UNIVERSITÉ DE LIÈGE
Faculté des Sciences Appliquées
Département d'Électricité, Électronique et Informatique
Institut Montefiore

Contributions to Automatic Learning:

Soft Decision Tree Induction

Thèse présentée par
Cristina Olaru
en vue de l'obtention du titre
de Docteur en Sciences Appliquées

Année Académique 2002–2003

Abstract

This thesis proposes, develops and validates a new fuzzy decision tree induction method called soft decision tree. It is a data mining method of automatic knowledge extraction from large databases in classification and regression tasks. The method is an extension of the classical crisp tree induction: fuzzy logic is embedded into the induction process with the effect of more accurate models, but still interpretable and autonomous.

The explored method combines tree growing and pruning, to determine the structure of the soft decision tree, with refitting and backfitting global optimizations, to improve its generalization capabilities. The method is explained and motivated and its behavior is analyzed empirically on 18 databases in classification. The results obtained for the soft decision method are compared with results of standard crisp decision and regression trees and with results of several aggregation methods (methods to reduce variance and thus to improve accuracy).

Complementary studies of the bias-variance tradeoff in the model accuracy are performed that justify the improved accuracy of fuzzy decision trees with respect to crisp decision and regression trees essentially by a lower prediction variance. Thus, fuzzy decision trees are indeed another tool for reducing variance, beside pruning and aggregation techniques. However, the study shows that fuzzy decision trees do not reduce variance as much as aggregation methods do. When a fuzzy decision tree presents better accuracy than one or more of the aggregation methods, this is due to a lower prediction bias.

Finally, a parameter variance and bias study shows that, the reduction in prediction variance of a fuzzy decision tree with respect to a crisp regression tree is not correlated with a reduction in parameter variance.

Acknowledgments

I am very glad to have, by means of these few words, the opportunity to share some of my thoughts with several persons I do respect a lot.

First of all, lots of thanks to Professor Louis Wehenkel. I always appreciated his professionalism, his abundance of ideas, his intuition in research, his open-minded and forthright spirit, his flexibility to manage a team where everyone feels good and motivated to do research. I thank him for introducing me with enthusiasm in his field of research, for his confidence in me, his patience, advice and valuable comments, his contribution and support from the beginning. This thesis would not have seen the light without him, a fortiori given that all started from his idea. Thank you professor.

A lot of soulful thanks to Professor Mania Pavella which welcomed me with confidence and her characteristic warmth in her team six years ago, when in April 1997 I walked in Liège for the first time, through her administrative efforts. A vivace and always motivated spirit, an open-minded person and of faultless professionalism, enthusiasm and faculty to motivate. She was always an example for me. I thank her a lot.

Special thanks to Pierre Geurts for the interesting discussions on the subject, his valuable help and fruitful comments. Thanks to Philippe Mack for his always-enthusiastic spirit as colleague. Thanks to Damien Ernst for being an unfailing friend and a funny colleague. Thanks to Raphaël Marée for his availability and help when some unexpected computer difficulty was retaining me from my work. Thanks to my colleagues and to the entire team for professional and logistic support, for the convivial atmosphere and recreational moments. A good thought for the following persons: Marie-Berthe Lecomte, Claude Diépart, Florin Capitanescu, Costica Bulac, Ibtissam El Khayat, Christophe Druet, Cédric Moors, Daniel Ruiz.

Thanks to all professors from the thesis jury for accepting to take part, for their willingness of reading this thesis and for their interest in my work.

Thanks to my parents and sister for their unquestioned love and support. Special thanks to Marius who although far lately, he has known to be however so close by me. I dedicate with affection this thesis to him.

Thanks to all who contributed to the defence of this thesis maybe even without knowing it, to my professors (in particular Professor Sorin Patrascoiu), to my best friends for their constant friendship, encouragement and stimulation.

From all I have learned something. This means a lot to me. A nice thought for every one.

As you may see, all is not but a question of idea, confidence, support, love, friendship, motivation, enthusiasm and special people. And some work, it's true!

Cristina Olaru
30 June, 2003

Contents

List of Figures

List of Tables

Frequently used notations

All the notations and abbreviations are fully defined at the place they are introduced. We have collected below some of the most frequently used ones.

Automatic Learning

o	an object
U	the universe of all possible objects for a problem
$\mathbf{a}$	the vector of attributes (inputs)
y	the output variable
$\hat{y}$	prediction of the output y in regression
$\hat{c}$	estimated class rule of the output c in classification
$E_{\mathbf{a}}\{f(\mathbf{a})\}$	the expectation of the function $f(\mathbf{a})$ with respect to the distribution of the random variable $\mathbf{a}$
$E_{y\|\mathbf{a}}\{f(y)\}$	the expectation of the function $f(y)$ with respect to the distribution of the random variable y given $\mathbf{a}$
E	error function to be minimized (mean squared error or squared error)
$f(\mathbf{a})$	the Bayes model
$\hat{f}(\mathbf{a})$	the estimated output
$\bar{\hat{f}}(\mathbf{a})$	the average prediction of the output
q	number of randomly drawn learning sets of the same size from the pool set in order to estimate bias and variance of a learning method
Pe	error rate of a learning algorithm (%)
MSE	mean squared error
MAE	mean absolute error

Fuzzy Theory

μ_A	membership function of the fuzzy set A
$\mu_{\bar{A}}$	membership function of the complement of the fuzzy set A
$\|A\|$	cardinality of fuzzy set A
$\top$	t-norm
$\perp$	t-conorm
$\wedge$	conjunction operator
$\vee$	disjunction operator

Fuzzy Decision Tree

LS, PS, TS	learning, pruning, and testing set, respectively
RS, BS	refitting and backfitting set, respectively
$\|\|GS\|\|$	number of objects in set GS
$o \in S$	object in node S
$\mu_C(o)$	real membership degree to the class C for object o
$\hat{\mu}_C(o)$	tree estimation of the membership degree to the class C for object o
$\mu_{S_j}(o)$	membership degree at node j for object o
$\mu_{S_{L_j}}(o)$	membership degree at leaf j for object o
$\hat{\mu'}_C(o)$	estimation of the membership degree to the target class in a test node of the tree, for object o
$\hat{\mu}_{C_{S_i}}(o)$	the membership degree to the class C for object o approximated by the subtree rooted at node S_i
L_j	label of leaf j
$\mu_S(o)$	membership degree of object o to fuzzy node S
$\mu_{S_L}(o)$	membership degree of object o to left successor of node S
$\mu_{S_R}(o)$	membership degree of object o to right successor of node S
L_L, L_R	labels of left and right successors of node S
$\sum_{o \in S} \mu_S(o)$	cardinality of the fuzzy set of node S
$\nu(\cdot)$	discriminator of a test node
a	attribute of a test node
α	cutting point parameter of the discriminator at a test node
β	width parameter of the discriminator at a test node
$\alpha_{asymptotic}$	reference value for the cutting point in the root node of a tree
E_S	local squared error function at node S
K	tree complexity (the number of its test nodes)
q	free current set of parameters
δq	set of parameters increment in backfitting
$E(q)$	error function - squared error computed with the set q of parameters
ΔE	gain in terms of squared error from a set of parameters (q) to another ($q + \delta q$)
ΔE_{min}	threshold on gain ΔE used as stopping limit for the iterative process
cy_{max}	maximum number of cycles allowed
cy	current number of cycles
λ	LEVENBERG-MARQUARDT algorithm factor
λ_{init}	initial selected value for factor λ
λ_{step}	multiplying factor at one step for factor λ

Chapter 1

Introduction

The research presented in this thesis turns around the development of a new and complete soft (fuzzy) decision tree technique[1] *and the study of its behavior when applied to different data sets. This chapter introduces the reader in the context, by stating the motivations behind such a project, the author's objectives together with the main contributions of this thesis to the Automatic Learning field.*

1.1 Motivations and context

1.1.1 Data Mining

In almost every real-life field we are confronted with huge amounts of data preserved in computer data collections and coming from measurements, software and hardware simulations or simply, from manual data registration and centralization procedures. Taking advantage from all these data means to analyze and understand them. Data Mining or Knowledge Discovery in Databases [FPSSU96] are the labels of the data analysis framework applied to real problems.

Data Mining (DM) is a folkloric denomination of a complex activity which aims at extracting synthesized and previously unknown information from large databases. It denotes also a multidisciplinary field of research and development of algorithms and software environments to support this activity in the context of real life problems. DM is sometimes considered as just a step in the broader overall process called Knowledge Discovery in Databases (KDD), but is often used as a synonym of the latter. Thus, DM software includes tools of automatic learning from data, such as machine learning and artificial neural networks, plus the traditional approaches to data analysis such as query-and-reporting, on-line analytical processing or relational calculus, so as to deliver the maximum benefit from data.

The general purpose of data mining is to process the information from the enormous stock of data we have or that we may generate, so as to develop better ways to handle data and support future decision making, be it in the form of a classification or a prediction problem, searching for causalities or modeling dependencies between data, extracting trends from temporal data or identifying abnormal deviations in data. Databases are usually expensive to create and maintain, and for a small additional investment in mining them, highly profitable information may be discovered hidden in the data.

