Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

ELSEVIER

Fuzzy Sets and Systems 138 (2003) 221–254

FUZZY
sets and systems

www.elsevier.com/locate/fss

A complete fuzzy decision tree technique

Cristina Olaru*, Louis Wehenkel

University of Liège, Department of Electrical Engineering and Computer Science, Montefiore Institute, B28, B-4000, Liège, Belgium

Received 5 February 2002; received in revised form 29 January 2003; accepted 18 February 2003

Abstract

In this paper, a new method of fuzzy decision trees called soft decision trees (SDT) is presented. This method combines tree growing and pruning, to determine the structure of the soft decision tree, with refitting and backfitting, to improve its generalization capabilities. The method is explained and motivated and its behavior is first analyzed empirically on 3 large databases in terms of classification error rate, model complexity and CPU time. A comparative study on 11 standard UCI Repository databases then shows that the soft decision trees produced by this method are significantly more accurate than standard decision trees. Moreover, a global model variance study shows a much lower variance for soft decision trees than for standard trees as a direct cause of the improved accuracy.

Keywords: Learning; Approximate reasoning; Fuzzy decision tree; Data mining; Soft split; Pruning; Global optimization; Regression tree; Neural network

1. Introduction

In almost every real-life field one is confronted with growing amounts of data coming from measurements, simulations or simply, from manual data registration and centralization procedures, and, most often, it would be a waste not to take advantage of these data. Recent developments in data storage devices, database management systems, computer technologies and automatic learning techniques make data analysis tasks easier and more efficient. In this context, data mining is a modern concept beginning to be widely used. The general purpose of data mining is to process the information embedded in data so as to develop better ways to handle data and support future decision-making. Machine learning, association rules, clustering methods, artificial neural networks, statistical and visualization tools are common techniques used in data mining. The perfect data

* Corresponding author. Tel.: +32-4-3662718; fax: +32-43662984.
E-mail addresses: ana.olaru@ulg.ac.be (C. Olaru), l.wehenkel@ulg.ac.be (L. Wehenkel).

0165-0114/03/$ - see front matter
doi:10.1016/S0165-0114(03)00089-7

mining technique would simultaneously: be able to manage large amounts of data, be accurate, be interpretable and comprehensible, be efficient when training and using it, be a problem-independent tool that manages to automatically extract the most relevant features for a given problem and to fix its parameters and its model complexity with minimal human intervention.

In the spirit of solving our-days-needs of learning methods, this paper proposes a method called *soft decision trees* (SDT), i.e. a variant of classical decision tree inductive learning using fuzzy set theory. Decision tree techniques have already been shown to be interpretable, efficient, problem-independent and able to treat large scale applications. But they are also recognized as highly unstable classifiers with respect to minor perturbations in the training data, in other words, methods presenting high variance. Fuzzy logic brings in an improvement in these aspects due to the elasticity of fuzzy sets formalism. The proposed method has been studied in detail and compared with alternative crisp methods and the results show a much improved prediction accuracy, explainable by a much reduced model variance. Also, more stability at the parameters level (almost 50% less parameter variance than for classical decision trees) leads to better interpretability.

The paper is organized as follows: Section 2 presents the proposed method. Section 3 reports simulation results. Section 4 mentions possible extensions of the method and further work, discusses three fundamental reasons for increased accuracy of soft decision trees and mentions related work. Finally, Section 5 concludes this work by pointing out the specific contributions of the proposed method. Appendix A collects mathematical and algorithmic details.

2. Proposed algorithm

2.1. Learning problem

We consider a supervised learning problem from examples, that may be generically formulated as: given a set of examples (the learning set LS) of associated input/output pairs, derive a general rule representing the underlying input/output relationship, which may be used to explain the observed pairs and/or predict output values for any new input. We use the term *attribute* to denote the parameters that describe the input information and the term *object* to denote an input/output pair. In a given database, a column is usually associated with an attribute or with an output and a row with an object.

Let U denote the universe of all possible objects for a given problem. A fuzzy set $S \subset U$ of objects is characterized by a *membership function* $\mu_S : U \to [0,1]$ which associates with each object o of U a number $\mu_S(o)$ in the interval $[0,1]$ representing the degree of affiliation of the object o to S.

2.2. Soft decision trees versus crisp regression trees

We present intuitively the formal representation of a soft decision tree by explaining first the regression tree (RT) type of induction. Regression trees and soft decision trees are extensions of the decision tree induction technique, predicting a numerical output, rather than a discrete class. Both trees may be used in regression problems given their output (numerical by definition), or in classification problems, by a priori defining symbolic classes on the numerical output. Fig. 1 shows

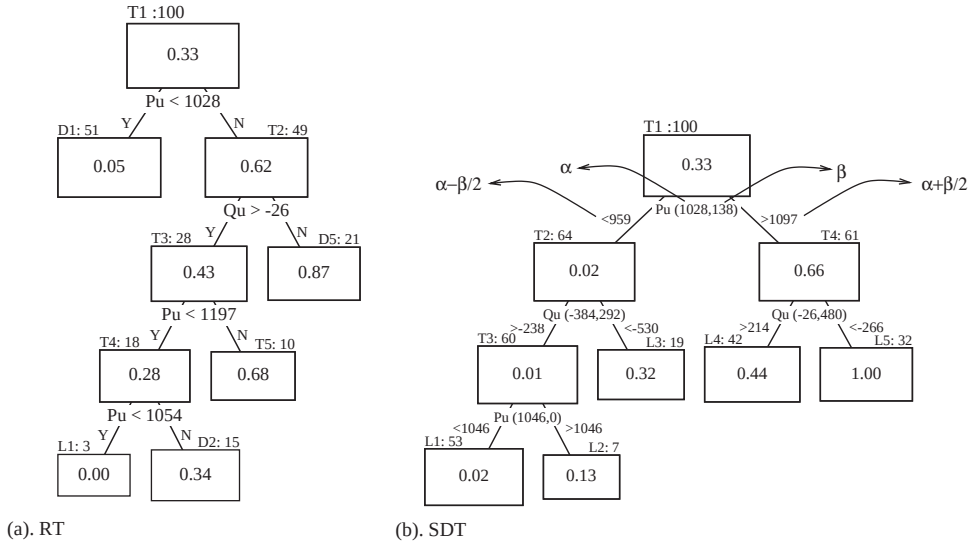

Fig. 1. Regression tree versus soft decision tree.

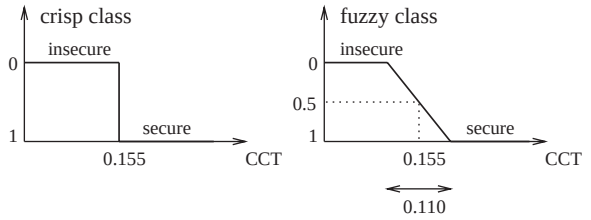

Fig. 2. Example of a crisp class and a fuzzy class for omib data.

a crisp regression tree (left part (a)) and a soft decision tree (right part (b)). Both were built for an illustrative electric power system security problem (the so-called omib database described in Section 3.1.1). The input space is here defined by two attributes characterizing the system state, denoted respectively by "Pu" and "Qu" (active and reactive powers of the (single) synchronous generator of this system). The output is in both cases numerical and reflects the security level of the electric power system. We formulated it as a fuzzy class based on a parameter called critical clearing time (CCT) of the system state (see Fig. 2). The trees predict the membership degree of instances to this fuzzy class. In a crisp decision (see Fig. 2), the system could be considered "insecure" if the CCT is smaller than 155 ms, "secure" otherwise (classification problem). In a soft decision, there is a transition region of 110 ms (as we defined it) in which the system is neither "secure" nor "insecure" (regression problem). Next, significant differences between soft and crisp trees are pointed out on the OMIB example.

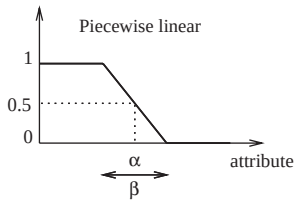

Fig. 3. Example of discriminator function: piecewise linear.

2.2.1. Crisp regression tree

The regression tree in Fig. 1(a) has four test nodes and five terminal nodes. Each node box is stamped with the local estimation of the output. Under each test node, the selected test appears as a condition on a single attribute at a time, regarding a *single* threshold and having two possible answers: yes or no (left or right). The local input space is thus split into two (in our case of binary trees) *non-overlapping* subregions of objects. The objects in one such subregion should ideally have the same output value. The tree may be translated into an equivalent set of mutually exclusive rules, each one corresponding to a path from the top node to a terminal node. To classify a new instance, one starts at the top node and applies sequentially the dichotomous tests encountered to select the appropriate successor. Finally, a *unique* path is followed, a *unique* terminal node is reached and the output estimation stored there is assigned to this instance.

Example. In the case of an instance with attributes $Pu = 1100MW$ and $Qu = -40MVar$, the terminal node D5 is reached and the tree estimation of the membership degree to the stability class for this case is 0.87, i.e. the system is insecure with the degree of 0.87. We may also express the result in a crisp way (see crisp class of Fig. 2): since degree 0.87 corresponds to a CCT value smaller than 155 ms, the conclusion is that the class estimated by the regression tree is "insecure". By translating the tree into a rule base, the rule extracted from the tree fired by our instance looks like:

If $Pu \geqslant 1028MW$ and $Qu \leqslant -26MVar$, then *degree* 0.87.

2.2.2. Soft decision tree

The soft decision tree in Fig. 1(b) has also four test nodes and five terminal nodes. Each node is also marked with its local estimation of the output. Under each test node, the selected test appears as a condition on a single attribute at a time, regarding a pair of *two* parameters (values in brackets). These two parameters characterize the function called *discriminator* needed to fuzzily split the local set of objects of a given current test node. A widely used shape of discriminator function is the piecewise linear one (see Fig. 3). The two parameters defining it are: α, which is the location of the cut-point and corresponds to the split threshold in a test node of a decision or a regression tree, and β which is the width, the degree of spread that defines the transition region on the attribute chosen in that node. With such a piecewise linear discriminator function, the local input space of a node is split (fuzzy partitioned) into two *overlapping* subregions of objects. Some objects go only to the left successor, some only to the right one, and the objects in the overlap region go to both successors. The larger the transition region in a test node, the larger the overlap and the softer the decision in that node. In consequence, any given instance is in general propagated through the tree by *multiple* decision paths in parallel: in the simplest case through one path, in the most complex case,

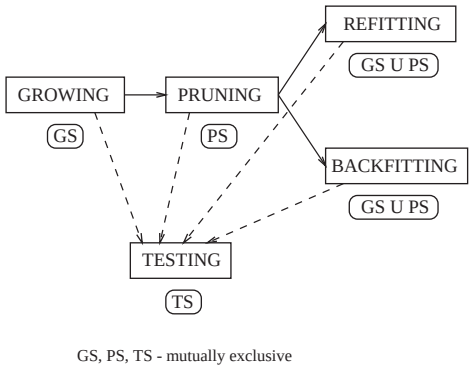

Fig. 4. The procedure of building a soft decision tree.

through all the paths. This given instance does not effectively belong to a node it passes through, but has a membership degree attached to that node. Thus, the node may be seen as a fuzzy set. Finally, the given instance reaches *multiple* terminal nodes and the output estimations given by all these terminal nodes are aggregated through some defuzzification scheme in order to obtain the final estimated membership degree to the target class.

Example. In the case of the instance with attributes $Pu = 1100MW$ and $Qu = -40MVar$, only two paths are followed: T1-T4-L4 and T1-T4-L5. The membership degree of our instance to the fuzzy set of test node T4 equals 1.0 as far as all the objects with $Pu = 1100MW$ go only to right ($1100MW > 1097MW$) (see Fig. 1(b)). Since $-40MVar$ is in between $-266MVar$ and $214MVar$, the instance goes both to left and right successors of T4, thus reaching leaf L4 with membership degree of 0.43 and leaf L5 with membership degree of 0.57. Finally, by aggregating the two paths, the tree estimation of the membership degree to the stability class for the given instance is $1.0 * 0.43 * label_{L4} + 1.0 * 0.57 * label_{L5} = 1.0 * 0.43 * 0.44 + 1.0 * 0.57 * 1.00 = 0.76$ (compare this result with the one obtained by the crisp tree). The result expressed in a crisp way says that the system is "insecure", since degree 0.76 corresponds to a CCT value smaller than 155 ms. The rules fired by our instance correspond to the two identified paths:

If $Pu \geqslant 959MW$ and $Qu \geqslant -266MVar$, then *degree* 0.44

If $Pu \geqslant 959MW$ and $Qu \leqslant 214MVar$, then *degree* 1.00.

