bulletin, 38 (1958), pp. 1409–1438.
140. R. SRIKANT AND R. AGRAWAL, *Mining generalized association rules*, in VLDB'95: Proceedings of the 21st International Conference on Very Large Data Bases, Morgan Kaufmann, Zurich, Switzerland, 1995.
141. R. SRIKANT AND R. AGRAWAL, *Mining quantitative association rules in large relational tables*, in Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, ACM Press, 1996.
142. P. TAYLOR, C. BRENNELL, AND B. SNOOK, *Problems of classification in investigative psychology*, in Classification, Clustering, and Data Analysis: Recent Advances and Applications, K. JAJUGA, A. SOKOLOWSKI, AND H. H. BOCK, eds., Springer-Verlag, Berlin, Heidelberg, New York, 2002.
143. H. TOIVONEN, *Sampling large databases for association rules*, in Proceedings of the 22nd International Conference on Very Large Databases, Mumbai, India, 1996, pp. 134–145.
144. O. TROYANSKAYA, K. DOLINSKI, A. OWEN, R. ALTMAN, AND D. BOTSTEIN, *A Bayesian framework for combining heterogeneous data sources for gene function prediction*, in Proceedings of the National Academy of Science, 100(14) (2003), pp. 8348–8353.
145. J. VITTER, *Random sampling with a reservoir*, ACM Transactions on Mathematical Software, 11 (1985), pp. 37–57.
146. W. WANG, J. YANG, AND R. MUNTZ, *STING: A statistical information grid approach to spatial data mining*, in Proceedings of the International VLDB Conference, 1997.
147. W. WANG, J. YANG, AND R. MUNTZ, *An approach to active spatial data mining based on statistical information*, IEEE Transactions on Knowledge and Data Engineering, 12 (2000), pp. 715–728.
148. J. WARD, *Hierarchical grouping to optimize an objective function*, Journal of American Statistical Association, 58 (1963), pp. 236–244.
149. M. WASAN, *Parametric Estimation*, McGraw-Hill Book Company, New York, 1970.

150. C. WEI, Y. LEE, AND C. HSU, *Empirical comparison of fast partitioning-based clustering algorithms for large data sets*, Expert Systems with Applications, 24 (2003), pp. 351–363.
151. D. WETTSCHERECK, *A Study of Distance-Based Machine Learning Algorithms*, PhD thesis, Oregon State University, Computer Science Department, Corvallis, OR, 1994.
152. R. WILCOX, *Introduction to Robust Estimation and Hypothesis Testing*, Academic Press, New York, 1997.
153. L. WILKINSON, *Tree structured data analysis: Aid, chaid, and cart*, in Sawtooth/SYSTAT Joint Software Conference, Sun Valley, ID, 1992.
154. P. WILLETT, *Recent trends in hierarchic document clustering: A critical review*, Information Processing and Management, 24 (1988), pp. 577–597.
155. D. WILSON AND T. MARTINEZ, *Improved heterogeneous distance functions*, Journal of Artificial Intelligence Research, 6 (1997), pp. 1–34.
156. I. WITTEN AND E. FRANK, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann Publishers, San Mateo, CA, 2000.
157. K. WU AND M. YANG, *Alternative c-means clustering algorithms*, Pattern Recognition, 35 (2002), pp. 2267–2278.
158. W. WU, H. XIONG, AND S. SHEKHAR, *Clustering and Information Retrieval*, Kluwer Academic Publishers, Norwell, MA, 2004.
159. Y. WU, Q. TIAN, AND T. HUANG, *Discriminant-em algorithm with application to image retrieval*, in IEEE Computer Vision and Pattern Recognition (CVPR'00), South Carolina, 1 (2000), p. 1222.
160. X. XU, M. ESTER, H.-P. KRIEGEL, AND J. SANDER, *A distribution-based clustering algorithm for mining in large spatial databases*, in Proceedings of the IEEE International Conference on Data Engineering, 1998.
161. M. YANG AND N. AHUJA, *Gaussian mixture model for human skin color and its applications in image and video databases*, Technical Report, Beckman Institute, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.
162. Y. YANG AND X. LIU, *A re-examination of text categorization methods*, in Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval, Berkeley, CA, 1999.
163. N. YE, *The Handbook of Data Mining*, Lawrence Erlbaum Associates, Mahwah, NJ, 2003.
164. P. N. YIANILOS, *Data structures and algorithms for nearest neighbor search in general metric spaces*, in Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1993.
165. M. J. ZAKI, *Parallel and distributed association mining: A survey*, IEEE Concurrency, 7 (1999), pp. 14–25.
166. M. J. ZAKI, M. OGIHARA, S. PARTHASARATHY, AND W. LI, *Parallel data mining for association rules on shared-memory multi-processors*, Technical Report, TR 618, University of Rochester, Computer Science Department, 1996.
167. M. J. ZAKI, S. PARTHASARATHY, M. OGIHARA, AND W. LI, *Parallel algorithms for fast discovery of association rules*, Data Mining and Knowledge Discovery, 1 (1997), pp. 343–373.