The interest in the data mining field and its exploitation in different domains (marketing, finances, banking, engineering, health care, power systems, meteorology, ...) has increased recently due to a combination of factors. They include: the emergence of very large amounts of data (terabytes - 10^{12} bytes - of data), the dramatic cost decrease of mass storage devices, the emergence

[1] We will use interchangeably throughout this thesis the denominations soft decision tree and fuzzy decision tree when referring to our method.

and quick growth of data base management systems, the advances in computer technology such as faster computers and parallel architectures, the continuous developments in automatic learning techniques and the possible presence of uncertainty in data (noise, outliers, missing information).

1.1.2 Automatic Learning

As an engine of the data mining framework, Automatic Learning (AL) [Weh98] denotes a broad field of research concerned with the development of the algorithms and methods used in data analysis. Automatic learning techniques identify models of a system, phenomenon, specific problem, real or simulated (i.e. modeled by analytical equations) that may be used to predict information about future situations, by exploiting recorded data about already "seen" situations. The techniques are called automatic due to their capacity of doing the job of learning from past experience without much help or dependence on a human being, thus computer based. Due to their usefulness and practical impact on real life problems, the most well known and the most widely used AL techniques are: machine learning techniques, neural networks, and statistical techniques. These are methods open to improvements but enough mature in the same time.

As a short review of the possible contributions of AL to data analysis, here are some hints. Sometimes, the patterns to be searched for, and the models to be extracted from data are subtle, and require complex calculus and/or significant specific domain knowledge. Or even worse, there are situations where one would like to search for patterns that humans are not well suited to find, even if they are good experts in the field. For example, in many power system related problems one is faced with high dimensional data sets that can not be easily modeled and controlled on the whole, and therefore automatic methods capable of synthesizing structures from such data become a necessity. Another example would be the AL possibility of systematically screening all the possible situations of a high order dimensionality problem in order to decide on the best alternative (e.g. protection design in power systems, maintenance planning, identifying the faults likely to create instability among all possible faults, etc). Further on, some parts of a complex system are intrinsically difficult to be accurately modeled analytically, either because they need to be represented in a reduced way, or because they change with time, or they vary significantly from one geographical region to another, thus because they are depending also on other factors than the analytically modeled ones. AL techniques are useful tools in order to identify new valuable dependencies in data not known already by the domain experts, therefore helping in a better understanding of the phenomena.

AL is used both in a preventive (anticipatory) way, so as to estimate the best decisions to take in the future and in a emergency way, so as to determine online in a short time span usually, the best move to make which takes in consideration drastic (some times) changes of the initial modeled conditions and hypothesis. In both cases, it gives a better insight into what *really* happens in the studied phenomena.

Many times AL techniques provide redundant results with the ones obtained by other means, but this is only the best way to verify the tools and to convince the domain experts which have had, mostly in the past, some inertial non-confidence in these tools.

Further efforts are done in AL field on directions like algorithms scalability for multidimensional AL [GGR99] (the property of having a runtime linear in the database records number), algorithms interpretability, interactivity for the on-line processing systems (with the *anytime* algorithms, i.e. algorithms that can produce a meaningful approximate result at any time during the execution [HAC+99]), time-dependencies exploration [Geu02], algorithms succinctness (by means of pruning techniques), incremental algorithms (algorithms that do not begin learning from scratch every time database grows).

This thesis attempts to contribute to this field of AL by proposing a new approach emerged from a well known method, namely decision tree induction.

1.1.3 Decision trees

The most known representative of the Machine Learning (ML) paradigm, decision tree (DT) induction [Qui86, Qui90, Qui93, BFOS84, SL91, ZR00] is basically concerned with the automatic design of **if-then** rules similar to those used by humans, in a hierarchical structure. It is used in classification problems (decision tree), regression problems (regression tree) or time-dependent prediction (temporal decision tree). Their main strength is the *interpretability* character, because the form in which the results are provided (a graphical structure or a rule base) facilitates the comprehension and matches to the human reasoning. For every new situation of the studied phenomenon, the *transparency* of a tree conclusion may be easily validated and thus accepted. Another asset is the ability to automatically identify the most representative data inputs for a given task. Therefore, from maybe thousands of possible inputs it has the "power" to choose only a few ones, the most discriminating for the studied case. And this because as a model it makes little assumption on the hypothesis space, being *flexible and autonomous*. Practically, it is a model that chooses by itself its own inputs, particularly useful approach for large input spaces problems. Finally, it is a *computationally efficient* tool, being able to handle very large scale problems, thus particularly useful in data mining. As a counter part, it is generally less accurate than other AL models (e.g. neural networks).

1.1.4 Bias-variance tradeoff

The accuracy of a AL model built from a training data, comes from three sources of errors [Fri96, Weh94]:

- a *residual error*. It is the lower bound on the expected error for a given problem, i.e. the lowest error one can get by training a model of unrestricted complexity, with an infinite amount of data concerning the given problem;
- a *bias* error. It reflects the systematic error that the AL model (produced by a given algorithm) is expected to have on average when trained on data sets of the same (finite) size. Generally, the simpler the model, the higher the bias.
- a *variance* error. It measures the sensitivity of the AL model to changes in the training data and captures random variation in the model from one training set to another of the same size. A too complex model is likely to have a high variance and the model may exhibit what we call an *overfitting* phenomenon, an over adjustment of the model parameters to the training data.

The first term is linked to the analyzed data, it is the expected error of the best possible model (also 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. There should be a *tradeoff* between these two types of errors so as to obtain an improved AL model accuracy.

Decision trees are recognized as highly unstable classifiers with respect to minor perturbations in the training data, in other words, methods presenting high variance. Reducing complexity will decrease variance but increase bias. Reducing variance translates in improved accuracy, provided that it is not obtained at the expense of a much higher bias.

Some literature is dedicated to the handling of this tradeoff between bias and variance no matter the AL model, and in principle, to the reduction of the variance in decision trees. Geurts systematically investigates this tradeoff in the case of different AL models with a special interest in decision trees [GOW01, GW00, Geu01, Geu02].

There are mainly three ways to reduce the variance in decision trees and handle a better trade-off. Firstly, there are different ways to reduce the model complexity or to control it. Pruning is such a way [EMS97, Min89a, Weh92a]. Its serious drawback is that it does not succeed always to improve the accuracy, because together with the variance reduction appears also a bias increase. Secondly, recently, aggregation methods [Die00a] are well established as a way to obtain highly accurate classifiers by combining less accurate ones, but unfortunately loosing also in interpretability. The most well known aggregation techniques (called also ensemble methods) are bagging [Bre96] and boosting [FS96]. Thirdly, there are methods using fuzzy sets inside the tree induction, called **fuzzy decision trees** (FDT). The present thesis has the ambition to present a new such model and in particular, to demonstrate that the model improved accuracy in classification comes from a reduction in variance.

1.1.5 Fuzzy reasoning

What is fuzzy theory? It is a way to consider in the mathematical representation of logic, the vagueness present in every day life. Fuzzy sets [DP80] are a generalization of the conventional set theory and have been introduced by Zadeh in 1965 [Zad65]. They are a concept that can bring the reasoning used by a computer (e.g. an AL model) closer to that used by people. Whereas a conventional, or "crisp" set has sharp boundaries (0 **or** 1), the transition between full membership (1) and non membership (0) is rather *gradual* in a fuzzy set. So that a model can ask questions such as "Is the temperature too high?" instead of clear-cut yes-or-no-answers questions like "Does the temperature exceed 40 degrees?".

Why do we need fuzzy reasoning in practice? Because the imprecision has some advantages:

- The elasticity of fuzzy sets simplifies the task of translating between human reasoning, which is inherently elastic, and the rigid operation of digital computers based on conventional mathematical reasoning [2]. For a human, certain instructions could appear as too precise and *the precision may be quite useless*, while vague directions can be interpreted and acted upon. Fuzziness also allows computers to use the type of human knowledge called common sense (knowledge in the form of statements that are usually, but not always, true) [Zad84].

- From some real-world applications, precise data are difficult or expensive, if not impossible, to obtain. Usually, the information observed from a system is available either in the form of numerical (precise) information from measurements or computations and/or in the form of linguistic (imprecise) information from human experts. The theory of fuzzy logic is designed to *exploit both kinds of information*.

- Fuzzy theory comes with a natural way of expressing knowledge and *the resulting models are more acceptable* because they represent information similarly to the way human beings understand problems and because an answer in the form of a fuzzy class provides richer information comparatively with the yes-no type of answer.

1.1.6 Fuzzy decision trees

A fuzzy decision tree is a combination between fuzzy reasoning and the decision tree induction. It is a model concerned with the (automatic) design of **fuzzy if-then** rules in a tree structure. It is

[2] As a motivating anecdote, in the 1970s, Ebrahim Mamdani, a control engineer at Queen Mary College in London, was attempting to develop an adaptive system that could learn to control an industrial machine. Unfortunately, the computer learned the human operator's mistakes as well as the operator's successes (an overfitting problem). Afterwards, the use of fuzzy rules of thumb directly in the automating process controls was a pioneering work that led a decade later to a growing literature in fuzzy control.

used in classification problems in most of the cases but sometimes also in regression problems.