2.3. Building a soft decision tree

Fig. 4 presents an overview of the complete procedure for building a soft decision tree. The process starts by *growing* a "sufficiently large" tree using a set of objects called growing set *GS*. Tree nodes are successively added in a top-down fashion, until stopping criteria are met. Then the grown tree is *pruned* in a bottom-up fashion to remove its irrelevant parts. At this stage, a cross-validation technique is used which makes use of an another set of objects, called the pruning set *PS*. Next, a third step could be either a *refitting* step or a *backfitting* step. Both consist of tuning certain parameters of the pruned tree model in order to improve its approximation capabilities further. These steps use the whole learning set: $LS = GS \cup PS$. At the end of every intermediate stage, the

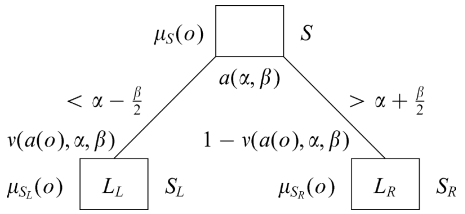

Fig. 5. Fuzzy partitioning of a node in a soft decision tree.

obtained trees (fully developed, pruned, refitted or backfitted) may be *tested* in order to quantify their generalization capability. A third sample, independent from the learning set, called test set *TS*, is used to evaluate the predictive accuracy of these trees.

Thus, a given dataset is split initially into two disjoint parts, the learning set *LS* and the test set *TS*. The learning set is then used to create two other disjoint sets: the growing *GS* and the pruning *PS* sets. Growing, pruning and test sets are (normally) composed of mutually independent samples, as far as one can assume that this is the case for the original dataset. We now describe the basic principles of each step of this method. Technical details are provided in Appendix A.

2.4. SDT growing

By analogy with the CART method of decision and regression tree building [9], the procedure for growing a soft decision tree relies on three basic items: a method to select a (fuzzy) split at every new node of the tree, a rule for determining when a node should be considered terminal, and a rule for assigning a label to every identified terminal node.

2.4.1. Soft tree semantics

A soft tree is an approximation structure to compute the degree of membership of *objects* to a particular class (or concept), as a function of the attribute values of these objects. Let us denote by C the output class, be it crisp (0/1) or fuzzy, by $\mu_C(o)$ the degree of membership of an object o to this class, and by $\hat{\mu}_C(o)$ this membership degree as estimated by a tree. We also assume that all the attribute values are numerical and normalized in $[0,1]$.

Fig. 5 shows the split of a tree node corresponding to a fuzzy set S into two fuzzy subsets, S_L the left one and S_R the right one, based on the chosen attribute a at the node S (μ_S, μ_{S_L} and μ_{S_R} denote the membership functions of fuzzy sets S, S_L and S_R, respectively). In general, a discriminator function $v(a(o),\alpha,\beta,\gamma\ldots) \to [0\ldots1]$ is attached to the node. It determines the node's fuzzy dichotomy based on the attribute and on the values taken by several parameters $(\alpha,\beta,\gamma\ldots)$. In what follows we restrict our discussion to piecewise linear discriminator templates (see Fig. 3); these are defined by a threshold α and width parameter β. Then, the membership degree of an object to the left successor's subset S_L is determined in the following way:

$$\mu_{S_L}(o) = \mu_S(o)v(a(o),\alpha,\beta). \tag{1}$$

We say that an object goes (or belongs) to the left successor if $\mu_{S_L}(o)$ is strictly positive. Since in a piecewise linear discriminator function $v(a(o),\alpha,\beta) = 0$ for all objects for which $a(o) > \alpha + \beta/2$,

only those objects of S such that $a(o) \leqslant \alpha + \beta/2$ go towards the left successor S_L. Similarly, the objects directed toward the right successor S_R are weighted by the complement of the discriminator function value:

$$\mu_{S_R}(o) = \mu_S(o)(1 - v(a(o), \alpha, \beta)), \tag{2}$$

and correspond to the condition $a(o) \geqslant \alpha - \beta/2$.

In a tree, all nodes, except the root node, are successors of their (single) parent node. The membership degree of any object to the fuzzy subset [1] of any node S in the tree is thus defined recursively as a function of attribute values, parameters defining the discriminator functions used at the different ancestors of this node, and its membership degree to the root node. As concerns the root node (let us denote it by R) of the tree, the membership degree $\mu_R(o)$, may have a priori user defined values. But usually $\mu_R(\cdot) = 1.0$, since in most databases all objects are equally weighted at the beginning. Nevertheless, the possibility to give different weights to objects may be of interest in the context of certain data mining applications.

Let j denote a node of a tree, let L_j denote a numerical value (or label) attached to this node, and let S_{L_j} be the fuzzy subset corresponding to this node. Then, the membership degree to C estimated by a tree for a certain object o is the average of all the labels L_j attached to the leaves, weighted by the membership degrees of the object to the fuzzy subsets of these leaves, $\mu_{S_{L_j}}(o)$:

$$\hat{\mu}_C(o) = \frac{\sum_{j \in leaves} \mu_{S_{L_j}}(o) L_j}{\sum_{j \in leaves} \mu_{S_{L_j}(o)}}, \tag{3}$$

where *leaves* denotes the set of indices of the terminal nodes of the considered tree. Notice that $\sum_{j \in leaves} \mu_{S_{L_j}(o)}$ is equal to the degree of membership of the object to the root node $\mu_R(o)$, and is normally equal to 1. In the rest of this paper we will make this assumption so as to simplify our notations.

2.4.2. Automatic fuzzy partitioning of a node

Some fuzzy decision tree induction methods assume that the discriminator functions are a priori defined. In our method, on the other hand, soft decision tree growing implies the *automatic* generation of the best fuzzy split of the most discriminating attribute at each new developed node of the tree. The procedure used to achieve this goal is further detailed below.

Objective: Given S, fuzzy set in a soft decision tree, find attribute $a(\cdot)$, threshold α and width β (parameters defining the discriminator function v) together with successors labels L_L and L_R, so as to minimize the squared error function

$$E_S = \sum_{o \in S} \mu_S(o)[\mu_C(o) - \hat{\mu}'_C(o)]^2 \tag{4}$$

where

$$\hat{\mu}'_C(o) = v(a(o), \alpha, \beta) L_L + (1 - v(a(o), \alpha, \beta)) L_R, \tag{5}$$

[1] In our notations we use the same symbol to denote a node and the (fuzzy) subset of objects which corresponds to this node.

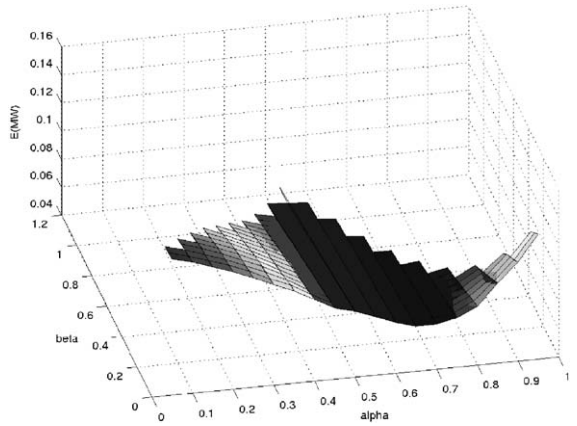

Fig. 6. $E = E(\alpha, \beta)$ surface for the fuzzy class of omib database.

depending non-linearly on α and β, and linearly on L_L and L_R. A motivation for using this kind of error function is the fact that it does not present the pathological character of preferring a crisp partitioning to the fuzzy ones, as other measures do [6,32,38,44] and as [6] proves is the case of convex measures.

Strategy: The local search is decomposed as follows (Section A.1 of Appendix A gives mathematical and algorithmic details):

- *Searching for the attribute and split location.* With a fixed $\beta = 0$ (crisp split) we search among *all* the attributes for the attribute $a(\cdot)$ yielding the smallest crisp E_S, its optimal crisp split threshold α, and its corresponding (provisional) successors labels L_L and L_R, by using crisp heuristics adapted from CART regression trees.
- *Fuzzification and labeling.* With the *optimal* attribute $a(\cdot)$ and threshold α kept frozen, both already chosen in the foregoing step, we search for the optimal width β by Fibonacci search; for every new β value, the two successors labels L_L and L_R are automatically updated to every candidate value of β by explicit linear regression formulas.

Justification: A systematic search in the space of the four free parameters was performed by plotting the error function E_S of Eq. (4) in the particular case of a *SDT* with a single test node, with the aim of understanding the shape of the function we want to minimize so as to settle the best way of searching for its minimum [34]. Fig. 6 presents the error surface in terms of two of the four parameters (α and β). Concerning the other two parameters (labels in successors), we already know that the error function depends linearly on them. The graphic scans values of $\alpha \in (0.0, 1.0)$ and values of $\beta \in [0.0, \min(2\alpha, 2(1-\alpha))]$ with small enough step size and a piecewise linear discriminator. One can notice from Fig. 6 that for any given β, the minimum value of E_S is reached for the same α value. That is, the value of α corresponding to the E_S minimum for *any* β is generally close to the value of α corresponding to the global minimum. The affirmation is true also for $\beta = 0$. Thus it is possible to decompose the search in the space of the four parameters. As result, all four parameters are more efficiently found than if a non-linear optimization technique were used for simultaneous search of all the parameters.

2.4.3. Stop splitting

Splitting should be stopped as soon as it is almost certain that further development of a node would be useless. Because the grown trees are pruned afterwards, these stopping conditions have little influence on the accuracy of the tree, with the remark that too crude conditions would lead to excessively small trees suffering from high bias. On the other hand, over-relaxed thresholds of the stopping conditions could generate unnecessarily large trees that take unneeded time to build. Nevertheless, this is the price we prefer to pay in order to obtain accurate trees after pruning. Hence, we adopt less drastic conditions so as to obtain at this stage large enough trees. The conditions are: (i) limitation of the cardinality of the local growing set defined as $\sum_{o\in S}\mu_S(o)$; (ii) limitation of the node squared error computed as $\sum_{o\in S}\mu_S(o)[\mu_C(o)-L_S]^2$, where L_S represents the label of fuzzy node S; (iii) limitation of the squared error reduction provided by the best splitting of the node.

2.4.4. Multiple classes

The soft decision tree approach is suited for having a single output class C, the result for the complementary class being deduced as $\hat{\mu}_{\bar{C}}(o)=1-\hat{\mu}_C(o)$ (in the given example, "insecure" with the degree 0.76 means "secure" with the degree 0.24). For problems with more than two classes, a forest of *SDTs* is built, each tree being dedicated to the recognition of a single class against the union of all the other classes [33]. Suppose each tree from the forest returns a value of the membership degree to one single class C_i for a given object o, $\hat{\mu}_{C_i}(o)$, and suppose also that the costs of misclassification are independent of the type of error, then the overall elected class for object o is the one with the maximal degree $\hat{\mu}_{C_i}(o)$ over the forest (majority vote): $C^*=\arg\max_{C_i}\hat{\mu}_{C_i}(o)$.

In the more general case where misclassification costs are not uniform, we could adapt this scheme by interpreting these class-membership degrees as (estimates of) conditional probabilities to belong to the different classes and by choosing the class that minimizes the expected misclassification cost estimated for a given object.

2.5. SDT pruning

Pruning is a standard procedure within tree-based models. Its goal is to provide a good compromise between a model's simplicity and its predictive accuracy, by removing irrelevant parts of the model. By pruning a tree, the new complexity of the model is automatically identified without the need to a priori establish thresholds for the stopping conditions which could be sensitive to problem specifics. Pruning also enhances the interpretability of a tree, a simpler tree being easier to interpret. The pruning algorithm which is an important part of our SDT method is further detailed below.

Objective: Given a complete SDT and a pruning sample of objects *PS*, find

$$\arg \min_{\text{subtrees of SDT}} MAE_{PS} \tag{6}$$

i.e. find the subtree of the given SDT with the best mean absolute error (*MAE*) on the pruning set among all subtrees that could be generated from the complete SDT.

A subtree of an SDT is a tree obtained from it by *contracting* one or several of its test nodes, i.e. by replacing these nodes by a terminal node. The total number of different subtrees of a given tree grows exponentially with the tree complexity, which makes an exhaustive search to find the best subtree impossible in most practical applications. We therefore apply a classical strategy used in

tree pruning, which consists in considering as candidate subtrees only a sequence of *nested* subtrees. Note that the number of candidate trees in such a nested sequence is necessarily bounded by the number of test nodes of the complete tree, i.e. is linear in the tree complexity.