168. T. ZHANG, R. RAMAKRISHNAN, AND M. LIVNY, *BIRCH: An efficient data clustering method for very large databases*, in ACM SIGMOD International Conference on Management of Data, 1996.
169. C. ZHOU, W. XIAO, T. TIRPAK, AND P. NELSON, *Evolving accurate and compact classification rules with gene expression programming*, IEEE Transactions on Evolutionary Computation, 7 (2003), pp. 519–531.

This page is intentionally left blank

## INDEX

- A priori* algorithm, 158, 191
- kd*-trees, 127
- 1R algorithm, 91
  
- AID, 41
- Aldridge, Matthew, 99, 193
- algorithm
  - agglomerative, 104, 109, 111, 112, 145
  - distributed, 171
  - divisive, 104, 109, 111
  - parallel, 171
  - partitioning, 162, 173
- Angoss KnowledgeSEEKER, 41
- APM algorithm, 177
- architecture
  - distributed memory, 174
  - shared memory, 171
- arcing, 47
- association rules, 54, 153, 169, 184
  - confidence, 155, 189, 197
  - conviction, 190
  - generalized, 185
  - multiple-level, 187
  - spatial, 191
  - support, 155, 189, 191, 197
  - temporal, 191
- attribute hierarchy, 195
  
- bagging, 47
- Bayes decision rule, 15, 20
- Bayes theorem, 15, 20, 76
- Bayesian networks, 15, 24, 66
- bias, 9
- bioinformatics, 29, 37
  
- biological taxonomy, 107
- BIRCH, 137
- boosting, 47, 83
- Build DT, 81
  
- C4, 92
- C4.5, 41, 82
- C5.0 (See5), 41, 83
- CACTUS, 151
- CaD, 175
- candidate itemsets, 157
- CART, 41, 84
- catalog design, 169
- CCPD, 176
- CD, 173
- Central Limit Theorem, 10
- centroid, 106
- CHAID, 41
- chemometrics, 50
- chi-squared test, 183, 188, 190
- chromosome, 54
- CLARA, 129, 203
- CLARANS, 129, 202
- classification
  - Bayes, 17
  - document, 120
  - gene, 25
  - nonparametric, 67
  - rules, 87
  - spatial, 200
  - tree, 40
  - unsupervised, 99, 110
- classifier
  - lazy, 76
  - non-parametric, 19

- classifiers
  - parametric, 19
- CLS, 41
- clustering, 99
  - k*-means, 123
  - assessment, 105
  - categorical, 143
  - divide-and-conquer, 135
  - hierarchical, 36, 103, 109, 110, 144
    - divisive, 118
  - incremental, 135
  - nearest neighbor, 126
  - parallelization, 135
  - partitional, 105, 110, 121
  - spatial, 202
  - squared error, 122
  - tendency, 105
  - validation, 107
- computer vision, 8
- concept hierarchy, 194
- consistency, 11
- consumer credit
  - fraud, 103
  - risk, 50
- COOLCAT, 149
- correlation
  - rules, 188
  - upward closure, 189
- cosine measure, 31
- covariance, 21
- covering algorithms, 90
- CURE, 140
- curse of dimensionality, 54, 122
- customer segmentation, 169
  
- DAIRY, 97
- data abstraction, 105
- data parallelism, 171
- Daylight<sup>TM</sup> Chemical Information System, 110
- DBCLASD, 204
- DBSCAN, 139, 203
- DD, 174
- decision support, 169
- decision trees, 39, 77, 79, 88, 201
- dendrogram, 103, 111
  
- DIANA algorithm, 118
- DIC algorithm, 177
- Dice coefficient, 30
- discrimination, 15
- dissimilarity measure, 32
- distance
  - $L_p$ , 145
  - city block, 32, 69, 103, 111
  - edit, 71
  - Euclidean, 32, 70, 103, 111
  - Manhattan, 32, 103
  - measure, 32, 67, 124
  - tangent, 71
  - Tanimoto, 71, 111
- distance-based algorithms, 68
- Distributed Mining of Association (DMA), 172
- DNA sequences, 37
  
- earthquakes, 204
- efficiency, 10
- entropy, 45, 81, 149
- error
  - mean squared, 9
  - standard, 10
- Expectation-Maximization algorithm, 1, 6
- expected reproduction, 58
  