To make a distinction, the introduction of the fuzzy environment in decision tree technique is based within the fuzzy decision tree community on three reasons. Firstly, one wanted to enlarge the use of decision trees towards the ability to *manage fuzzy information* in the form of fuzzy inputs, fuzzy classes, and fuzzy rules. There are indeed cases where the lack of data in a field is supplied by linguistic, imprecise information coming from experts, in the form of fuzzy information (e.g. system *almost* unstable, level of oil *considerably* high) and one needs to be able to process this type of (precious) data with interpretable results. Secondly, one realized that the use of fuzzy logic on many learning techniques could *improve their predictive accuracy* even in the context of precise data and fuzzy sets are tools to enforce this ability with respect to a classic decision tree, while preserving the interpretability character. For this purpose, available crisp data, usually in numerical continuous form, are especially fuzzified before, during or after the growing stage of a decision tree and different tree heuristics are proposed so as to cope with the fuzzified data. Finally, the appealing characters of *interpretability and autonomy* in decision tree models are kept intact and possibly improved in a fuzzy tree due to a possible reduction in the model complexity.

Building a fuzzy decision tree model is in most of the literature solely a question of growing. We speak about a top-down induction either as a crisp tree that afterwards is fuzzified, or as a tree that directly integrates fuzzy reasoning during the growing phase. The numerical inputs must be fuzzified (transformed from continuous values to fuzzy values) and this fuzzification step is rarely automatic. There are approaches that consider also a pruning stage after growing in order to tradeoff better the model to data. A few authors have implemented after growing, also a global optimization phase of some parameters of the model. A global optimization aims at tuning the parameters of the model not anymore based on local learning samples but on the whole learning set and the already identified structure of the tree. Chapter 5 gives more insight in the fuzzy decision tree current state of advancement.

1.1.7 Automatic Learning techniques needs

In a data mining process, the quality of the result (mined information) is a function of the quality of the data base (often size) and moreover, of the effectiveness of the AL techniques. Here are simultaneous needs for an AL technique to be considered effective:

- to be *accurate*, with very good generalization capabilities on new data;
- to be *interpretable* and comprehensible in order to gain the people confidence in the model output;
- to 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 as less human intervention as possible;
- to be *efficient* when training and using it, thus be able to manage large amounts of data and to model complex real-life large-scale problems.

Unfortunately, studies on different learning algorithms applied to multiple sets of simulated or real-life data [LLS00, HTF01] and evaluated in terms of model error, model complexity, training time or robustness (influence of noise) have shown empirically that the techniques of our days still can not present all these simultaneous needs. Placing the constraint of interpretability on an approximator usually degrades its predictive accuracy [Fri96], and the price paid for globally improved accuracy usually appears as increased training time and/or loss of interpretability (aggregation methods case).

1.2 Objectives

In the spirit of coming with a contribution to our-days-needs of learning methods, the main objective of this thesis is to propose a new method of fuzzy decision trees with the aim of satisfying the above needs. In other words, our main objective is to improve the accuracy of the models with respect to crisp decision trees and to bring the new proposed models near the most accurate techniques, while trying to retain the crisp trees main features concerning interpretability, autonomy and processing time.

We start our work by exploring a computationally efficient method of growing fuzzy decision trees based on a CART [BFOS84] regression tree type of induction, and a least squares error criterion adapted to fuzzy reasoning. This methodology aims at avoiding inputs fuzzification a priori or a posteriori to tree growing or any other human intervention throughout the process. We then address the important question of avoiding overfitting to the training data with the objective of providing a better compromise between accuracy and size of the models, through the phase of pruning. Further on, we propose two ways of accuracy improvements through global optimizations and we study advantages and disadvantages for each. A particular concern was in this work the computational efficiency throughout all these algorithms. Algorithm complexities have been reported step by step. Incremental formulas have been used as much as possible in order to speed up the algorithms. Finally we have extensively compared our proposals with existing methods. Firstly, we studied the behavior of the proposed FDT method on 7 large databases of several thousand objects since in data mining analyses we may be confronted with large amounts of data. We compared the proposed FDT with: (i) interpretable methods like crisp decision trees (ULG [Weh98] and C4.5 [Qui93]) and regression trees (CART [BFOS84]) and (ii) more accurate methods like aggregation methods (bagging, boosting, randomized trees, dual perturb and combine decision trees), results published in details in [Geu02]. Secondly, we applied our FDT method on 11 well known datasets and its performance is evaluated using multiple 10-fold cross-validations and statistical significance tests. Here the goal is to compare FDT with standard methods on standard datasets, using a standard protocol. All the data used in experiments excepting 2 datasets come from the UCI Machine Learning Repository.

Our second objective of the thesis is to prove empirically that a reduction in the global variance of the fuzzy decision tree model is the cause of its improved generalization capabilities. An adjacent objective is to study the variance and bias of the model's parameters, as a reflection of the stability of the parameters and thus of the interpretability of the method.

Due to the output formalism, the method works both in regression and classification problems, and it is able to process fuzzy outputs. Due to the induction procedure formalism, the method handles exclusively two-class problems, and forests of trees may solve multiclass tasks.

An important statement: all the empirical studies presented in this dissertation are concerning classification problems.

1.3 Main contributions

In the context of the background presented in chapter 5, concerning existing fuzzy decision tree methods, next we highlight the main contributions of this thesis:

- A new decomposed discretization procedure for growing a fuzzy decision tree discussed in section 3.4. The aim is to improve the tool autonomy compared with other fuzzy decision tree approaches.

- A complete backward pruning procedure in section 3.5 adapted to the formalism of the proposed growing induction. The aim is to control the model complexity.

- An algorithm to globally tune some of the model parameters, based on linear least squares regression in subsection 3.6.1. In our knowledge, no other publication implemented such a simple but useful trick. We inspired us from [BW99], there being presented as a sub-step of their global optimization process. The aim is to improve the model accuracy.

- An algorithm to globally tune all the model parameters, based on a LEVENBERG-MARQUARDT nonlinear optimization that integrate backpropagation techniques in subsection 3.6.2. This step comes from an idea presented but not implemented in [BW99]. Again the aim is to improve further the model accuracy.

- A global bias-variance quantitative analysis of the proposed fuzzy decision tree model and comparison with crisp trees and aggregation methods in this aspect, in section 4.2.4. We have no knowledge of previous literature that estimates quantitatively variance on fuzzy decision trees. This is the first research work which proves that a fuzzy decision tree method may reduce prediction variance, this being reflected in the model accuracy improvement.

- A parameter bias and variance quantitative study on our FDT in section 4.2.5 with the aim of analyzing parameters stability. The same type of analysis has been done for decision trees by [Weh97] and more in depth by [Geu02], but never for fuzzy decision trees.

In addition, section 6.4 of the last chapter proposes a new alternative to FDT growing, that we called residual fitting induction, because during tree growing, it takes always in consideration the parts of the tree already developed. This is not the case of any other fuzzy decision tree method proposed so far and may be an interesting approach to implement and test empirically in a future work.

The fuzzy decision tree model has been implemented by the author[3] as a complementary module to a software of Data Mining called $PEPITO^{®}$ [4], developed so far at University of Liège [5] for research, teaching and applications of automatic learning [Weh98], presently, a commercial software [6]. The FDT code is written in Common Lisp, which is not the fastest language in terms of computational efficiency. That is why, CPU times are indicated throughout the empirical studies mostly in the intention to compare methods. The fuzzy decision tree software has also a module on fuzzy decision tree visualization. ULG decision trees and CART regression tree modules are part of this data mining software. Aggregation methods modules are complementary modules of this software, implemented by Geurts [Geu02]. The software for C4.5 decision trees (Release 8) is taken from *http* address www.cse.unsw.edu.au/∼quinlan/.

1.4 Outline of the thesis

The dissertation is organized as follows.

Chapter 2 introduces elements of Automatic Learning and Fuzzy Logic theories necessary for a good understanding of the proposed fuzzy decision tree method. The general framework of supervised Automatic Learning is defined. Three supervised learning families of methods are presented in short: linear regression, multilayer neural network, crisp decision and regression trees. All three methods are useful in a way of another to the growing and optimization of our method of fuzzy decision trees. The bias and variance tradeoff is then analysed together with the overfitting phenomenon. Two principal methods to reduce variance in tree-based methods are described: pruning and aggregation methods. Among these later, we explain several methods to

[3] email Ana.Olaru@ulg.ac.be
[4] formerly named ATDIDT
[5] www.montefiore.ulg.ac.be/services/stochastic/
[6] www.pepite.be

which our fuzzy decision tree algorithm will be compared later on in chapter 4. A section on how we evaluate learning algorithms and in particular their accuracy is provided. In the second part of the chapter, basics on fuzzy theory, fuzzy sets and fuzzy systems are equally addressed.

Chapter 3 describes in detail our proposed soft decision tree induction method. The method combines growing and pruning, to determine the structure of the soft decision tree, with refitting and/or backfitting, two optimization strategies able to improve the soft tree generalization capabilities. All these four steps and their variants are explained throughout this chapter.

Chapter 4 groups the experimental part of the thesis into two parts, corresponding to two complementary protocols of results. Soft decision trees are compared with other automatic learning methods. Various behaviors are analysed. Finally, some words are dedicated to the visualization of a soft decision tree.

Chapter 5 gives a short review of the major related literature to fuzzy decision trees.

Finally, chapter 6 highlights the conclusions of all the work, discusses three reasons for which fuzzy decision trees have a better accuracy than crisp decision tree methods and gives some directions for further work.

Appendix A provides a CART-FDT parallel summarized in formulas. Appendix B describes all the datasets used in our experiments. Appendix C provides the mathematical details behind the residual fitting approach. Appendix D collects detailed (numerical) results which have been synthesized in the body of the chapter 4.