Strategy: The pruning procedure takes the following three steps:

- *Test nodes sorting* by increasing order of their relevance. The relevance of a test node S labeled L_S is assessed by the squared error estimate computed at the growing stage with $E_S = \sum_{o \in S} \mu_S(o) [\mu_C(o) - L_S]^2$. Each node preceded in this ordered list by one of its parents is removed from the sequence, since a node is pruned together with the pruning of its parents. The list is invariably closed by the root node.
- *Subtrees sequence generation.* The previous list gives the order in which the critical nodes are contracted. There will be as many trees in the sequence as there are critical nodes in the list. At each step, the first node in the list is removed and contracted, and the resulting tree is stored in the trees sequence. Finally, we obtain a sequence of trees in decreasing order of complexity. During this process, we start with the complete tree which is first tested on the *PS* to compute its *MAE*. Subsequently, each time a new tree is inserted in the sequence, we use efficient incremental formulas to update its *MAE*. Thanks to these formulas, the computational complexity of the pruning algorithm is essentially linear in the complexity of the initial tree (see Section A.2 of Appendix A).
- *Best subtree selection.* We use the so-called "One-standard-error-rule" [9,57] to select a tree from the pruning sequence. This rule consists of evaluating the predictive accuracy of each tree in the sequence in some fashion (in our case, we use the *PS* to get an unbiased estimate of the *MAE*, together with its standard error estimate), and then selecting among the trees not the one of minimal *MAE* but rather the smallest tree in the sequence whose *MAE* is at most equal to $\min\{MAE\} + \sigma_{MAE}$, where σ_{MAE} is the standard error of *MAE* estimated from the *PS*. Notice that, if the *PS* size is very large this rule tends to select the tree of minimal *MAE* (since $\sigma_{MAE} \to 0$ with pruning sample size). On the other hand, for small size of *PS*, this rule tends to slightly over-prune the tree.

2.6. SDT tuning

Tree growing together with tree pruning may be seen as a structure identification phase. A parameter identification phase may also be provided in order to improve the generalization capabilities of the final tree. During structure identification phase, tree parameters are determined locally on the basis of the information available in the local growing samples. A global approach may better tune them, aiming at a global minimum. We developed two optimization procedures respectively, called refitting and backfitting. Based on linear least squares, refitting optimizes only terminal nodes parameters being very efficient and accurate. Based on a Levenberg-Marquardt non-linear optimization technique, backfitting optimizes all model free parameters, thus being more time consuming and in principle more accurate.

2.6.1. Tree refitting

Let us consider the vector $[q_j] = L_j$ of all the labels of the terminal nodes of the tree, $j = 1 \ldots K+1$, where K represents the number of tree test nodes (in a binary tree, the number of terminal nodes is

always the number of test nodes plus one). We also note $[\hat{Y}_i] = \hat{\mu}_C(o_i)$, $[Y_i] = \mu_C(o_i)$ and $[M_{ij}] = \mu_{S_{L_j}}(o_i)$ with $i = 1 \dots \|RS\|$, $j = 1 \dots K+1$, where RS represents the set of objects used for the refitting stage.

Objective: Given SDT and a refitting sample of objects RS, find SDT parameters q^* so as to minimize the squared error function

$$\|Y - \hat{Y}\|^2, \tag{7}$$

where $[\hat{Y}] = [M][q]$ in conformity with Eq. (3) and q denotes the labels in SDT terminal nodes.

Solution: This optimization problem may be solved by matrix inversion. The solution is

$$[q^*] = [[M]^{\mathrm{T}}[M]]^{-1}[M]^{\mathrm{T}}[Y]. \tag{8}$$

This is a way of tuning only the labels in terminal nodes. It is global but still suboptimal, because not all the free parameters of the model are updated. It is fast, since the refitting set RS is composed of objects used for growing and pruning, for which membership degrees in test nodes have already been computed, thus matrix $[M]$ is already settled. Note that $[M]^{\mathrm{T}}[M]$ is a $(K+1)\times(K+1)$ matrix. All in all, the computation of the solution is in principle cubic with respect to the complexity of the pruned tree and linear in the sample size. Nevertheless, in the case of small to moderate tree complexities the dominating term corresponds to the computation of the $[M]^{\mathrm{T}}[M]$ matrix, which is only quadratic with respect to the tree complexity.

2.6.2. Tree backfitting

Objective: Given $SDT(q)$ and a backfitting sample of objects BS, find the set of parameters q^* so as to minimize the squared error function

$$E(q) = \sum_{o \in BS} [\mu_C(o) - \hat{\mu}_C(o,q)]^2, \tag{9}$$

where q denotes all $3K+1$ free parameters of the $SDT(q)$, namely: (i) the parameters defining the soft transition region at every test node fuzzy partitioning, i.e. threshold α_i and width β_i, $i = 1 \dots K$, and; (ii) the labels in all the terminal nodes, i.e. L_j, $j = 1 \dots K+1$.

Strategy: According to Eq. (3), the model $\hat{\mu}_C$ is linear in its terminal nodes parameters (labels) and nonlinear in its test nodes parameters (thresholds and widths). The model is continuous and differentiable with respect to its free parameters almost everywhere in the case of piecewise linear discriminators. We implemented Levenberg-Marquardt non-linear optimization technique, considered the standard of non-linear least squares functions minimization [35]. It has the advantage of needing only the computation of the output gradients with respect to parameters. They can be obtained efficiently using back-propagation. Mathematical details of the method are presented in Section A.3 of Appendix A together with the algorithm.

The output gradients with respect to parameters are given by formulas like

$$\frac{\partial \hat{\mu}_C(o)}{\partial L_j} = \mu_{S_{L_j}}(o), \quad j = 1 \dots K+1, \tag{10}$$

$$\frac{\partial \hat{\mu}_C(o)}{\partial \alpha_i}, \frac{\partial \hat{\mu}_C(o)}{\partial \beta_i} = f\left(\frac{\partial \hat{\mu}_C(o)}{\partial v_i}\right), \quad i = 1 \dots K. \tag{11}$$

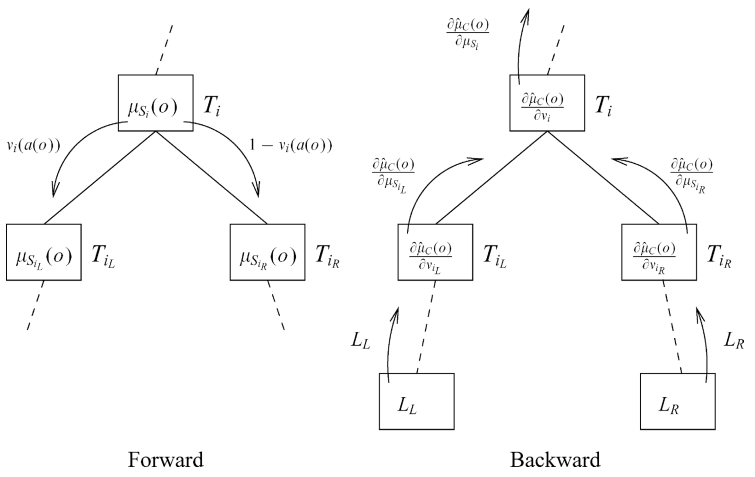

Fig. 7. Computation of partial derivatives.

The most delicate part of the optimization procedure is the computation of the partial derivatives $\partial\hat{\mu}_C(o)/\partial v_i$, $i=1\ldots K$. Fig. 7 presents intuitively the two stages of the back-propagation algorithm for one object o: forward and backward. At the *forward* stage, we start from the root node, and for every test node T_i we compute discriminator $v_i(a(o))$ based on current parameters q. We send this information to the two successors, so as to compute $\mu_{S_{i_L}}(o)=v_i(a(o))\mu_{S_i}(o)$ at left successor T_{i_L}, and $\mu_{S_{i_R}}(o)=[1-v_i(a(o))]\mu_{S_i}(o)$ at right successor T_{i_R}, in conformity to the tree semantics. At the *backward* stage, we start from the terminal nodes and we send backward the labels. Every non-terminal node T_i thus receives $\partial\hat{\mu}_C(o)/\partial\mu_{S_{i_L}}$ from its left successor T_{i_L} (which equals label L_L if this successor is a terminal node) and $\partial\hat{\mu}_C(o)/\partial\mu_{S_{i_R}}$ from its right successor T_{i_R} (which equals label L_R if this successor is a terminal node). T_i node also disposes of $\mu_{S_i}(o)$ and $v_i(a(o))$ computed at forward stage. Thus, we may calculate the partial derivative we searched for at node T_i

$$\frac{\partial\hat{\mu}_C(o)}{\partial v_i}=\mu_{S_i}(o)\left[\frac{\partial\hat{\mu}_C(o)}{\partial\mu_{S_{i_L}}}-\frac{\partial\hat{\mu}_C(o)}{\partial\mu_{S_{i_R}}}\right],\quad\text{and} \tag{12}$$

$$\frac{\partial\hat{\mu}_C(o)}{\partial\mu_{S_i}}=v_i(a(o))\frac{\partial\hat{\mu}_C(o)}{\partial\mu_{S_{i_L}}}+[1-v_i(a(o))]\frac{\partial\hat{\mu}_C(o)}{\partial\mu_{S_{i_R}}}. \tag{13}$$

Eq. (13) represents the quantity that node T_i sends at upper nodes for further computations. In this way, partial derivatives are obtained in a time complexity of $\mathcal{O}(\|BS\|\cdot K)$, thus linearly in the size of the backfitting set and in the tree complexity. Note that the presented formulas are also applicable to non-binary trees and are independent of the discriminator shape.

2.7. Variance and bias studies

If one adopts a regression algorithm and a squared-error type function, then the expected value of the model error can be decomposed in three components [19,20,57]: the *residual error* term, i.e. the lower bound on the expected error of the learning algorithm given the learning task, called also

the Bayes error, the squared *bias* term reflecting the persistent or systematic error that the learning algorithm is expected to make on average when trained on sets of the same size, and the *variance* term measuring the sensitivity of the learning algorithm to changes in the training data and capturing random variation in the algorithm from one learning set to another of the same size. The first term is linked to the analyzed data, since it is the expected error of the best possible model (called Bayes model) and cannot be influenced by a change in the learning algorithm. However, modifying the learning algorithm in some way affects the other two terms, bias and variance, generally in opposite directions. Decision trees are recognized as learning algorithms with low bias and high variance. Increasing tree complexity will decrease bias and increase variance. Reducing variance translates in improved accuracy, provided that it is not obtained at the expense of a much higher bias.

Objective: From this point of view, the behavior of our SDT algorithm has been studied in detail. The experiments included two different variance studies: (i) a global variance study, in order to see how the variance changes in the SDT case and to compare it to the crisp regression tree case; (ii) threshold parameter variance and bias studies versus crisp trees, as a measure of the variability of the threshold in a particular test node of the SDT.

Definitions: In the following we use a probabilistic framework, in which U (the universe of all possible objects) denotes the sample-space on which a probability measure is defined, and where the attributes a_i and output μ_C are considered as random variables (respectively, continuous and discrete). Similarly, the output computed by a fuzzy tree is also a random variable defined on U, specified as a function of the attributes. Moreover, we consider that a LS used in order to construct a fuzzy tree is a sample of independent and identically distributed objects drawn from this probability space. Let us then denote by $\hat{\mu}_{C,LS}(o)$ the output estimated by a SDT built from a random learning set LS of size N at a point $o \in U$ of the universe, leaving the fact that actually $\hat{\mu}_{C,LS}(o) = \hat{\mu}_{C,LS}(a_1(o), \ldots, a_n(o))$ implicit. We notice that this output is "doubly" random, in the sense that it depends both on o and on the random LS. Then the *global variance* of the SDT learning algorithm can be written as

$$Var_{SDT} = E_U\{E_{LS}\{(\hat{\mu}_{C,LS}(o) - E_{LS}\{\hat{\mu}_{C,LS}(o)\})^2\}\} \tag{14}$$

where the innermost expectations are taken over the set of all learning sets of size N and the outermost expectations over the whole input space. This quantity reflects the variability of the model induced by the randomness of the learning set used.

Denoting by α_{LS} the threshold parameter at a given node of a SDT built from a random learning set LS of size N, the *parameter bias* of the SDT represents the error the parameter is having in average when SDT is trained on learning sets of the same size, whereas the *parameter variance* of the SDT reflects the variability of the parameter induced by the randomness of the learning set used. They can be written as

$$b_\alpha = E_{LS}\{\alpha_{LS}\} - \alpha_{\text{asymptotic}}, \quad Var_\alpha = E_{LS}\{(\alpha_{LS} - E_{LS}\{\alpha_{LS}\})^2\} \tag{15}$$

where $\alpha_{\text{asymptotic}}$ is a reference value.