- False candidates, 164
- Fast Update (FUP) method, 184
- feature
  - extraction, 18, 101
  - representation, 56, 102
  - selection, 77, 101
  - vector, 23
  - weighting, 124
- financial decision making, 132
- financial forecasting, 169
- Fisher, Sir Ronald A., 4
- fitness
  - function, 57
  - value, 54
- Forgy Approach (FA), 125
- Frequent Pattern (FP) trees, 191
- Fuentes, Erika, 15

- function
  - crossover, 60
  - likelihood, 4
  - multivariate density, 23
  - negative border, 161
- fuzzy systems, 50
- gain ratio, 43
- gamma distribution, 3
- Gaussian behavior, 21
- GDBSCAN, 203
- genetic algorithms, 50, 53
- Geographic Information System (GIS), 194
- goodness measure, 147
- graph
  - acyclic, 25
  - neighborhood, 201
- greedy algorithms, 43
- Heinrich, Kevin E., 27, 121
- Holland, James, 54
- homeland security, 87
- Hoveland, 42
- Hunt, 42
- ID3 (Interactive Dichotomizer 3), 80
  - extension, 201
- IDD, 175
- image
  - analysis, 107, 132
  - compression, 66
  - processing, 120
  - recognition, 87
  - retrieval, 8
- information gain, 80, 81
- information retrieval, 28
  - 3D models, 35
  - vector space model, 29
- investigative psychology, 107, 132
- IREP, 97
- ISODATA algorithm, 125
- iterative evolution, 62
- Jaccard coefficient, 31, 103, 145
- jackknife method, 11
- Kaufman Approach (KA), 125
- KNN (*k*-Nearest Neighbors), 68, 72, 74
- linkage method
  - average, 116
  - centroid, 116
  - complete, 114
  - single, 112
- Liu, Huadong, 1, 109
- load balancing, 172, 176
- M-trees, 127
- machine learning, 68, 97, 132
- Macqueen Approach (MA), 125
- market basket, 154
- Markov Chains, 16
- matrix
  - boolean, 146
  - correlation, 23
  - dissimilarity, 101
  - distance, 112
  - proximity, 101
  - similarity, 101
- maximum likelihood, 1
- Mazeroff, Geoffrey, 53, 153
- medical diagnosis, 87, 132, 169
- medioid, 127
- method of moments, 3
- microarray analysis, 25
- minefield identification, 204
- Minkowski metric, 102
- molecular biology, 25
- monothetic method, 118
- mutation operator, 62
- neural networks, 47, 50
- nonlinear prediction methods, 41
- numerical methods, 22
- object, 100
- Omitaomu, Olufemi A., 39, 143
- outlier detection, 68
- overlap coefficient, 30

- PAM, 127, 202
- Parallel Data Mining (PDM), 172
- pattern, 100
  - matrix, 100
  - proximity measure, 102
  - recognition, 8, 19, 87, 120, 132
  - representation, 101
- PCCD, 177
- point estimation, 1
- polythetic method, 118
- population, 56
- precision, 29
- principal component analysis, 41
- PRISM algorithm, 94
- probability
  - conditional, 20
  - density function, 21
  - of selection, 58
  - posterior, 20
  - prior, 20
- probabilty
  - posterior, 17
- Pulatova, Shakhina, 87, 169
- quad trees, 127
- query
  - region, 199
  - similarity, 68
- R\*-trees, 127
- R-trees, 127
- Random Approach (RA), 125
- recall, 29
- remote sensing, 194
- REP, 97
- retail business, 50, 153
- RIPPER, 97
- ROCK, 145
- rule-based algorithms, 90
- scaleup, 178
- Shahnaz, Farial, 79, 133
- Shen, Zeqian, 67
- similarity measure, 28
  - asymmetric, 31
- sizeup, 178
- sketch query, 36
- SLIPPER, 97
- SOM (Self-Organizing Map), 131
  - Kohonen, 132
- spatial
  - characteristic rule, 196
  - data, 194
  - discriminant rule, 196
  - generalization, 194
- speedup, 178
- splitting variable, 46
- STING, 198
- stochastic-optimization methods, 54
- summarization, 2, 151
- supervised methods, 19
- SWAP-1, 97
- task parallelism, 171
- telecommunication alarm prediction, 169
- text mining, 153
- THCA, 148
- traveling-salesman problem, 63
- tree pruning, 47, 97
- TREEDISC, 41
- trie, 177
- unsupervised methods, 19
- variance, 10
- Ward method, 117
- Web mining, 153
- Web-based hypermedia, 66
- WPGMA, 116
- X-trees, 127
- YarKhan, Asim, 183