1.5 My publications related to this thesis

The core of the research presented in this thesis, including method and results, has been published in

C. Olaru and L. Wehenkel. A complete fuzzy decision tree technique, Fuzzy Sets and Systems *138 (2003) 221-254.*

A parallel between neurofuzzy and fuzzy decision tree approaches has been done in

C. Olaru and L. Wehenkel. On neurofuzzy and fuzzy decision tree approaches. In B. Bouchon-Meunier, R. R. Yager and L. A. Zadeh, editors, Information, Uncertainty, Fusion*, pages 131-145. Kluwer Academic, 2000.*

Studies on the criterion based on squared error function and preliminary bias and variance studies have been done in

C. Olaru. Fuzzy decision tree induction using square error type of criterion. Internal Report, University of Liège, Department of Electrical Engineering, Belgium, October 1998.

Chapter 2

Notions of Automatic Learning and Fuzzy Logic

This chapter is a general introduction to Automatic Learning and Fuzzy Logic. It presents all the elements of Automatic Learning and Fuzzy Logic theories necessary for a good understanding of the proposed fuzzy decision tree method. Similarly, notions or algorithms that are considered in the rest of the thesis are equally explained here (for example, algorithms to which our fuzzy decision tree algorithm will be compared later on). Readers already accommodated with these notions may skip this chapter and visit it when necessary.

2.1 Automatic Learning

Automatic Learning denotes a broad field of research concerned with the development of methods used in the automatic extraction of information in various forms from data bases. Automatic learning methods tool-box is one of the pillars of the data mining field, which being a multidisciplinary field, is concerned not only with the research and development of algorithms (the automatic learning framework), but also with the software environment (database management system, visualization tools framework) that supports this activity in the context of real life problems.

Learning means behaving better as a result of experience. Automatic learning aims at formulating this experience through models or patterns automatically found in data, or as relationships between data which give a better understanding of the phenomenon as correlations, similarities, cause-and-effect dependencies and deviations from the normal behavior. In the machine learning community the terms model and pattern are used interchangeably, or there is a distinction between them: a *model* is a global description of a data set making statements about any point in the full measurement space, while a *pattern* is a local feature of the data, making statements only about restricted regions of the variables space [HMS01]. We adopt the first terminology and in what follows, model refers also to pattern.

A model is a formulation of a relation between some variables of interest in data called *outputs* and some observable variables called *inputs*. A model predicts outputs function of inputs.

The work presented in this thesis is a form of *inductive inference* learning. Induction refers to the inference from the specific case to the general, deriving conclusions from given facts, conclusions that may be afterwards used to infer unknown information for new situations. The automatic learning methods presented in this chapter belong all to this type of applications.

There are three ways to perform learning:

- In *supervised learning* ("learning with a teacher"), for each input, the learning system receives the desired output. This situation occurs when the user has already identified its objective (i.e. output) and this output is observable (available in data).

- In *reinforcement learning* ("learning with a critic"), the learning system takes an action and gets a response, but it is not as direct, immediate and informative as in supervised learning. It is only an evaluation feedback of its action from the environment, not the correct action.

- In *unsupervised learning* ("learning by observation"), the correct response is not provided at all and data records with similarities are grouped in clusters. This situation occurs when the user does not have a hint at all about the correct output.

We restrict our interest in this thesis to the supervised learning approaches.

2.1.1 Representation of objects by attributes

Let U denote the *universe* of all possible cases for a given problem. They are called *objects* and we may sometimes refer to them as records or examples. A given database is a subset of objects of the universe U. Each object is described by a certain number of *attributes* a (called also variables or features), which are measurable properties of the object. In practice, these attributes provide detailed information about each object but not necessarily a full description fact which prevents the problems from being deterministic. In a database, a column is usually associated with an attribute and a row with an object.

We note typically by $a(\cdot)$ the function of attribute a defined on the universe and with $a(o)$ the value of the attribute for a given object o.

In the sequel, we consider two types of attribute domains:

- *numerical* attributes that take values in a subset of the real axis: $a(\cdot) : U \to \mathbb{R}$

- *symbolic* attributes that assume values in a finite set $a(\cdot) : U \to \{a_1, a_2, \ldots, a_m\}$.

There exists also the notion of *ordered* variables where there is an ordering between the values. These may be symbolic (e.g. small, medium, big) or numerical (e.g. 1, 2, 3). The symbolic unordered attributes are referred to as qualitative, categorical or discrete variables (e.g. red, green, blue). They represent information which is not structured and the attribute values cannot be compared among themselves, nor ordered.

Symbolic attributes are typically represented numerically by codes. In the easiest case of two categories attributes such as "on" or "off", "secure" or "insecure" the coding may be done by binary variables like 0 and 1 or -1 and 1. Coding via *dummy variables* is possible for symbolic attributes with $m > 2$ categories, when the variable is represented by a vector of m 0/1 bits, only one being "on"(1) at a time.

Other attribute types may be *temporal* ones, represented by numerical time series or sequences of events, or *structured* attributes where the domain is defined by a hierarchical taxonomy, but the methods of this thesis do not refer to these types.

2.1.2 Supervised learning problem

We consider a supervised learning problem from examples, that may be generically formulated as [Weh98]:

> Given a set of *examples* (the learning set) 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 unseen 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 the general setting, both input and output may be vectors of values, but we reduce the discussion to single output learning problems.

A *learning set* (or a learning sample) noted LS is a collection of N objects o_i randomly and independently drawn from universe U, described by a vector

$$(a_1(o_i), a_2(o_i), \ldots, a_m(o_i), y(o_i)),$$

where $a_k, k = 1, \ldots, m$ are the input attributes, y the output variable and N the learning set size, with $i = 1, \ldots, N$. The learning set is used to build the model (i.e. the general rule). A *test set* (or a test sample) noted TS is another collection of M objects o_i randomly and independently drawn from universe $U, i = 1, \ldots, M$. The test set is used to test the predictive performance of the model.

A model is characterized by a *structure* and some *parameters*. In general, once the structural form is established for a model, the next step is to estimate its parameters from the available data. Once the parameters have assigned values we refer to a particular model, as a fitted model, trained model, optimized model, or just model for short. The structures represent the general functional forms of the models with unspecified parameter values. A fitted model has specific values for its parameters. The number of parameters to be fitted (optimized) gives the *complexity* of a model.

A supervised learning problem is a search process in a space of candidate models called *hypothesis space*. This space is defined by the possible structures and/or the possible parameter values of the candidate models, e.g. the space of all linear regression models given the inputs, the set of all neural networks of fixed structure, or the set of all decision trees. The hypothesis space defines implicitly the boundaries of what we can learn with the future model. Once chosen a hypothesis space, the search process typically identifies or builds the model providing the maximum *score function* or quality or the minimum *error function* (called also *loss function* in the statistical decision theory), estimated on the observed data i.e on the learning set. The choice of score function may reflect a preference bias towards models that ensure better predictive performance on future cases (the generalization performance) or towards simpler models. This preference bias imposes implicitly an ordering among the hypothesis. The *search strategy* or optimization method used to search over candidate hypotheses is the third component of the learning algorithm.

A *parametric technique* makes assumptions on the hypothesis space by forcing the model to have some *fixed* structure and set of parameters. It is the case of linear models. The hypothesis space consists of a single fixed structure and the search is conducted solely in the parameter space to optimize the score function. A *non-parametric technique* does not fix a priori the structure of the model and its shape is adapted (fitted) to the learning data. It is the case of tree-based models (decision and regression trees). The hypothesis space consists of a family of structures and there is a search over both structure and parameter spaces.

We distinguish between two types of supervised learning problems:

- **Regression** problem when the output y is a numerical, continuous or not, real-valued variable (or a vector of such values). The goal usually is to model smooth input/output relationships. If we note the vector of inputs $\mathbf{a} = (a_1, a_2, \ldots, a_m)$, we search for a prediction model

$$\hat{y} = f(\mathbf{a}), \qquad y : U \to y(U) \tag{2.1}$$

  minimizing an error function like for example the *mean squared error* (called also *squared error loss*) written in terms of probabilistic expectation[1] as

$$E_{o \in U}\{(\hat{y}(\mathbf{a}(o)) - y(o))^2\}, \tag{2.2}$$

[1]This expectation is defined over a probability measure defined on U; in our notations we will assume that U is finite and denote by $P(o)$ the probability mass of any object $o \in U$; we assume that LS and TS are sampled from U using this probability distribution; $E_{o \in U}\{f(o)\}$ stands for $\sum_{o \in U} P(o) f(o)$.

or the *mean absolute error* written in terms of probabilistic expectation as

$$E_{o\in U}\{|\hat{y}(\mathbf{a}(o)) - y(o)|\}. \tag{2.3}$$

Since we have at disposal only objects $o \in LS$, not all $o \in U$, what we search for in practice is the model that minimizes the *empirical mean squared error*

$$E_{regr} = \frac{1}{N}\sum_{i=1}^{N}(f(\mathbf{a}(o_i)) - y(o_i))^2, \tag{2.4}$$

or the *empirical mean absolute error*

$$E_{regr} = \frac{1}{N}\sum_{i=1}^{N}|f(\mathbf{a}(o_i)) - y(o_i)|, \tag{2.5}$$

both computed on LS.