Experiments: In order to compute the expectations over LS, E_{LS}, we should draw an infinite number of learning sets of the same size and build *SDTs* on them. We make the compromise of randomly sampling without replacement the available learning set LS into a number of q learning samples LS_i of the same size, $i = 1, \ldots, q$. Notice that the size of the sub-samples LS_i should be

Table 1
Datasets—Part I

Dataset	# Attributes	# Classes	# Samples	$\|LS\|$	$\|TS\|$	Pe^{Bayes}
Omib	6	2	5000	4000	1000	0.0
Twonorm	20	2	3998	3000	998	2.3
Waveform	21	3	5000	4000	1000	13.2

significantly smaller than the size of the *LS* from which they are drawn. For the expectation over the input space E_U we choose to sum up over the whole test set *TS*.

3. Empirical results

In order to consistently validate our method, we report here two complementary protocols of results. Part I studies the behavior of the proposed SDT method on 3 large databases of several thousand objects since in data mining analyses we are confronted with large amounts of data. Model accuracies are estimated by averaging multiple holdout estimates. Global model variance and parameter variance and bias studies are also done on these datasets. Part II applies SDT method on 11 well known datasets from the UCI Repository and its performance is evaluated using multiple 10-fold cross-validations and statistical significance tests. Here the goal is to compare SDT with standard methods on standard datasets, since from the point of view of data mining these databases present little interest, being very small, not bigger than hundreds of objects.

We compare the soft decision tree with three interpretable tree-structured-inductive methods: standard C4.5 [37] decision trees of Quinlan, standard CART [9] regression trees of Breiman and Wehenkel's method of decision trees published in [57,58], called in what follows ULG method. The last two methods are implemented at University of Liege, Belgium and the first one is C4.5 Release 8, taken from the http address: www.cse.unsw.edu.au/∼quinlan/. CART and ULG trees were completely grown and then pruned. ULG method uses an optimally backward post-pruning in conformity with [55]. CART and SDT were trained for a regression goal. Afterwards, the results were translated in a crisp classification. ULG and C4.5 were trained directly on the crisp classification goal. For multiple-class datasets, forests of CART and SDT were built. The compared methods are all run on the same machine and programming language (lisp), with identical growing, pruning and test samples, excepting C4.5 which is written in C language and does not need a pruning set *PS*, since its pruning is based on estimates of the error rate on unseen cases computed on GS (see [37]).

3.1. Part I

3.1.1. Data sets

Table 1 describes the first 3 datasets used. The last column gives the Bayes error rate, i.e the minimal error one can hope to obtain given the learning task on every dataset. The attributes in datasets are all numerical. The three datasets investigated and the associated *classification*

tasks are:

(i) *Omib* [57] is an electric power system database simulated at University of Liège, Belgium, already described in Section 2.2. It does a security assessment task. The input space is defined by 6 attributes representing pre-fault operating conditions of the OMIB system. The output reflects the system security state after a short-circuit fault occurrence. The continuous output is defined as the fuzzy class of Fig. 2.

(ii) *Twonorm* [8] is a database with two classes and 20 attributes. Each class is drawn from a multivariate normal distribution with unit covariance matrix. The optimal separating surface is an oblique plane, hard to approximate by the multidimensional rectangles used in a crisp tree. We formulated the numerical output as a 0/1 function.

(iii) *Waveform* [9] is an artificial three-class problem based on three waveforms. Each class consists of a random convex combination of two waveforms sampled at integer values with noise added. 21 numerical attributes are explaining the output. Each class is learned against the union of the other two. The numerical output is defined for every subproblem as a 0/1 function. Then the results are aggregated as shown in Section 2.4.4 and a final symbolic class is pulled out.

3.1.2. Protocol

For the omib and twonorm data sets, the reported results (complexity, error rate, global variance, parameter variance and bias) were obtained by averaging over a number of $q = 20$ trees built on randomly chosen (GS, PS) pairs, where GS and PS are mutually exclusive sets of identical size drawn from the complete LS (see Table 1). For waveform data, only results concerning single forest building on a single (GS, PS) pair are reported. In order to study also the evolution of SDT with the learning set size, experiments were conducted on different GS sizes. Note that we considered equally sized growing and pruning sets, i.e. $\|GS\| = \|PS\|$, thus $\|LS\| = 2\|GS\|$, since there is a sufficiently large number of instances for the three databases that allows to not limit us to sets $\|GS\| > \|PS\|$.

3.1.3. Comparing soft decision tree with other methods

Table 2 reports results for C4.5, ULG, CART, refitted SDT(R) and backfitted SDT(B), in terms of model complexity (the number of test nodes) after pruning and test set error rate. Certainly, SDTs are the most accurate among the investigated inductive learning tools, be it in the form of refitted or backfitted version. The most complex are CART for omib, SDT for twonorm and C4.5 for waveform and the less complex is ULG in all three cases. The SDT error gain is the most evident compared with the other methods for twonorm data. For all three datasets, SDTs built for small growing sets (of 250 objects for example) give more accurate results than C4.5, ULG or CART built for the largest growing sets (of 1500 or respectively 2000 objects).

3.1.4. Accuracy

Table 3 shows the impact of the four stages of SDT building, i.e. growing (G), pruning (P), refitting (R), backfitting (B), on the method accuracy and on the model complexity. The first part of Table 3 displays test set error rates and the following observations can be made. Pruning does not have a positive effect on the accuracy for omib and twonorm datasets contrary to waveform dataset. Refitting always improves the accuracy. Backfitting has different effects on the SDT results, depending on the database. In the case of omib and waveform databases there is small improvement

Table 2
Comparing complexity and classification error rate

DB	GS size	Complexity				Error rate (%)				
		C4.5	ULG	CART	SDT	C4.5	ULG	CART	SDT (R)	SDT (B)
	50	3.8	1.4	3.4	20.2	19.5	21.6	19.2	12.2	11.6
	250	12.4	7.7	22.2	52.1	11.2	11.9	11.0	4.9	4.7
Omib	500	20.1	12.2	45.9	52.5	10.1	10.5	9.2	4.6	4.2
	1000	33.8	24.1	93.0	54.9	8.3	8.1	7.6	4.2	4.0
	1500	46.0	37.5	164.0	60.2	7.4	7.9	7.1	4.3	3.9
	2000	56.6	40.4	223.8	59.2	7.0	7.0	6.5	4.4	4.1
	50	3.6	1.6	2.9	9.8	28.7	32.3	29.3	24.7	24.8
	250	14.6	9.1	13.2	53.1	22.9	24.1	23.2	14.8	14.7
Two	500	28.1	17.5	26.1	71.5	20.4	22.4	20.7	10.3	10.5
Norm	750	39.4	33.7	38.2	81.7	19.0	19.9	19.9	8.3	7.6
	1000	51.9	41.7	53.9	83.4	18.1	19.8	19.0	7.3	6.1
	1500	74.5	56.4	77.9	88.7	17.4	19.5	18.5	6.4	4.1
	50	9	2	2	2	48.0	36.0	29.5	25.6	28.2
	250	22	10	4	25	26.4	28.4	26.3	21.0	21.9
Wave	500	43	13	12	24	33.4	28.8	23.9	19.9	19.2
form	1000	66	50	26	44	26.1	26.3	25.4	18.0	17.8
	1500	113	21	35	38	25.9	26.3	21.6	17.5	16.8
	2000	135	25	52	50	24.0	28.0	23.4	17.9	16.6

Table 3
SDT accuracy and complexity (before and after pruning)

DB	GS size	Error rate (%)				Complexity	
		G	P	R	B	Before	After
	50	12.5	12.9	12.2	11.6	61.0	20.2
	250	7.5	8.1	4.9	4.7	110.0	52.1
Omib	500	6.7	7.1	4.6	4.2	113.1	52.5
	1000	6.0	6.7	4.2	4.0	121.7	54.9
	1500	5.5	6.3	4.3	3.9	124.2	60.2
	2000	5.5	6.3	4.4	4.1	126.2	59.2
	50	23.6	25.6	24.7	24.8	41.9	9.8
	250	16.1	17.1	14.8	14.7	159.0	53.1
Twonorm	500	13.2	13.6	10.3	10.5	229.4	71.5
	750	11.2	11.2	8.3	7.6	256.7	81.7
	1000	9.7	10.0	7.3	6.1	254.1	83.4
	1500	8.9	9.1	6.4	4.1	245.3	88.7
	50	30.7	25.5	25.6	28.2	58	2
	250	23.2	22.4	21.0	21.9	104	25
Waveform	500	22.5	21.0	19.9	19.2	170	24
	1000	19.6	19.6	18.0	17.8	192	44
	1500	20.9	18.8	17.5	16.8	176	38
	2000	19.7	18.5	17.9	16.6	201	50

Table 4
Comparing CPU times

DB	GS size	CPU time CART	ULG	SDT(R)	SDT(B)	cy
Omib	50	1.4 s	0.8 s	9.0 s	10.0 s	18
	250	2.5 s	1.7 s	25.0 s	2.2 m	32
	500	6.2 s	2.5 s	40.0 s	5.9 m	50
	1000	10.2 s	4.3 s	1.3 m	6.8 m	23
	1500	16.3 s	6.3 s	2.0 m	6.6 m	28
	2000	26.0 s	7.6 s	3.3 m	13.1 m	31
Twonorm	50	0.5 s	1.0 s	8.7 s	25.0 s	51
	250	1.1 s	2.5 s	36.6 s	2.6 m	51
	500	2.0 s	4.4 s	1.7 m	9.6 m	50
	750	2.5 s	6.6 s	3.4 m	10.0 m	50
	1000	5.4 s	11.6 s	4.2 m	14.6 m	50
	1500	5.6 s	16.0 s	9.8 m	22.2 m	46
Waveform	50	1.5 s	1.0 s	12.0 s	16.9 s	22
	250	3.0 s	2.5 s	44.5 s	3.3 m	41
	500	10.0 s	5.3 s	2.0 m	3.1 m	18
	1000	10.8 s	9.2 s	4.1 m	14.2 m	31
	1500	15.4 s	14.0 s	6.4 m	12.6 m	22
	2000	29.8 s	18.7 s	9.6 m	28.1 m	29

with respect to refitted tree results. In the case of twonorm data, the global optimization clearly contributes a lot to the accuracy improvement with respect to refitting at large growing samples (compare 4.1% with the 2.3% Bayes minimum). At smaller growing samples backfitting worsens accuracy, due to an overfitting phenomenon. The evolution of performance with the growing sample size is evident for the three sets of data. As expected, the larger the learning sample, the better the performance. However, the improvement of the trees error rate saturates for large values of growing sample sizes. Notice that even without any refitting or backfitting optimization, SDT results are significantly more accurate than crisp trees results (compare columns G and P of Table 3 with C4.5, ULG and CART columns of Table 2).

3.1.5. Model complexity

The second part of Table 3 displays SDT complexities before and after pruning. As a multi-class problem, for the waveform data, the complexities displayed are the average over the forest built in every case. One may observe the drastic reduction of tree complexity after pruning. Correlating this with the results concerning accuracy, we conclude that the pruned trees are always preferable to completely grown ones, since they offer much reduced complexity and comparable predictive accuracy. Of course, during the refitting and backfitting tree stages the tree complexity remains frozen, since the model structure is not adjusted anymore at this stage.

3.1.6. CPU times

Table 4 gives an idea about CPU times of CART, ULG, refitted (R) and backfitted (B) SDT. They are linked to a Unix Pentium III, 1.0 GHz, 512 MB machine and to Lisp programming language

Table 5
Example of CPU times when building a SDT for Omib, $\|GS\| = 2000$

Growing	Testing	Pruning	Testing	Refitting	Backfitting	Testing
94 s	5 s	32 s	2 s	48 s	10.1 m	2 s

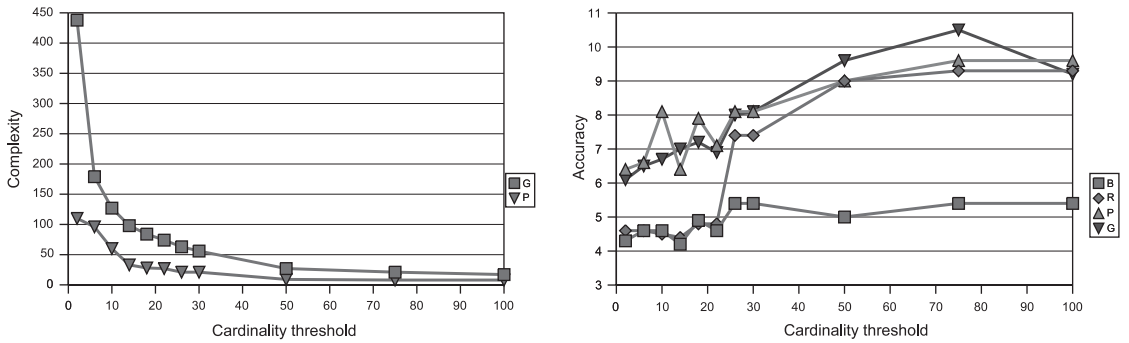

Fig. 8. How accuracy and complexity differ with cardinality threshold parameter for omib, $\|GS\| = 2000$.

and represent a single typical run. For waveform data the time needed to get the whole forest is indicated. The CPU time results reflect a big gap between SDTs and the other methods. This is the price paid for having a more accurate and still interpretable tool in the same time. The number of cycles of non-linear optimization needed to backfit the tree in every case (cy) is also reported. By cycle we denote every passing through the forward step of the optimization procedure. For waveform data, the average number of cycles over the whole forest is indicated. We observe that the backfitting does not need many iterations to converge toward a satisfactory solution. Optimizing by refitting takes less time than backfitting optimization, but as Table 3 shows, the precision is generally better for backfitting. This leads to the idea of a compromise one has to make between the accuracy and the efficiency for a given problem when using SDTs, since more optimal solutions are more costly in terms of CPU times. This compromise is especially valuable when the data we analyze seem to positively respond to the global optimization process, as in the case of the twonorm database, i.e. when the possible gain in precision could cover the time loss. Table 5 provides an example of the distribution of the CPU times for the different tasks when building a soft decision tree, for omib data and a growing set size of 2000 objects.

3.1.7. Choice of algorithm parameters

The results reported in this article are obtained for soft decision trees using piecewise linear shape of discriminators. The parameters for the stopping conditions are: (i) the cardinality of a local growing set in a terminal node is limited to 1% of the *GS* size; (ii) the node squared error is limited to 10^{-4} in absolute value; (iii) the squared error reduction provided by the best next splitting of the current node is limited to 10^{-2}.

The first threshold is the one with the greatest influence on the complexity of the grown tree. In order to avoid troubles in choosing it, one should leave it relaxed as we did. Fig. 8 shows the

Table 6
Comparing global variance

DB	GS size	Var_{CART}	$Var_{SDT(R)}$	$Var_{SDT(B)}$
Omib	50	0.043301	0.055054	0.024785
	250	0.023058	0.005354	0.004777
	500	0.017687	0.002559	0.002264
	1000	0.013400	0.001253	0.001000
	1500	0.011501	0.000971	0.000835
	2000	0.009871	0.000825	0.000540
Twonorm	50	0.144617	0.093214	0.102976
	250	0.132754	0.056538	0.057105
	500	0.123566	0.035461	0.035188
	750	0.120671	0.026806	0.024246
	1000	0.116515	0.020694	0.018158
	1500	0.113813	0.016644	0.010020

influence of this cardinality threshold parameter on accuracy and complexity, at fixed GS, PS, TS sets, with $\|GS\| = 1000$ objects, for the grown (G), pruned (P), refitted (R) and backfitted (B) versions of the SDT and omib data. As we see, below value 10 of the cardinality threshold (corresponding to $1\%\|GS\|$), there is no significant accuracy improvement after backfitting or refitting optimization, whereas the model complexity starts suddenly increasing around that value.

The other two thresholds for stopping have little influence on the tree complexity or accuracy as the empirical studies show. The number of evaluations in Fibonacci search was 5. The parameters of the Levenberg-Marquardt algorithm were: 10^{-3} for the starting value of the algorithm factor λ_{init}, 10 for its step of adjustment λ_{step}, $10^{-5}\|BS\|$ for the threshold value of the error gain $\Delta E_{\min}$, 50 for the maximum allowed number of cycles. Increasing the number of evaluations in Fibonacci search or decreasing $\Delta E_{\min}$ in Levenberg-Marquardt algorithm would produce slower algorithms with only slightly improved results, whereas the opposite case could make the algorithms suboptimal.

3.1.8. Global variance

Table 6 displays global variance for SDT and CART regression trees, for omib and twonorm datasets. Both refitted (R) and backfitted (B) versions of SDT present significantly much smaller global variance than CART regression trees (almost one order of magnitude). Backfitted SDTs present the smallest global variance, with some exceptions for twonorm dataset at small growing sets, where an overfitting phenomenon has already been detected in Table 3. We may conclude that the better accuracy offered by a SDT comes from a reduced overall tree variance. Moreover, the global variance reduces drastically with the growing set size, in strong contrast with the behavior of CART.

3.1.9. Parameter variance and bias

Two different runs of a tree learning algorithm on similar data may happen to choose different attributes in the root node, or different cut points on the same attribute. Refs. [21,22,56] show that classical discretization methods of decision trees actually lead to very high variance of the cut point parameter, even for large training sample sizes, and also that, there is a large variability of the

Table 7
Trade-off bias-variance for the cutting point parameter α. Omib data, attribute *Pu*

GS size	σ_α (MW)				b_α (MW)			
	ULG	CART	SDT(R)	SDT(B)	ULG	CART	SDT(R)	SDT(B)
50	89	74	74	72	57	21	21	19
250	64	36	36	40	38	15	15	20
500	40	22	22	33	25	24	24	9
1000	48	29	29	44	47	22	22	10
1500	29	15	15	27	47	27	27	11
2000	28	13	13	32	35	20	20	5

decision tree structure, visible principally through the variation of the chosen attribute in a particular node. Our simulations on omib and twonorm data reveal that the same two behaviors are encountered at backfitted SDT. *Firstly*, concerning the variability of the model structure, for twonorm dataset, GS = 1000 and 20 attributes, from a total of 20 trees, 35% of the trees have chosen the attribute A18 in the root node, 25% of the trees have chosen attribute A3 and the rest of the trees, attributes A0, A7, A8, A9, A13 and A19. In the same circumstances, 30% of crisp ULG decision trees have chosen attribute A9 at root node, 25%—attribute A18, 15%—attribute A7 and the rest of trees—attributes A0, A3, A4, A8, A19. On the other hand, omib database presents almost no variation at root node, i.e. from a total of 20 trees built on a small growing set of size 50, both ULG and SDT have chosen 19 times the same *Pu* attribute out of the 6 attributes in the root node. Thus this structure variability seems to depend strongly on the data we analyze and on the problem. *Secondly*, concerning the variability of the chosen threshold in a particular node of the tree, Table 7 presents the variance (standard deviation) of the cutting point α parameter in the root node σ_α, together with its bias b_α, for ULG, CART, refitted (R), backfitted (B) SDT and omib database. Given the variability of the model structure, the same statistics could not be done equally on twonorm database. The asymptotic value for attribute *Pu* threshold in the root node has been determined by a ULG decision tree built on the whole available dataset of 5000 objects: $Pu_{\text{asymptotic}} = 1060.5MW$. Note that the range of variation of attribute *Pu* is $600MW$ and its standard deviation σ_{Pu} is $169MW$.

Due to the adopted fuzzy partitioning approach, the chosen attribute in the root node of a non-backfitted soft decision tree and its α value will always coincide with the ones chosen in the root node of a CART regression tree. For this reason, the variance σ_α as well as the bias b_α are identical for CART and refitted soft decision trees (see columns CART and SDT(R) of Table 7). Once backfitted, a soft decision tree changes its thresholds in all the tree nodes, and thus also its parameter variance and bias (see columns SDT(B) of Table 7). One may observe that a non-backfitted soft decision tree, identically to a CART regression tree, presents less parameter variance (almost 50%) and less parameter bias than a ULG decision tree. By backfitting, parameter variance increases whereas parameter bias decreases. The explanation resides in the fact that by globally optimizing, the location parameters are not any more restricted to fall in the range (0;1) and therefore they are more variable with respect to the average. In fact, backfitting may be seen as a relaxation of the hypothesis space, that generally lowers the parameter bias, i.e. the systematic error the model transmits to the location parameter. Nevertheless, backfitted soft decision trees present generally less parameter variance than

Table 8
Datasets—Part II

Dataset	# Attributes	# Classes	# Samples
Balance	4	3	625
Glass	9	6	214
Heart	13	2	270
Ionosphere	34	2	351
Iris	4	3	150
Monks-1	6	2	432
Monks-2	6	2	432
Monks-3	6	2	432
Pima	8	2	768
Sonar	60	2	208
Wine	13	3	178

Table 9
Classification error rates and standard deviations for a 10 times 10-fold cross-validation

Database	C4.5	ULG	CART	SDT (R)	SDT (B)
Balance	(−)21.74 ± 0.8	(−)22.29 ± 0.7	(−)21.01 ± 0.8	14.33 ± 0.5	17.10 ± 0.9
Glass	(−)31.42 ± 2.0	31.22 ± 2.4	(−)32.27 ± 1.6	28.91 ± 1.5	29.09 ± 2.2
Heart	(+)21.98 ± 2.1	(−)27.78 ± 2.0	(−)27.19 ± 2.0	25.81 ± 2.0	25.59 ± 1.9
Ionosphere	10.65 ± 1.3	9.96 ± 0.8	11.05 ± 0.9	10.56 ± 1.0	10.36 ± 1.1
Iris	4.94 ± 0.6	(−)6.13 ± 0.8	(−)6.47 ± 0.8	4.73 ± 0.9	5.00 ± 0.5
Monks-1	(−)25.01 ± 0.0	(−)5.75 ± 2.0	(−)11.65 ± 3.2	17.56 ± 4.3	1.95 ± 2.1
Monks-2	(−)10.56 ± 1.5	(−)3.01 ± 1.3	(−)3.15 ± 0.8	2.61 ± 1.4	1.53 ± 1.2
Monks-3	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
Pima	25.68 ± 1.7	(−)30.22 ± 0.9	(−)29.89 ± 1.2	26.43 ± 1.3	25.57 ± 1.2
Sonar	26.45 ± 2.3	(+)25.27 ± 2.9	(−)29.19 ± 2.0	26.72 ± 1.9	27.44 ± 2.1
Wine	(−)7.27 ± 1.1	(−)6.58 ± 1.1	(−)10.66 ± 1.4	3.52 ± 0.8	3.53 ± 0.7

ULG decision trees. Thus both non-backfitted and backfitted versions of soft decision trees are more stable from the point of view of the chosen parameters in the test nodes than a crisp decision tree. A smaller parameter variance makes the tree easier to interpret. Backfitting a soft decision tree could influence negatively this interpretability.

3.2. Part II

3.2.1. Experiments

For the second part of experiments, designated to compare the predictive accuracy of soft decision trees with respect to C4.5 and ULG decision trees and CART regression trees, we have chosen 11 datasets from the UCI Repository. They are summarized in Table 8. All of them were previously used in other comparative studies. The attributes in datasets are all numerical with no missing values. Table 9 shows comparatively accuracy of C4.5 decision trees, ULG decision trees and CART regression trees, and refitting (R) and backfitting (B) versions of our SDT method. For each pair

Table 10
Results of paired t-tests ($p = 0.05$). x/y indicates method in row is significantly better x times and significantly worse y times than method in column, from a total of 11 datasets

	C4.5	ULG	CART
SDT (R)	5/1	6/1	7/1
SDT (B)	5/1	7/1	9/0

algorithm—dataset, the mean error rate is reported, averaged over *ten* times 10-fold non-stratified cross-validation runs and also standard deviations of the ten are shown. Comparisons between algorithms have been performed across all the datasets using a paired two-sided t-test [17] with significance set at the 5% level. Relative to each algorithm, a $+(-)$ sign means that the error rate of this algorithm is significantly better (worse) than the error rate of the backfitted (B) SDT, according to this significance test. Pruning has been done by all methods directly on the growing set ($PS = GS$).

3.2.2. Discussion of results

Since databases are small this time, there is a risk of overfitting, which can be seen at certain datasets (balance, glass, iris, sonar) where backfitting worsens accuracy with respect to refitting. In spite of this, both refitting and backfitting SDTs offer the best results overall. Table 10 displays results of the statistical tests. x/y indicates method in row is significantly better x times and significantly worse y times than method in column. These results show that classifiers based on soft decision trees generated by our method, be it refitted or backfitted, are significantly more accurate than C4.5 decision trees, ULG decision trees and CART regression trees. Also, between refitting and backfitting results there are some visible differences, but they are not significant. Thus, we may conclude that overall, for small datasets, backfitting may not be necessary, since with refitting which is faster we obtain already significantly better results than C4.5, CART or ULG ones.

4. Discussions

4.1. Possible extensions

The fuzzy discriminator used at the fuzzy partitioning step has an a priori chosen shape. We considered in our simulations the piecewise linear shape, but sigmoidal, Gaussian or triangular shapes may equally be employed (see Fig. 9). Note that in the case of ordered attributes, monotonic discriminator templates are preferred since they yield trees which are easier to interpret. Simulations were conducted on sigmoidal discriminators and test results are comparable to those reported here.