- **Classification** problem when the output c is a symbolic variable, with a finite and usually rather small number of possible mutually exclusive values called *classes*. The goal is to partition the universe into a finite number of regions of the same class. We search for a classification rule

$$\hat{c} = f(\mathbf{a}), \qquad\qquad c : U \to \{C_1, C_2, \ldots, C_n\} \tag{2.6}$$

minimizing an error function like for example the *misclassification rate* (called also *error rate*, or *zero-one loss*)

$$E_{o\in U}\{I(\hat{c}(\mathbf{a}(o)), c(o))\}, \tag{2.7}$$

where $I(a, b)$ is 1 if $a = b, 0$ otherwise. Thus, we search for the model that minimizes the *empirical error rate* [2]

$$E_{0/1} = \frac{1}{N}\sum_{i=1}^{N} I(f(\mathbf{a}(o_i)), c(o_i)). \tag{2.8}$$

Both these tasks are considered in the language of the statisticians, *function approximation* problems as they realize a mapping from a m-dimensional input variable $\mathbf{a}$ to the output.

2.1.3 Some supervised learning methods

We present three very well known supervised learning algorithms indirectly correlated with the work on the new method of fuzzy decision trees. It is a short qualitative presentation. We do not have the pretension to be complete, but just to emphasize some elementary notions so as to accustom the reader with the methods. As a much in detail and complete reference of automatic learning methods, we invite the reader to consider [HTF01].

[2] In the sequel, each time we speak about the (mean) squared / absolute error or error rate, we refer actually to the empirical error.

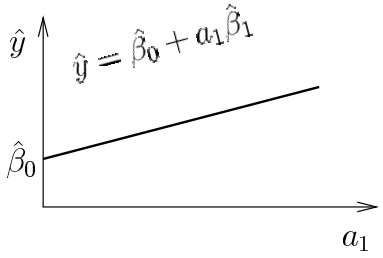


Figure 2.1: An example of a linear regression model

Linear regression

Hypothesis space. Given a vector of inputs $\mathbf{a} = (a_1, a_2, \ldots, a_m)$, the output y is predicted via the linear model

$$\hat{y}(o) = \hat{\beta}_0 + \sum_{j=1}^{m} a_j(o)\hat{\beta}_j, \qquad or \qquad \hat{Y} = A\hat{\beta} \tag{2.9}$$

where

$$\hat{Y} = \begin{bmatrix} \hat{y}(o_1) \\ . \\ . \\ . \\ \hat{y}(o_N) \end{bmatrix}, \quad A = \begin{bmatrix} 1 & a_1(o_1) & \ldots & a_m(o_1) \\ . & . & \ldots & . \\ . & . & \ldots & . \\ . & . & \ldots & . \\ 1 & a_1(o_N) & \ldots & a_m(o_N) \end{bmatrix}, \quad \hat{\beta} = \begin{bmatrix} \hat{\beta}_0 \\ . \\ . \\ . \\ \hat{\beta}_m \end{bmatrix} \tag{2.10}$$

and the term $\hat{\beta}_0$ is known as the *bias* in machine learning. Figure 2.1 depicts a linear model with $m = 1$.

Error function. In conformity with the most popular method called *least squares*, the linear model is fit to the learning set by optimizing the vector $\hat{\beta}$ of parameters based on the minimization of the squared error

$$E(\beta) = \sum_{i=1}^{N} (y(o_i) - \hat{y}(o_i))^2. \tag{2.11}$$

Search strategy. $E(\beta)$ is a quadratic (semi) positive definite function of the parameters $\hat{\beta}$, and hence its minimum always exists. If $A^T A$ is nonsingular, then the unique solution is given by

$$\hat{\beta} = [A^T A]^{-1} A^T Y. \tag{2.12}$$

The main strength of linear regression is its computational efficiency for reasonable input spaces. When the input space dimension m is high, the method is less efficient due to an inverse matrix computation that is cubic in m. The structure of the model is simple and easy to interpret: the parameters tell us the effect of changing any of the input variables "keeping the others constant". But due to the fixed structure, the model also may not be able to model a lot of problems that need models with more degrees of freedom in the choice of the structure.

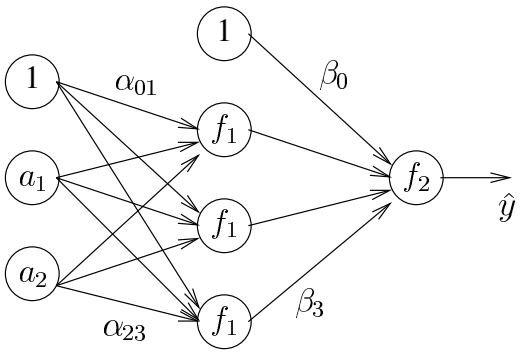


Figure 2.2: An example of a single hidden layer MLP for regression

Multilayer neural networks

Neural networks [Bis97, Weh98, Zur92] are models trying to simulate the activity of a human brain. Feedforward multilayer perceptrons (MLPs) are the most widely used artificial neural models in supervised learning. The MLP structure provides a nonlinear mapping from a numerical input vector to a numerical output vector and it may be used as a nonlinear model for regression problems, as well as for classification, in which case the outputs are modeling the probability of classes. Unlike methods like regression trees, neural networks are *smooth* functions of real-valued parameters. The main strength of MLP is its *universal approximation capability* of being able to approximate any function (input-output mapping). MLP is among the most accurate automatic learning methods. Unfortunately, from the point of view of interpretability it is perceived as a *black-box* tool. Lately, a lot of efforts are paid for expanding the transparency and interpretability character of neural networks [DAG01, SSR01]. It is less effective for problems where the goal is to describe the physical process that generated the data and the role of each input. It is heavy in terms of computational complexity concerning the learning stage, and may become cumbersome for large dimension input spaces. That is why it is advisable to be used in conjunction with other methods that firstly reduce the input space, like tree-based approaches.

Hypothesis space. A structure of a MLP consists in multiple layers of neuron units, called perceptrons. The first layer is the inputs layer, the last layer is the outputs layer. In between are one or more hidden layers. In practice, most of input-output mappings can be well approximated by a MLPs with a single hidden layer. Figure 2.2 illustrates such a MLP. Input layer neurons are fed with input attributes, hidden and output layers' neurons receive linear combinations of outputs from the neurons in the preceding layers. This is called *forward* stage, when outputs are computed starting from inputs. An additional constant equal-1 input unit is sometimes included in the input and/or in the hidden layers as a bias term. There are fully or partially connected networks function of the topology of the interconnections between neurons. Typically, for a regression problem there is only one output neuron, for a n-class classification, there are n units in the output layer.

Given a vector of inputs $\mathbf{a} = (a_1, a_2, \ldots, a_m)$, the output y is predicted via the nonlinear model

$$\hat{y}(o) = f_2(\hat{\beta}_0 + \sum_{j=1}^{K} \hat{\beta}_j f_1(\hat{\alpha}_{0j} + \sum_{i=1}^{m} \hat{\alpha}_{ij} a_i(o))) \tag{2.13}$$

where K is the number of the neurons in the hidden layer, α_{ij} and β_j for $i = 0, \ldots, m$, $j = 0, \ldots, K$ represents the parameters to be learned, named also *weights* and f_1 and f_2 are called *activation functions*, and are chosen usually as one of the following:

- *identity* function

$$f(x) = x,$$

- *sign* or Heaviside *step* function

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- *piecewise linear* function

$$f(x) = \begin{cases} 1 & \text{if } x > \frac{1}{2} \\ 0 & \text{if } x < -\frac{1}{2} \\ x + 0.5 & \text{if } -\frac{1}{2} \leq x \leq \frac{1}{2} \end{cases}$$

- or *sigmoidal* function

$$f(x) = \frac{1}{1 + e^{-sx}}$$

with s a fixed parameter controlling the activation rate.

Therefore, the hypothesis space of a MLP is defined by the number of neurons, number of hidden layers, interconnections and the shape of the activation functions.

In the more general case of feedforward neural networks, neurons may be of two types:

- *summing units* (as in our MLP example), since the activation function is applied to a sum of products input-weight. Here, the sign activation function is characteristic to the *threshold unit*, known also as McCulloch-Pitts neuron or Perceptron, the sigmoidal function is characteristic to the *sigmoidal unit* and the identity function - to the *linear unit*.

- *multiplicative units* which multiply their inputs instead of summing them [Sch00]; thus instead of

$$f_1(\hat{\alpha}_{0j} + \sum_{i=1}^{m} \hat{\alpha}_{ij} a_i(o))$$

in the hidden layer, for example, we have

$$f_1(\hat{\alpha}_{0j} + \sum_{i=1}^{m} \hat{\alpha}_{ij} \prod_{i=1}^{m} a_i(o)^{d_i})$$

called a *higher-order unit* (or a *sigma-pi unit*), or

$$f_1(\prod_{i=1}^{m} a_i(o)^{\hat{\alpha}_{ij}})$$

called a *product unit*, where d_i is the *degree* of variable a_i. Linear units are mainly used as output nodes of networks that approximate real-valued functions. The standard neural network containing product units is an architecture with one hidden layer such as shown in figure 2.2 where the hidden nodes are product units and the output node is a sigmoidal unit.