We assumed through the presented algorithm that all the attribute values are numerical. A possible extension of the method would be toward the handling of qualitative attributes. Supposing a_q qualitative attribute with m different values, $a_q : U \to \{s_1, \ldots, s_m\}$, the fuzzy partitioning approach searches for a discriminator function $v : \{s_1, \ldots, s_m\} \to [0 \ldots 1]$, $v = [v_1, \ldots, v_m]$, that minimizes the same error function as in the numerical attribute case, but instead of two parameters there are m parameters to optimize: $v_1, \ldots, v_m$. A non-linear optimization technique or a clustering approach may be used

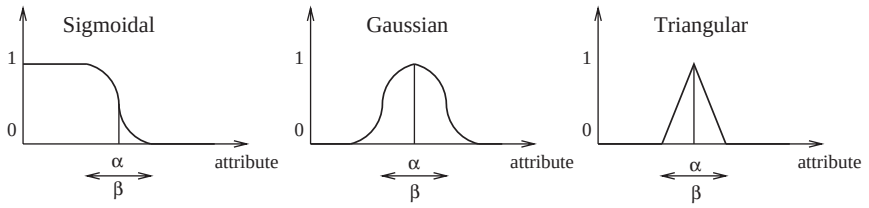

Fig. 9. Example of other discriminator functions.

in order to optimize the m parameters defining v, or they may be determined by simple estimated conditional probabilities in the considered node, as $v_i = P(C|s_i)$ and $1 - v_i = P(\bar{C}|s_i)$.

4.2. Further work

Given the obtained results, there are four directions in which the algorithm could be improved. One is the pruning approach, given that after pruning, the model accuracy does not seem to improve for omib and twonorm data. The relevance measure we use for pruning does not guaranty at all that the obtained sequence of trees includes the best possible sub-trees of the complete tree. Other pruning criteria for sorting test nodes could be: set as relevance measure the total error amelioration produced by the node S, $E_S - E_{S_L} - E_{S_R}$, or the global error on the subtree of node, E_{T_S}, where T_S is the subtree rooted at node S, or simply the local growing sample size of the node, knowing that estimates based on small samples are potentially unreliable (thus we eliminate the node whose error estimate is potentially the least reliable [46]). A second direction for further investigation is the optimal fuzzy partitioning. As the parameter variance study shows, the soft decision trees have, at least in the root node, the same behavior as regression trees concerning the variability of the threshold and the variability of the chosen attribute in a node. Due to the decomposition of the parameters search at the optimal splitting stage, the attribute in a node together with its threshold α are chosen based on a crisp split as in the regression tree case. We could try to simultaneously find all the parameters defining the split by a non-linear optimization technique instead of decomposing the search. A third investigation would concern the refitting during pruning approach, which refits every tree from the sequence of pruned trees, before choosing the best pruned one. So far, experiments reflect accuracy improvement. In this respect, further work should be done in terms of computational complexity, so as to reduce the amount of computations which then becomes quadratic in terms of model complexity before pruning. The last direction for further work concerns CPU times: improvements can be done on the implementation side so as to render the whole method more efficient.

4.3. Three fundamental reasons for increased accuracy of soft decision trees with respect to crisp decision trees

An explanation of why a soft decision tree would give a priori better results than a crisp tree lies in the possibility of overlapping instances. In a crisp tree, the cut-point test performs badly on examples with attribute values close to the cut point [11,19,20]. Two objects that are close to each other in the space of the attributes may happen to be split on separate branches and therefore situated “faraway” one of each other in the output space. On the contrary, within a soft decision

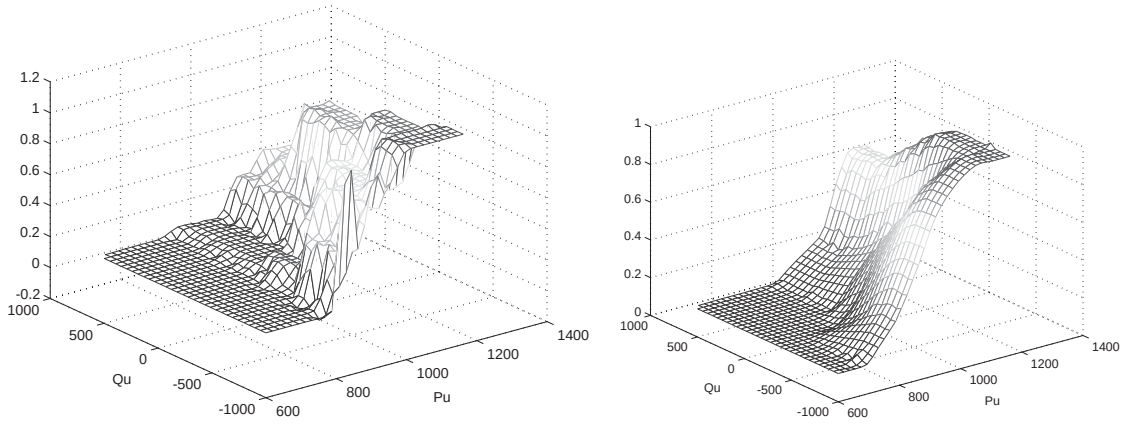

Fig. 10. Regression tree versus soft decision tree output.

tree, two examples close to each other in the space of the attributes are treated in a similar fashion. In this way, soft decision trees decrease bias near the region boundaries and thus the tree errors. As a correlated factor, the noise possibly present in the training data may aggravate this phenomenon, leading to models sensitive to noise. By softening the split threshold, the noise influence becomes less significant for the generalization capabilities of the model.

Another aspect of recursive partitioning of crisp decision trees is the fast decrease of local growing samples when going down into the tree. The local decisions may become too particular to the data, thus leading to overfitting and to high variance [20]. In soft decision trees, the local growing sets are allowed to keep more instances, even if the instances are not all strictly belonging to those sets. Thus, local decisions are more stable in a node as they are established on richer information and the variance linked to this aspect is less significant. The larger the degree of fuzziness in a node, the larger the local growing sets of its successors. See for example, how the soft tree in Fig. 1(b) presents 61 instances in node T4, whereas the regression tree has only 49 instances in the corresponding node T2.

An immediate effect of this growth of local samples size is that a soft tree gives surprisingly good results for very small learning sets, as the empirical studies show, highly out-running thus the crisp trees. Another effect of the increase of the local growing samples is the improvement of the stability of the learner, as [49] states, i.e. the improvement of the ability of the algorithm to produce similar results with respect to small variations in the training set. Decision tree algorithm is a very unstable learning algorithm [18,20] and this may cause the users loss of confidence in the tool as soon as perturbations in data lead to different trees and different understandings of the analyzed problem, even if the trees present high predictive accuracy. Empirical results of Section 3.1.8 showed that soft decision trees present much less global variance than crisp regression trees. Hence, indeed, the stability of the learner is improved.

The third reason for a better accuracy is linked to the continuity of the soft decision tree output. Fig. 10 shows the discontinuous (staircase) character of a regression tree output (left part) versus the smooth output surface of a soft decision tree (right part), both designed for the fuzzy defined class of OMIB database. Soft partitioning together with the way the terminal nodes are aggregated assure the continuity of the soft decision tree output. As a consequence, the tree may be seen as

a smooth parametric model and global optimization techniques may be applied in order to further improve the predictive accuracy of the tree.

4.4. Related work

What we call a soft decision tree is in fact what people brought out in the early nineties under the generic name of fuzzy decision tree inductive learning. The first fuzzy decision tree reference is attributed to Chang and Pavlidis [12], in 1977. Since then, there are more than thirty references on this kind of algorithm.

To make a distinction, the introduction of the fuzzy framework in the decision tree technique is based on two premises. Firstly, one wants to enable the use of decision trees to *manage fuzzy information* in the form of fuzzy inputs, fuzzy classes, or fuzzy rules. Secondly, one realized that the use of fuzzy logic in many learning techniques could *improve their predictive accuracy*. Fuzzy sets are tools to enforce the predictive accuracy with respect to a classic decision tree, while preserving the interpretability character.

Considering the literature about the construction of fuzzy decision trees, chronologically, the following principal directions have been addressed. First, there are crisp decision trees that soften the threshold in a node and the final decision, by means that do not use fuzzy sets, but rather probabilistic frameworks [10,11,31,37], or neural implementations [41]. Then, the fuzzy logic starts to show up as fuzzy decision rules obtained by fuzzifing the crisp rules extracted from a crisp decision tree [13,23,39,45]. Later, fuzzy decision trees appear, either as methods that prefer starting with a crisp tree and once the tree architecture has been obtained, searching for the degree of softness in every node of the built tree [30,44], or as fuzzy decision trees that directly integrate fuzzy techniques during the growing phase [1,4,5,7,15,24,27,28,29,33,38,42,47,48,50–54,59,60]. The induction process sets the node fuzziness as a new node of the tree is being developed. The method presented in this paper belongs to this last category.

In this context, we should also mention another two representative groups of methods that originate in tree-structured inductive ones and are situated in the neighborhood of the fuzzy trees approaches: the fuzzy neural networks or neuro-fuzzy approaches obtained by transforming a crisp decision tree into a neural network which is then fuzzified [16,25], and neural networks derived from crisp decision trees [2,14,26,40,51], approaches useful at the optimization phase of a fuzzy decision tree.

Concerning fuzzy decision trees, many approaches start from the ID3 decision tree algorithm [36] which is not anymore the state of the art. They inherit its weak points. The attributes may be only of symbolic type, they require an a priori partitioning at the beginning or user defined fuzzy sets, stopping criteria do not bother about fitting the tree structure to data, etc.

Pruning is rarely provided and in most cases is not meant to cover large databases applications. Ref. [27] provides a kind of pruning based on Akaike information criterion. Ref. [59] proposes a rule simplification technique applied on the rules obtained by transforming the tree in a rule base and based on the truth level of a fuzzy rule. On the other hand, [3] provides pruning based on grouping words (labels of a linguistic variable), words generated when partitioning the attribute. In [52] one proposes a branch merging algorithm based on a fuzzy value clustering of branches, and [42] performs pruning based on the minimum description length principle. Finally, [4] settles a complete pruning procedure that works well on a large scale application, but at a cost of rather high computational requirements.

Due to the non-continuity and non-differentiability of the model error function, many of the present fuzzy decision tree methods cannot implement a global training or a global final optimization after identifying the model structure. However, there are global optimizations based on genetic algorithms [43], similarity measures [53], or backpropagation of neural networks [15,27,44,47]. In this last case the decision tree initially determines the structure of a neural net (important inputs, network complexity) and then the free parameters are globally optimized within the neural network. The backfitting step we implemented here comes from an idea presented but not implemented in [7]. For the refitting step also we inspired us from [7], there being presented as a sub-step of their global optimization process. And in our knowledge, no other fuzzy decision tree approach implemented such an optimization step.

5. Summary and conclusions

The aim of this paper was to give a global overview of a new soft decision tree methodology. The presented method comes with the following specific contributions:

- It allows the handling of fuzzy classes and numerical valued attributes.
- It works for regression and for classification problems.
- In accord with nowadays data mining practices, the algorithm is conceived to be properly used on high dimensional real world problems, with multiple attributes and training samples, with respect to time required, accuracy and interpretability (the software behind the implementation of soft decision trees has a module for soft decision trees visualization).
- The best degree of fuzziness in every new node of the tree is automatically found as the node comes up, by means of a local optimization procedure. This gives more accurate results than the algorithms with a priori fixed (ID3-based algorithms) or limited choice of the degree of fuzziness.
- A backward pruning technique is implemented for the soft decision tree as a tradeoff between the accuracy of the tree and its complexity. The model complexity is in this way adapted to the data.
- An algorithm for globally refitting the labels at terminal nodes of the tree is presented as a way of efficiently increasing the generalization capabilities of the tree.
- A global optimization procedure is established in order to optimize not only the labels in terminal nodes but also the two parameters characterizing every internal node splitting (threshold and width), with the effect of further increasing tree accuracy, but more time consuming. Experiments show that in the specific case of "small" datasets, backfitting does not come with a significant improvement in accuracy with respect to refitting and may even worsen the accuracy due to overfitting. In the case of "large" datasets, backfitting may contribute significantly to the accuracy improvement with respect to refitting. In all cases, refitting, a simple and new way of tree optimization, gives enough good accuracy with respect to standard crisp trees, demanding much less CPU time than backfitting.
- From a computational point of view, the method is intrinsically slower than crisp tree induction. This is the price paid for having a more accurate but still interpretable classifier.
- A global variance study is conducted for the soft decision tree. We have no knowledge of previous literature that estimates quantitatively variance on fuzzy decision trees. Simulations show that

in practice, the improved accuracy of a SDT with respect to a crisp CART regression tree is explainable by a much lower prediction variance.
- The parameter variance study shows that a non-backfitted soft decision tree presents 50% less parameter variance than a ULG decision tree and also that tree backfitting influences negatively the tree parameter variance.

Finally, two aspects should be kept in mind about soft decision trees. Primarily, they inherit the most attractive quality of crisp decision tree induction: the interpretability. Secondly, they can be significantly more accurate than standard decision and regression trees, as shown by the empirical tests carried out in this paper.

Appendix A.

A.1. SDT growing

A.1.1. Searching for the split location α in a SDT, when $\beta = 0$

As long as the split in a soft decision tree node is considered temporarily crisp, (with $\beta = 0$), the procedure of choosing the threshold α follows the same way of discretizing used in regression trees. The same squared error reduction is used as score function, the only difference being the permanent weighting by membership degrees.

Crisp splitting rule: Given S fuzzy set in a soft decision tree, the best crisp partition for S maximizes over all the possible partitions of all attributes the normalized squared error reduction

$$\max\left[1 - \frac{E_{S_L}}{E_S} - \frac{E_{S_R}}{E_S}\right],$$

where E_S, E_{S_L} and E_{S_R} are the squared error functions at nodes S, S_L and S_R respectively.

When $\beta = 0$, the two sets of objects S_L and S_R of Fig. 5 are mutually exclusive, the piecewise linear discriminator $v(\cdot)$ of Fig. 3 becomes crisp, and the error function E_S of Eq. (4) can be written based on Eq. (5)

$$\begin{aligned} E_S &= \sum_{o \in S_L} \mu_S(o)[\mu_C(o) - L_L]^2 + \sum_{o \in S_R} \mu_S(o)[\mu_C(o) - L_R]^2 \\ &= E_{S_L} + E_{S_R}. \end{aligned}$$

To minimize the error E_S translates thus in minimizing the two errors E_{S_L} and E_{S_R}.

Therefore,

$$\frac{\partial E_{S_L}}{\partial L_L} = 0 \quad \text{and} \quad \frac{\partial E_{S_R}}{\partial L_R} = 0,$$

$$-2 \sum_{o \in S_L} \mu_S(o)[\mu_C(o) - L_L] = 0,$$

$$-2 \sum_{o \in S_R} \mu_S(o)[\mu_C(o) - L_R] = 0,$$

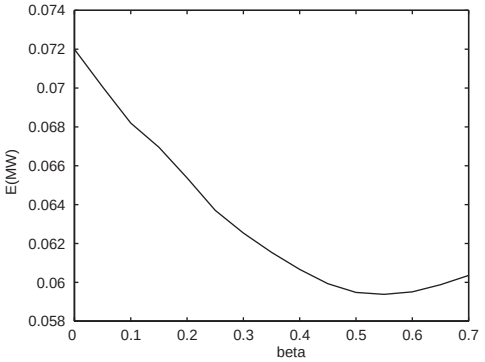

Fig. 11. $E_S = E_S(\beta)$ for the optimal value of α ($\alpha = 0.65$). Omib database.

and finally, the two successors labels are estimated as

$$L_L = \frac{\sum_{o\in S_L} \mu_S(o)\mu_C(o)}{\sum_{o\in S_L} \mu_S(o)}, \quad L_R = \frac{\sum_{o\in S_R} \mu_S(o)\mu_C(o)}{\sum_{o\in S_R} \mu_S(o)}, \tag{A.1}$$

i.e. as weighted average values of the membership degrees to the output class. Introducing these labels estimates in E_{S_L} and E_{S_R}, we get the incremental formulas for computing the errors:

$$E_A = \sum_{o\in A} \mu_S(o)\mu_C^2(o) - \frac{(\sum_{o\in A} \mu_S(o)\mu_C(o))^2}{\sum_{o\in A} \mu_S(o)},$$

where A stands for S_L or S_R. All the sums $\sum_{o\in S_L}$ and $\sum_{o\in S_R}$ that intervene in the preceding formulas are updated once an object changes the part of the split. For example, when the location α changes, there are some objects that are deleted from the left sum $\sum_{o\in S_L}$ and are added to the right sum $\sum_{o\in S_R}$. This avoid computing the two sums from scratch every time α changes.

A.1.2. Fibonacci search for β width

Fibonacci search is an optimal univariate optimization strategy that finds the minimum of a function on an interval only by functions evaluations at points placed according to a Fibonacci sequence, without making use of any function derivatives. The algorithm stops when an a priori imposed number N of function evaluations is accomplished. It is worthy to notice that the Fibonacci search minimizes the number of evaluations needed to find the minimum of a function on a given set of points. The only restriction the function to be minimized on interval $[a, b]$ has to encounter when using Fibonacci search, is to be unimodal, i.e. to strictly decrease between a and the minimum, and to strictly increase between the minimum and b. This is the case in all the effectuated empirical studies: Fig. 11 presents the function to be minimized of Fig. 6 $E_S = E_S(\beta)$ in terms of β once the optimal α location has been settled, in this case $\alpha = 0.65$, for OMIB database.

A.1.3. Optimizing the successor labels at fixed α and β

In order to minimize the error function of Eq. (4), we settle

$$\frac{\partial E_S}{\partial L_L} = 0 \quad \text{and} \quad \frac{\partial E_S}{\partial L_R} = 0.$$

In conformity with Eq. (5)

$$-2 \sum_{o \in S} \mu_S(o) v(a(o)) \{\mu_C(o) - [v(a(o))L_L + (1 - v(a(o)))L_R]\} = 0,$$

$$-2 \sum_{o \in S} \mu_S(o)(1 - v(a(o))) \{\mu_C(o) - [v(a(o))L_L + (1 - v(a(o)))L_R]\} = 0.$$

Solving this linear system in L_L and L_R, we get the formulas for updating labels at every new width β as

$$L_L = \frac{c(o)d(o) - e(o)b(o)}{b(o)^2 - a(o)c(o)} \quad L_R = \frac{a(o)e(o) - b(o)d(o)}{b(o)^2 - a(o)c(o)}$$

where all the terms generically noted $a(o), b(o), c(o), d(o)$ are sums computed in terms of $\mu_S(o)$, $\mu_C(o)$ and $v(a(o))$:

$$a(o) = \sum_{o \in S} \mu_S(o) v(a(o))^2 \quad b(o) = \sum_{o \in S} \mu_S(o) v(a(o))(1 - v(a(o)))$$

$$c(o) = \sum_{o \in S} \mu_S(o)(1 - v(a(o)))^2 \quad d(o) = -\sum_{o \in S} \mu_S(o)\mu_C(o) v(a(o))$$

$$e(o) = -\sum_{o \in S} \mu_S(o)\mu_C(o)(1 - v(a(o))).$$

We could have settled the labels directly by formulas of type (A.1) without making an optimization on them, but any a priori defined formula for the labels would introduce a bias on the final result, thus would conduct to sub-optimality. [7] already observed that using weighted average leads to over-smoothing.

A.1.4. Computational complexity of tree growing

The complexity of SDT growing is upper bounded by $\mathcal{O}(\|a\| \cdot K \cdot \|GS\| \cdot log\|GS\| + const \cdot K \cdot \|GS\|)$, where K is the complete tree complexity, $\|GS\|$ is the number of growing instances and $\|a\|$ the number of candidate attributes. We say it is bounded due to the fact that in the worst case all the *GS* objects are propagated through all the test nodes. The first part of the expression refers to the search of the location parameters in all the test nodes of the tree, the second part to the search of the width parameter together with the successors labels for all the test nodes of the tree. The constant *const* is proportional to the given maximum number of evaluations in Fibonacci search. Note that for a crisp decision tree, the growing complexity is also $\mathcal{O}(\|a\| \cdot K \cdot \|GS\| \cdot \log \|GS\|)$. However, the constant factors are much higher in the case of fuzzy trees than crisp ones.

A.2. SDT pruning

A.2.1. Subtrees error computation

At the beginning of the pruned trees generation procedure, some operations are done. The *PS* objects are propagated forward through the complete tree and for every object o and every tree node S the membership degree of the object to the node $\mu_S(o)$ is computed and the products membership degree—node label $\mu_S(o)L_S$ are stored. Also, the output estimation $\hat{\mu}_C(o)$ for the complete tree is assessed for every object by summing up these products linked to the terminal nodes as in Eq. (3).

Then every time a node N_X from the sorted list of irrelevant nodes is pruned, the output of the new tree for every object belonging to this node is recursively updated by removing from $\hat{\mu}_C(o)$ the terms corresponding to the disappeared old terminal nodes and by adding the term corresponding with node N_X which has become a new terminal node, i.e.

a. $\forall L_j$ below node N_X and $\forall o \in S_{L_j}$ set $\hat{\mu}_C(o) = \hat{\mu}_C(o) - \mu_{S_{L_j}}(o)L_j$
b. $\forall o \in S_{N_X}$ set $\hat{\mu}_C(o) = \hat{\mu}_C(o) + \mu_{S_{N_X}}(o)L_{N_X}$.

This artifice makes the second (and in practice most time consuming) step of the pruning algorithm linear in the number of the test nodes, thus in the model complexity. The procedure stops when there are no more nodes candidate for pruning in the sorted list.

A.2.2. Computational complexity of tree pruning

Since for test node sorting, the squared errors for all the test nodes are evaluated based on already known membership degrees computed for the growing set, this step is upper bounded in complexity by $\mathcal{O}(K \cdot \|PS\| + const \cdot K \cdot \log K)$, where K is the complete tree complexity and $\|PS\|$ denotes the pruning set size. The pruned tree sequence generation has also a bounded computational complexity of $\mathcal{O}(K \cdot \|PS\|)$ due to the use of updating formulas. The tree selection is only proportional with the number of pruned trees obtained in the sequence which is upper bounded by K.

A.3. SDT backfitting

A.3.1. Levenberg-Marquardt optimization

Suppose we are currently at point q_0 in the parameters space and we move to point $q_0 + \delta q$, where δq represents the increment of parameter q at the current iteration. If this displacement δq is small we can expand the gradient of the error function in a Taylor series keeping only two terms:

$$\nabla_q E(q)|_{q_0+\delta q} = \nabla_q E(q)|_{q_0} + \nabla_q^2 E(q)|_{q_0}\delta q. \tag{A.2}$$

$\nabla_q E(q)|_{q_0+\delta q} = 0$ if $q_0 + \delta q$ is the point corresponding to the minimum of E function. Thus Eq. (A.2) translates in

$$[A][\delta q] = [B], \tag{A.3}$$

where we noted

$$A = \frac{1}{2}\nabla_q^2 E(q)|_{q_0} \quad \text{and} \quad B = -\frac{1}{2}\nabla_q E(q)|_{q_0}.$$

On the other hand, as (A.2) may be a poor approximation, the Steepest Descent method proposes to take a step down the gradient and to jump with the increment

$$[\delta q] = constant * [B]. \tag{A.4}$$

Combining Eqs. (A.3) and (A.4), Levenberg-Marquardt algorithm proposes a factor λ that governs the step size and alters only the diagonal elements of the matrix $[A]$, i.e. for all $p = 1 \dots 3K + 1$

$$[A_{pp}] = [A_{pp}](1 + \lambda). \tag{A.5}$$

A.3.2. Deducing involved matrices

The two matrices $[A]$ and $[B]$ are deduced as

$$[A_{pl}] = \sum_{o \in BS} \frac{\partial \hat{\mu}_C(o)}{\partial q_p} \frac{\partial \hat{\mu}_C(o)}{\partial q_l}, \quad p, l = 1 \dots 3K + 1, \tag{A.6}$$

$$[B_p] = \sum_{o \in BS} [\mu_C(o) - \hat{\mu}_C(o)] \frac{\partial \hat{\mu}_C(o)}{\partial q_p}, \quad p = 1 \dots 3K + 1. \tag{A.7}$$

They only need first derivatives of the model in order to be computed (Eqs. (10) and (11)). In the Levenberg-Marquardt method, the second derivative terms for matrix A are considered canceled when summarizing over all the objects in BS.