Error function. The most widely used function is again the mean squared error

$$E(\alpha, \beta) = \frac{1}{N} \sum_{i=1}^{N} (y(o_i) - \hat{y}(o_i))^2. \tag{2.14}$$

Search strategy. The optimization of the parameters (called *training* in Artificial Intelligence

literature) is a search of a global minima of the function $E(\alpha, \beta)$. Depending highly non-linearly on its parameters, this error function has generally multiple solutions, (the error surface presents multiple local minima). The original training method proposed for MLPs known as *fixed step gradient descent* essentially starts from initial conditions (values of parameters) and performs a steepest-descent on the error function in the space of the parameters so as to get closer to a local minimum, updating the parameters at every iteration based on a gradient calculus

$$\hat{\gamma}^{new} = \hat{\gamma}^{old} - \eta \nabla_{\gamma} E(\gamma)|_{\hat{\gamma}^{old}} \tag{2.15}$$

where γ refers to any of the parameters to be optimized (α_{ij} and β_j) and η is a (constant or variable decreasing to zero) learning rate. The neural network can be trained either *on-line*, updating the weights after each object in learning set, or in *batch* mode, updating the weights after seeing the whole learning set of objects.

The backpropagation algorithm is an efficient method to compute the gradient. This is done in a backward manner, starting from output layer and roaming the network until ending with the input layer. The weak points of this optimization technique are: the dependence on the choice of starting points in the error space, the instability of the convergence towards the minimum, trapping in local minima and slowness. Other alternative optimization methods, improved in certain of these aspects, are *conjugate gradient* methods (like Fletcher-Reeves and Polak-Ribiere algorithms), *variable metric methods*, called also quasi-Newton (like Broyden-Fletcher-Goldfarb-Shanno algorithm), or finally, more bulky computational *second-order* derivatives methods (like Newton method). The first two classes of mentioned methods avoid explicit computation of the second derivative matrix while still providing faster convergence.

Decision and regression trees

Machine learning field [LC84, MCM83, MBK98, MST94, Weh94, Weh98] is basically concerned with the automatic design of *if-then rules* similar to those used by human experts. Decision tree induction is the most known and developed method of machine learning. The interest in decision trees comes from their most important feature which is the capability to break down a complex decision-making process into a collection of simpler decisions, hence providing a solution that is easier to interpret. They are in particular the most used methods in data mining. They are computationally efficient, and able to perform feature selection as an integral part of the building procedure, that is why they are used with success in large dimension input spaces. Their counterpart is their predictive accuracy that may be however improved by techniques presented in section 2.1.6.

The most popular in the field are methods like ID3 [Qui86] and its later versions, C4.5 and C5.0 [Qui93, Qui96] for building decision trees and CART [BFOS84] for building both regression and classification trees.

Tree-based methods like decision and regression trees are methods able to partition the input space into a set of rectangles and then approximate the output in each rectangle by a constant, either a class in the case of decision trees, or a number in the case of regression trees. The goal is to recursively split the input space into non-overlapping subregions of objects which have the same class (decision trees case) or the same output value (regression trees case). What one obtains usually are rectangle regions which, speaking about decision trees, are almost pure (the majority of the objects has the same class), and speaking about regression trees, are regions where the output value is almost constant (the distribution of the output values in a rectangle is not far from the mean, thus having a small standard deviation). The effect of this partitioning in regression trees is the lack of smoothness of the prediction surface which degrades performance in the regression setting where one normally expects the function to be smooth.

Hypothesis space. The structure of a tree-based model is given by a connected acyclic finite

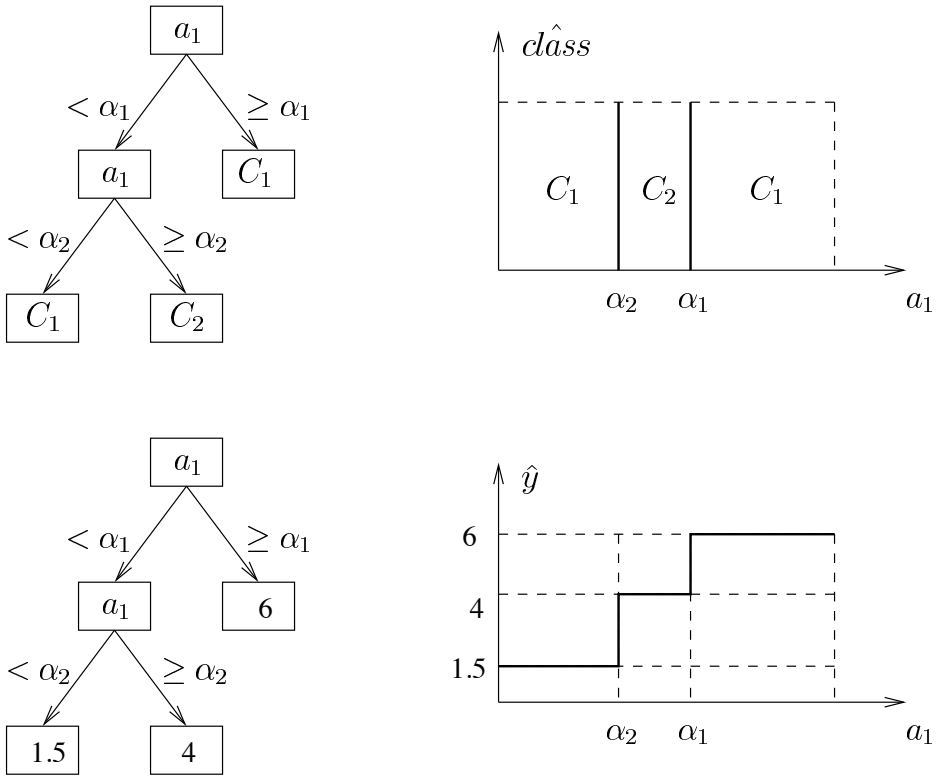


Figure 2.3: Example of a decision and a regression tree

graph like the ones exemplified in the left part of figure 2.3: a decision tree in the upper part, a regression tree in the lower part of the figure. The most upper node is called *root node* and contains the whole learning set of objects. This node has some successor nodes, two in the case of binary trees, the learning set being divided into two subsets of objects based on a *test* applied to an attribute. The test has the form $a < \alpha$ when attribute a is numerical, where α is a discretization threshold for attribute a, and $a \in S$ when attribute a is symbolic, where S is a subset of the finite set $\{a_1, a_2, \ldots, a_m\}$ in which the attribute a takes values [3]. The two *branches* that link the parent node to its successors correspond to confirmation and invalidation of the test in the parent node, sometimes labeled *yes* and *no*. A node that the algorithm decides is pure or pure enough is called *terminal* node and does not have any assigned test nor successors.

Taking the examples of figure 2.3, both trees have two test nodes based on attribute a_1 and three terminal nodes. Thus, the input space is automatically split into three, in this example, regions of objects, corresponding to the three terminal nodes. For the decision tree one obtains three class demarcation rectangles, for the regression tree - an approximation of the numerical output (see the right part of figure 2.3 with respectively the estimation of the class and the numerical output).

The hypothesis space of a tree-based model is a family of structures. A structure of a tree-based model is determined by the graph of the tree and by the attributes attached to the tree test nodes. The discretization thresholds (cutting points) of all these attributes (thresholds α_1 and α_2 in the case of figure 2.3) together with the labels of all the terminal nodes (assigned classes or output estimations) represent the parameters of the tree-based model.

There is a search over both structure and parameter spaces so as to learn a model from experience.

[3] In the rest of the thesis, unless specified explicitly, we will always consider numerical attributes.

Search strategy. The top-down approach of divide-and-conquer strategy recursively partition the given set of objects into smaller and smaller subsets thus growing the tree. Each subset of objects is defined by a sequence of tests. The splitting strategy is based on a *score* maximization and finds in a tree node a pair attribute-threshold that discriminate the best between objects of different outputs of that node. This is done in two steps: firstly, for each attribute from a list of *candidate attributes* the split point (the threshold) that maximizes the score is found, then secondly, the optimal pair attribute-threshold is chosen, among all the pairs attribute-threshold. Having found the best split, the data is partitioned into two resulting regions (successor nodes). If the node is pure enough in terms of the output value or class, the termination criterion is accomplished and the splitting process stops, the node being considered terminal, and labeled. If the termination criterion is not satisfied, the process is repeated on each of the new regions (nodes). In theory, the *stop splitting criterion* prevents the tree from being unnecessarily too large thus overfitted, or not enough developed, thus underfitted, i.e. in both cases suboptimal. In practice, *pruning* (explained below) is the preferred alternative, of adaptively choosing the optimal tree size from data, for the reason that stop splitting criterion is based on user parameters that need to be a priori settled, are problem dependent, and the user have no guaranty that the chosen ones are the best ones.

Three rules define the methodology of building a tree-based model: the rule of selecting the best split at every node, a rule for determining when a node should be considered terminal and a rule for assigning a label to every identified terminal node. In order to determine the tree estimated output for a new object, the object is dropped into the root-node and follows a single path, the one imposed by applying the tests encountered on the path. Finally, a terminal node is reached and the node's label dictate the output or class for the object.

As an exceptional case, the early ID3 algorithm is limited to processing only categorical attributes or numerical attributes discretized *a priori* to tree building, each node tree corresponding to a non-categorical attribute and each arc to a possible value of that attribute. A path from the root to a terminal node admits only one single time a given attribute. The C4.5 extension algorithm is able to treat missing (unavailable) values, continuous attribute value ranges, pruning, rule derivation. An if-then rule may be derived from the tree for each path, from the root to the leaf, the rule antecedent (the left-hand side) being a conjunction of path' tests.

There are a lot of variants of classical decision trees, for example: the *incremental* decision trees [Utg89a] called ID5R where the learning objects are presented serially to the induction process, *oblique* decision trees where in the tree tests, a linear combination of attributes is allowed, *model trees* which are a type of decision trees with linear regression functions at the leaves [FWI+98], or *discriminant trees* that combine a decision tree with a linear, quadratic or logistic discriminant [Gam99].