A.3.3. Backfitting algorithm

Given the nonlinear dependencies, the minimization of the error function must proceed iteratively. The starting values for the parameters are the values available in the tree nodes after the structure identification phase. They are improved at every new trial and the procedure ends when the error function stops decreasing or almost, i.e. when the error gain $\Delta E = E(q) - E(q + \delta q)$ is less than a threshold $\Delta E_{\min}$. Also, if the algorithm did not converge in a certain number of cycles to the imposed $\Delta E_{\min}$, the process stops, but never after a negative ΔE, because this shows that λ has not yet adjusted itself optimally. The all-in-all optimization algorithm that search for the set q^* of all the free parameters such that $E(q^*) = \min E(q)$ given the set BS of objects, the tree $SDT(q)$ corresponding to set q of parameters, the threshold value for the error gain $\Delta E_{\min}$, the starting value for the algorithm factor λ_{init} and its step for adjustment λ_{step} is:

1 set $\lambda = \lambda_{\text{init}}$
2 **Forward**: compute $E(q)$ with Eq. (9)
3 **Backward**: compute first derivatives $\partial \hat{\mu}_C(o, q)/\partial q_p$ with Eqs. (10) and (11) and set matrices $[A]$ and $[B]$ with Eqs. (A.6) and (A.7)
4 alter matrix $[A]$ with Eq. (A.5)
5 solve linear system $[A][\delta q] = [B]$ by Gauss elimination
6 **Forward**: compute $E(q + \delta q)$ with Eq. (9)
7 compute gain $\Delta E(q) = E(q) - E(q + \delta q)$
8 **if** $0.0 \leqslant \Delta E(q) \leqslant \Delta E_{\min}$ **then** $q^* = q + \delta q$; go to **11** (end)
9 **if** $\Delta E(q) < 0.0$ **then** $\lambda = \lambda * \lambda_{\text{step}}$; go to **4** (alter)
10 **if** $\Delta E(q) > \Delta E_{\min}$ **then** $\lambda = \lambda / \lambda_{\text{step}}$; $q = q + \delta q$; go to **3** (backward)
11 Update all others SDT parameters (tests in nodes, …)

References

[1] B. Apolloni, G. Zamponi, A.M. Zanaboni, Learning fuzzy decision trees, Neural Networks 11 (1998) 885–895.
[2] R. Araya, P. Gigon, Segmentation trees: a new help building expert systems and neural networks, Proceedings of Stats, 1992, pp. 119–124.
[3] J.F. Baldwin, J. Lawry, T.P. Martin, Mass assignment based induction of decision trees on words, in: Proceedings of the Information Processing and Management of Uncertainty in Knowledge-Based Systems, Vol. 1, July 6–10, Paris, 1998, pp. 524–531.
[4] X. Boyen, Design of fuzzy logic-based decision trees applied to power system transient stability assessment, Master's Thesis, University of Liège, 1995.
[5] X. Boyen, L. Wehenkel, Fuzzy decision tree induction for power system security assessment, Proceedings of SIPOWER'95, IFAC Symposium on Control of Power Plants and Power Systems, Cancun, Mexico, December 1995, pp. 151–156.
[6] X. Boyen, L. Wehenkel, On the Unfairness of Convex Discriminator Quality Measures for Fuzzy Partitioning in Machine Learning, Technical Report, University of Liege, 1995.
[7] X. Boyen, L. Wehenkel, Automatic induction of fuzzy decision trees and its applications to power system security assessment, Fuzzy Sets and Systems 1 (102) (1999) 3–19.
[8] L. Breiman, Arcing classifiers, Ann. Statist. 26 (3) (1998) 801–849.
[9] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Chapman & Hall, New-York, 1984.
[10] W. Buntine, Learning classification trees, Statist. Comput. 2 (1992) 63–73.
[11] C. Carter, J. Catlett, Assessing Credit Card Applications Using Machine Learning, IEEE Expert, Fall 1987, pp. 71–79.
[12] R.L.P. Chang, Th. Pavlidis, Fuzzy decision tree algorithms, IEEE Trans. Systems Man, Cybern. SMC-7 (1) (1977) 28–35.
[13] Z. Chi, H. Yan, ID3-Derived fuzzy rules and optimized defuzzification for handwritten numeral recognition, IEEE Trans. Fuzzy Systems 4 (1) (1996) 24–31.
[14] K.J. Cios, N. Liu, A machine learning method for generation of a neural network architecture: a continuous ID3 algorithm, IEEE Trans. Neural Networks 3 (2) (1992) 280–291.
[15] K.J. Cios, L.M. Sztandera, Continuous ID3 algorithm with fuzzy entropy measures, Proceedings of The First IEEE Conference on Fuzzy Systems, San Diego, 1992, pp. 469–476.
[16] K.J. Cios, L.M. Sztandera, Ontogenic neuro-fuzzy algorithm: F-CID3, Neurocomputing 14 (1997) 383–402.
[17] T.G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, Neural Comput. 10 (7) (1998) 1895–1924.
[18] T.G. Dietterich, Ensemble methods in machine learning, in: J. Kittler, F. Roli (Eds.), First International Workshop on Multiple Classifier Systems, Cagliari, Italy, Lecture Notes in Computer Science, Vol. 1857, Springer, Berlin, 2000, pp. 1–15.
[19] T.G. Dietterich, E.B. Kong, Machine Learning Bias, Statistical Bias, and Statistical Variance of Decision Tree Algorithms, Technical Report, Department of Computer Science, Oregon State University, 1995.
[20] J.H. Friedman, Local Learning Based On Recursive Covering, Technical Report, Dept. of Statistics, Standford University, August 1996.
[21] P. Geurts, C. Olaru, L. Wehenkel, Improving the bias/variance tradeoff of decision trees: towards soft tree induction, Eng. Intelligent Syst. 9 (4) (2001) 195–204.
[22] P. Geurts, L. Wehenkel, Investigation and reduction of discretization variance in decision tree induction, Proceedings of 11th European Conference on Machine Learning, ECML 2000, Barcelona, Spain, May/June 2000, pp. 162–170.
[23] L.O. Hall, P. Lande, Generating fuzzy rules from decision trees, Proceedings of International Fuzzy Systems Association of World Congress, Vol. 2, Prague, 1997, pp. 418–423.
[24] I. Hayashi, J. Ozawa, L.C. Jain, Generation of Fuzzy Decision Trees by Fuzzy ID3 with Adjusting Mechanism of AND/OR Operators, IEEE 1998.
[25] A.P. Heinz, Learning and generalization in adaptive fuzzy logic networks, in: H.-J. Zimmermann (Ed.), EUFIT'94, Proceedings of the Second European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany, 20–23 September 1994, pp. 1347–1351.

[26] A.P. Heinz, Pipelined neural tree learning by error forward-propagation, ICNN'95, Proceedings of the IEEE International Conference on Neural Networks, Vol. I, Perth, Western Australia, 27 November–1 December, 1995, pp. 394–397.
[27] H. Ichihashi, T. Shirai, K. Nagasaka, T. Miyoshi, Neuro-fuzzy ID3: a method of inducing fuzzy decision trees with linear programming for maximizing entropy and an algebraic method for incremental learning, Fuzzy Sets and Systems 81 (1996) 157–167.
[28] A. Ittner, J. Zeidler, R. Rossius, W. Dilger, M. Schlosser, Feature space partitioning by non-linear and fuzzy decision trees, Proceedings of International Fuzzy Systems Association World Congress, Vol. 2, Prague, 1997, pp. 394–398.
[29] C.Z. Janikow, Fuzzy decision trees: issues and methods, IEEE Trans. Systems Man, Cybernetics—Part B: Cybernetics 28 (1) (1998) 1–14.
[30] B. Jeng, Y.-M. Jeng, T.-P. Liang, FILM: a fuzzy inductive learning method for automated knowledge acquisition, Decision Support Systems 21 (1997) 61–73.
[31] M.I. Jordan, A statistical approach to decision tree modeling, in: M. Warmuth (Ed.), Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory, ACM Press, New York, 1994.
[32] C. Marsala, Apprentissage inductif en présence de données imprécises: construction et utilisation d'arbres de décision flous, Thèse de doctorat, Université Paris 6, 1998.
[33] C. Marsala, B. Bouchon-Meunier, Forests of fuzzy decision trees, Proceedings of the International Fuzzy Systems Association World Congress, Vol. 2, Prague, 1997, pp. 369–374.
[34] C. Olaru, Fuzzy Decision Tree Induction using Square Error Type of Criterion, Internal Report, University of Liege, Department of Electrical and Computer Engineering, Belgium, October 1998.
[35] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C. The Art of Scientific Computing, 2nd Edition, Cambridge University Press, Cambridge, 1994.
[36] J.R. Quinlan, Induction of decision trees, Machine Learn. 1 (1986) 81–106.
[37] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, Inc., San-Mateo, 1993.
[38] M. Ramdani, Système d'induction formelle à base de connaissances imprécises, Thèse de doctorat, Université Paris VI, Paris, France, February 1994.
[39] J.-S. Roger Jang, Structure determination in fuzzy modeling: a fuzzy CART approach, Proceedings of the Third IEEE International Conference on Fuzzy Systems, Vol. 1, June 26–29, Orlando, Florida, 1994, pp. 480–485.
[40] I.K. Sethi, Entropy nets: from decision trees to neural networks, Proc. IEEE 78 (10) (1990) 1605–1613.
[41] I.K. Sethi, Neural implementation of tree classifiers, IEEE Trans. Systems Man Cybernetics 25 (8) (1995) 1243–1249.
[42] G.H. Shah Hamzei, D.J. Mulvaney, On-line learning of fuzzy decision trees for global path planning, Eng. Appl. Artificial Intelligence 12 (1999) 93–109.
[43] T. Shibata, T. Abe, K. Tanie, M. Nose, Motion planning of a redundant manipulator based on criteria of skilled operators using fuzzy-ID3 and GMDH, Proceedings of Sixth IFSA World Congress, Vol. 1, Sao Paulo, Brazil, July 21–28, 1995, pp. 613–616.
[44] A. Suarez, F. Lutsko, Globally optimal fuzzy decision trees for classification and regression, IEEE Trans. Pattern Anal. Machine Intelligence 21 (12) (1999) 1297–1311.
[45] T. Tani, M. Sakoda, K. Tanaka, Fuzzy modeling by ID3 algorithm and its application to prediction of heater outlet temperature, Proceedings of the First IEEE Conference on Fuzzy Systems, San Diego, 1992, pp. 923–930.
[46] L. Torgo, Inductive Learning of Tree-based Regression Models, Ph.D. Thesis, Department of Computer Science, Faculty of Sciences, University of Porto, September 1999.
[47] E.C.C. Tsang, X.Z. Wang, Y.S. Yeung, Improving learning accuracy of fuzzy decision trees by hybrid neural networks, IEEE Trans. Fuzzy Systems 8 (5) (2000) 601–614.
[48] T. Tsuchiya, T. Maeda, Y. Matsubara, M. Nagamachi, A fuzzy rule induction method using genetic algorithm, Internat. J. Industrial Ergonomics 18 (1996) 135–145.
[49] P. Turney, Technical note: bias and quantification of stability, Machine Learn. 20 (1995) 23–33.
[50] M. Umano, H. Okamoto, I. Hatono, H. Tamura, Fuzzy Decision Trees by fuzzy ID3 algorithm and its application to diagnosis systems, Proceedings of The Third IEEE Conference on Fuzzy Systems, Vol. 3, June 26–29, Orlando, FL, 1994, pp. 2113–2118.
[51] P.E. Utgoff, Perceptron trees: a case study in hybrid concept representations, Connect. Sci. 1 (4) (1989) 377–391.

[52] X.Z. Wang, B. Chen, G. Qian, F. Ye, On the optimization of fuzzy decision trees, Fuzzy Sets and Systems 112 (1) (2000) 117–125.
[53] Q.R. Wang, C.Y. Suen, Large tree classifier with heuristic search and global training, IEEE Trans. Pattern Anal. Machine Intelligence PAMI-9 (1) (1987) 91–102.
[54] R. Weber, Fuzzy ID3: a class of methods for automatic knowledge acquisition, Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Networks, Iizuka, Japan, July 17–22, 1992, pp. 265–268.
[55] L. Wehenkel, An information quality based decision tree pruning method, Proceedings of the IPMU'92 Conference, Palma de Mallorca, Spain, July 6–10, 1992.
[56] L. Wehenkel, Discretization of continuous attributes for supervised learning. Variance evaluation and variance reduction, Proceedings of the Seventh IFSA World Congress (invited paper), Vol. 2, Prague, June 25–29, 1997, pp. 381–388.
[57] L. Wehenkel, Automatic Learning Techniques in Power Systems, Kluwer Academic, Boston, 1998.
[58] L. Wehenkel, M. Pavella, Decision tree approach to power system security assessment, Electrical Power Energy Systems 15 (1) (1993) 13–36.
[59] Y. Yuan, M.J. Shaw, Induction of fuzzy decision trees, Fuzzy Sets and Systems 69 (1995) 125–139.
[60] J. Zeidler, M. Schlosser, Continuous-valued attributes in fuzzy decision trees, Proceedings of Information Processing and Management of Uncertainty in Knowledge-Based Systems, Granada, 1996, pp. 395–400.