Error function. The score measures involved in the selection of the best split in a tree node in the growing step evaluate the node impurity and may be classified into four categories (references [Min89b, BN92, Geu02] make empirical comparisons of the most used selection measures):

- based on *information* or *Shannon's entropy*, like *information gain* in ID3 [Qui86] (called also *mutual information* or *apparent information*), *gain ratio* in C4.5 [Qui93], *normalized information gain* proposed by Wehenkel [WP93], and *Gini index of diversity* in CART decision trees [BFOS84]. The Gini index in the case of CART regression trees translates in a measure of the variance reduction that a node provides[4].

- based on *error* (in CART regression trees [BFOS84]) which measures the reduction in (resubstitution) error that the split would produce, or the reduction in variance the node will produce.

- based on *statistical significance* [Min89b] where potential tests are ranked by the confidence with which one can reject the hypothesis that they are irrelevant to the class of objects in the

[4]In a two class problem, the Gini index is equal to the node variance computed by assigning value 0 to objects in one class and value 1 to objects in the other class.

learning set.

- based on *Kolmogorov-Smirnov distance* [Fri77] counting for the maximal distance between two empirical class conditional cumulative distributions of an attribute in the node to be split.

2.1.4 Bias and variance

Bias-variance error decomposition

Let us adopt the learning problem

$$y = f(\mathbf{a}) + \epsilon \tag{2.16}$$

where the output is corrupted by noise ϵ, noise distributed according to a distribution of zero mean and σ standard deviation. We seek for the true value of the function $f(\cdot)$. $f(\mathbf{a})$ represents the *Bayes model*, i.e. the best possible model one can get for this learning problem according to a mean squared error minimization. In terms of probabilistic expectation[5],

$$f(\mathbf{a}) = E_{y|\mathbf{a}}\{y\}. \tag{2.17}$$

Let us note $\hat{f}(\mathbf{a})$ the output estimated by the model built from a randomly chosen learning set $LS \subset U$ of given size N, at a point $\mathbf{a}$ of the input space.

If we consider a regression learning algorithm, then the expected value[6] of the model mean squared prediction error MSE for a fixed input attribute vector, can be decomposed in three components [GBD92]:

$$E_{LS}\{MSE(\mathbf{a})\} = E_{LS}\{E_{y|\mathbf{a}}\{(y - \hat{f}(\mathbf{a}))^2\}\} = residual\_error(\mathbf{a}) + bias^2(\mathbf{a}) + variance(\mathbf{a}) \tag{2.18}$$

where

$$\begin{aligned} residual\_error(\mathbf{a}) &= E_{y|\mathbf{a}}\{(y - f(\mathbf{a}))^2\} \\ bias^2(\mathbf{a}) &= (f(\mathbf{a}) - \bar{\hat{f}}(\mathbf{a}))^2 \\ variance(\mathbf{a}) &= E_{LS}\{(\bar{\hat{f}}(\mathbf{a}) - \hat{f}(\mathbf{a}))^2\}. \end{aligned} \tag{2.19}$$

$\bar{\hat{f}}$ represents the average model which outputs the average prediction over the set of all possible learning sets LS of size N, $LS \subset U$, estimated as

$$\bar{\hat{f}}(\mathbf{a}) = E_{LS}\{\hat{f}(\mathbf{a})\}. \tag{2.20}$$

$residual\_error$ equals $\sigma^2(\mathbf{a})$ (i.e. the conditional variance (given $\mathbf{a}$) of the random noise ϵ) and represents the expected error of the Bayes model, thus the minimum error one can get training a model of unrestricted complexity, hypothesis space and amount of data concerning the learning problem. Given a learning problem and no matter the learning algorithm used and the training

[5] $E_{\mathbf{a}}\{f(\mathbf{a})\}$ denotes the expectation of the function $f(\mathbf{a})$ with respect to the distribution of the random variable $\mathbf{a}$ and it is defined as

$$E_{\mathbf{a}}\{f(\mathbf{a})\} = E_{o \in U}\{f(\mathbf{a}(o))\} = \sum_{o \in U} P(o) f(\mathbf{a}(o)).$$

$E_{y|\mathbf{a}}\{f(y)\}$ denotes the expectation of the function $f(y)$ with respect to the distribution of the random variable y given $\mathbf{a}$ and is defined as

$$E_{y|\mathbf{a}}\{f(y)\} = E_{o \in U|\mathbf{a}(o)=\mathbf{a}}\{f(y(o))\} = \frac{\sum_{o \in U|\mathbf{a}(o)=\mathbf{a}} P(o) f(y(o))}{\sum_{o \in U|\mathbf{a}(o)=\mathbf{a}} P(o)}.$$

[6] an average over repeatedly realized learning sets of the same size N from the learning problem.

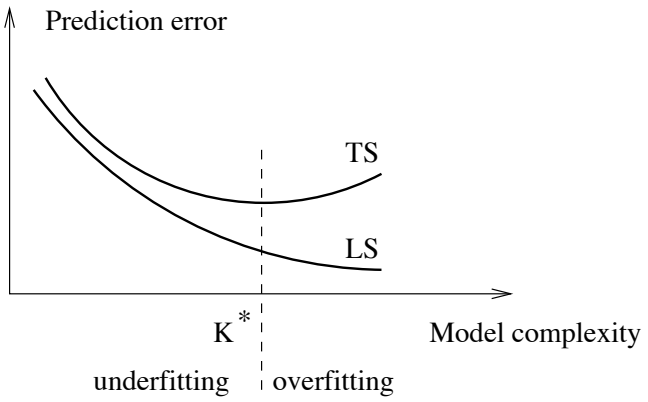


Figure 2.4: Learning and test error as a function of complexity

sample, this is the lowest error one can get in the best case. It reflects the irreducible prediction error due to the random nature of the output variable $y(\cdot)$ and is beyond control.

The *bias* error reflects the persistent or systematic error that the learning model is expected to have on average when trained on learning sets of the same finite size. It measures the difference between the Bayes model and the average model. It reflects essentially sensitivity of the model to the target function $f(\cdot)$.

The *variance* error reflects the variability of the model induced by the randomness of the learning set used. It reflects sensitivity of the model estimate to the learning set LS, by comparing for each randomly drawn LS, the model output estimation with the average estimation. Less sensitivity means that the output estimation will be more stable against changes in the data and less variable from one sampling to the other.

Researchers distinguish two types of bias: *exclusive* or *representational* bias present when, through constrains of representational order, the learning algorithm does not consider certain concepts, which cannot be expressed by it, and *preferential* or *procedural* bias present when the algorithm results in a preference for one class of concepts over another. A *weak* bias is the one that allows the learning algorithm to consider a relatively large hypothesis space (i.e. model complexity). A *strong* bias directs the model towards a relatively small hypothesis space. A strong exclusive bias accelerates learning by reducing the learner's search space. If one restricts the representational power of the learner too severely, the target concept may lie outside of what the learner can represent.

In the case of a classification learning algorithm, if residual error, bias and variance are defined by analogy with the regression case, they do not sum up to give the expected classification error, however theoretically, the behavior of these quantities is similar [DK95, Fri97].

Bias and variance tradeoff and the overfitting phenomenon

From eq. (2.18) one may remark the interest in reducing as much as possible both bias and variance terms, given that the term of the residual error cannot be reduced for a given problem. Since generally, reducing variance increases bias and vice versa, there is a *bias-variance tradeoff* in order to improve the error prediction for the learning models.

The principal factor in this tradeoff is the complexity parameter of the learning algorithm. Generally, we speak about an optimal complexity K^* of a model when the best bias-variance tradeoff has been found. If the model is less complex than this optimal threshold, we are in a zone where the model underfits the data, where estimated prediction error is high both on learning set LS and

test set TS. The model has a too simple structure and the source of error is mostly coming from a large bias. If the model is more complex than this optimal complexity, the model overfits the data used for learning. *Overfitting* appears when there is not enough data available for learning a complex concept or, in other words, when the model is too complex with respect to the information present in the learning set. It is a phenomenon masked by a small value of the prediction error on LS as figure 2.4 shows. An AL model is built based on the estimation of an error function on the learning set sample LS, but in practice, good performances of the model on the learning set do not guaranty good generalization performances as well on another sample containing objects different from the ones used for learning, for example test set TS. In this zone of overfitting, the most important source of error is the variance which determines fluctuations of the prediction error on different learning samples.

Therefore, increasing complexity will decrease bias and increase variance. Note also that at fixed complexity, the smaller the learning set size, the stronger the variance, while bias does generally depend less strongly on the learning set size. The best bias-variance tradeoff with respect to the model complexity is also problem dependent since bias increases with the complexity of the Bayes model $f(\mathbf{a})$ while variance remains essentially constant and since variance is increasing with the noise level ϵ while bias remains essentially constant. And finally, bias-variance tradeoff differs from one learning algorithm to another. [GOW01] performs an empirical comparison study of different learning algorithms on a non-linear regression problem. Variance is very high and bias small but not negligible in a tree-based method. With respect to a decision tree, a MLP has often both bias and variance smaller. Variance is typically negligible and bias is often important in a linear model, and so on.

Model stability

Stability is defined by [Tur95] as the degree to which a classification algorithm generates repeatable results, given different batches of data from the same process. Breiman also remarks in [Bre96] that the decision tree classifier produced by the CART algorithm can vary substantially when a small number of training examples are added or deleted from the learning set and he calls this also *instability*, pointing out that many other learning algorithms, including C4.5 algorithm and the error backpropagation algorithm are all unstable in this sense. This instability affects not only the structure of a decision tree but also the classification decisions made by the tree. Two runs of a tree in slightly different data sets will typically disagree on the classifications of many test-set cases.

Many researches have set as an important goal for the field to find biases that work well for a broad range of learning tasks. But once discovered the notions of variance and stability, they understood that the question of identifying the right hypothesis space is not a correct approach since due to the variance, we may find the right hypothesis space and yet still obtain high error rates.

As we say in the preceding section, the fundamental source of instability is a too large hypothesis space i.e. a too complex model. This is an *informational* instability, instability caused by the lack of information [Die96]. Some learning algorithms also suffer from *computational* instability. For example, in a backpropagation algorithm initialized with small random weights we get an instability of the search process.

An implication of the decision tree instability is that the rules produced by a tree could change very substantially if the learning set changed slightly, fact which straddles the trust users have in these tools, since the understandability is the “trump card” of decision tree models.

Bias and variance estimation

Since we do not have at disposal the explicit knowledge of $P(o), \forall o \in U$ for a learning problem, bias and variance quantities must be estimated based on the available sample of objects. To this end, we divide this amount of data into two parts: one called *pool set* of size P, sufficiently large so as to be able to draw randomly from it *without replacement* q learning samples LS_j of size N, with $N \ll P, j = 1 \dots q$, and a second one called test set TS of size M independent of pool set, used as a test sample to estimate error, bias and variance quantities. The larger the pool set with respect to the learning set size and the larger M, the better the estimation.

If $\hat{f}_{LS_j}$ is the learning model built from the learning sample LS_j, then the estimated mean squared error and variance of the learning algorithm are approximated as[7]:

$$MSE_{algo} = E_{\mathbf{a}}\{MSE(\mathbf{a})\} \simeq \frac{1}{M}\sum_{i=1}^{M}[\frac{1}{q}\sum_{j=1}^{q}(y(o_i) - \hat{f}_{LS_j}(\mathbf{a}(o_i)))^2] \tag{2.21}$$

$$variance_{algo} = E_{\mathbf{a}}\{variance(\mathbf{a})\} \simeq \frac{1}{M}\sum_{i=1}^{M}[\frac{1}{q}\sum_{j=1}^{q}(\bar{\hat{f}}(\mathbf{a}(o_i)) - \hat{f}_{LS_j}(\mathbf{a}(o_i)))^2] \tag{2.22}$$

where the estimated average model is computed as

$$\bar{\hat{f}}(\mathbf{a}(o_i)) = E_{LS}\{\hat{f}_{LS}\} \simeq \frac{1}{q}\sum_{j=1}^{q}\hat{f}_{LS_j}(\mathbf{a}(o_i)). \tag{2.23}$$

Because we do not know the Bayes model $f(\mathbf{a})$ in the case of *real* learning problems, algorithm bias can not be calculated separately from the residual error only from data. However, in order to study at least bias relative evolution, either we assume no noise, i.e. $f(\mathbf{a}(o_i)) = y(\mathbf{a}(o_i))$, or we compute the sum of the residual error and bias terms, since the residual term in this sum is a constant, as:

$$bias^2_{algo} + residual\_error \simeq \frac{1}{M}\sum_{i=1}^{M}(y(o_i) - \bar{\hat{f}}(\mathbf{a}(o_i)))^2 \tag{2.24}$$

or even easier, as the difference between the other two already calculated terms:

$$bias^2_{algo} + residual\_error = MSE_{algo} - variance_{algo}. \tag{2.25}$$

Parameter bias and variance

Denoting by γ_{LS_j} a parameter of a learning model built from a learning sample LS_j of size N, the *parameter bias* of the model represents the error the parameter has in average when the model is trained on learning sets of the same size, whereas the model *parameter variance* reflects the variability of the parameter induced by the randomness of the learning set used. They can be written as

$$variance_{\gamma} = E_{LS}\{(\bar{\gamma} - \gamma_{LS})^2\} \simeq \frac{1}{q}\sum_{j=1}^{q}(\bar{\gamma} - \gamma_{LS_j})^2 = \sigma^2_{\gamma} \tag{2.26}$$

$$bias_{\gamma} = \gamma_{asymptotic} - E_{LS}\{\gamma_{LS}\} \simeq \gamma_{asymptotic} - \bar{\gamma} \tag{2.27}$$

[7] We note $E_{\mathbf{a}}\{\cdot\}$ also as $E_{o\in U}\{\cdot\}$ or simply $E_U\{\cdot\}$

where the averaged value of parameter is

$$\bar{\gamma} = E_{LS}\{\gamma_{LS}\} \simeq \frac{1}{q}\sum_{j=1}^{q}\gamma_{LS_j} \tag{2.28}$$

and $\gamma_{asymptotic}$ is an asymptotic value of γ_{LS} when N approaches infinity, which can be estimated from the whole amount of available data.

2.1.5 Sources of variance in tree-based methods

[Geu02] finds based on empirical studies that more than 50% of the error can be attributed to variance in a decision tree and less than 30% may be attributed to bias. Variance being the dominant source of prediction error in a tree-based method, we are motivated to analyze its causes. They are:

- For numerical attributes, the discretization procedure produces thresholds with very high variance, even for large local learning sets, independently of the score measure used. This variance increases with the depth of the tree and decreases with the local learning set size. It represents more than 50% of the whole variance in a decision tree [Geu02].
- The attribute chosen in a test node depends strongly on the local learning set since scores at test nodes are sometimes very close or equal for one or more attributes and/or for different thresholds. The addition or deletion of a single learning object in/from the local learning set could alter the local decision and thus change the test in the node. This induces the variability of the whole tree structure, nodes disposal and tree depth. This variability is problem dependent.
- The variability of the structure is determined also by the stop splitting criterion.
- The output estimated based on the local learning set at a terminal node (class probabilities based on frequency counts in decision trees or numerical output based on averaging in regression trees) is subject to variance especially on small learning samples.

Moreover, since the recursive partitioning of the input space in non-overlapping regions of objects provides at tree nodes smaller and smaller local learning sets on which the choice of the local best pair (attribute, splitting threshold) is based, variability of choices and estimates at upper nodes propagates and amplifies when moving down at lower nodes, thus variance increases with the tree depth and complexity.

2.1.6 Methods to reduce variance in tree-based methods

As we saw, reducing variance translates in improved accuracy, provided that it is not obtained at the expense of a much higher bias. There are several possible techniques.

Firstly, there are techniques employed locally for reducing discretization threshold variance and/or attribute selection in a node [Weh97, GW00, Geu02], that succeed also to reduce global variance of the tree, but unfortunately the effect of global variance reduction is partly dissolved by an increase in global bias. The principal gain is however the improvement of the tree interpretability without any loss in accuracy.

Secondly, there are pruning techniques that in decision trees contrive to reduce variance by introducing bias, and the effect is possibly only a slight reduction of test error rates. Nevertheless, the model complexity is well adapted to data and the overfitting phenomenon is avoided.

Thirdly, aggregation techniques called also ensemble techniques by [Die00a] that aggregate by averaging or voting a set of models perturbed in different manners, in order to get a model with much less variance without changing significantly the bias.

Pruning

The idea behind pruning consists in reducing the size of the hypothesis space in such a way that we reduce variance enough without increasing too much bias.

By pruning, an appropriate size of the hypothesis space is determined, like the number of inputs in a linear regression, the number of terminal nodes for a decision tree or the number of hidden neurons in a multilayer perceptron, so as to change the bias-variance tradeoff, and if possible, to improve it. The most implemented pruning methods are two-stage methods which proceed as follows: firstly, a sequence of hypothesis spaces is considered and in each space a model is selected based on the learning set; secondly, the obtained models are evaluated in terms of their generalization capabilities and one is chosen. There exists also single-stage methods which in a forward or backward manner enlarge or coerce the input space.

In linear regression, *shrinkage* methods like *ridge regression* for example, shrinks the regression coefficients by imposing a penalty on their size and this is a way of discarding unnecessary input variables. Other methods transform the set of original inputs in a small number of linear combinations of these inputs. Or simply, the best subset of inputs is selected by surveying the evolution of the squared error of eq. (2.11) when adding or deleting an input from the model.

Pruning of less important links and/or hidden nodes in a neural network may be considered in order to get a near optimal network architecture by eliminating possible redundancy in the network. *Weight decay* is a shrinkage method which penalizes large weights, *sensitivity* pruning prunes the nodes with low sensitivity, and *cyclic* pruning removes links based on some user defined parameters [SSR01].

Pruning trees. In the case of a tree-based learning model, a two-stage pruning procedure consists in growing a *complete tree* by relaxing the condition for stop splitting, and then by removing irrelevant parts of the tree generally in a bottom-up fashion. The underlying idea of tree pruning is that, among two trees of the same generalization capability, the simpler one is preferred. The chosen tree is called the *right sized tree*. The two main issues which arise when searching for the best pruned tree are: the method used to explore the large space of all possible pruned trees and the method to evaluate the different obtained alternatives. Pruning is also called *post-pruning* in opposition to the *pre-pruning* approach which consists in stopping the development of a tree according to a stopping criterion before the tree becoming complete.

We will refer here to three categories of pruning methods.

1. Method based on an error-complexity compromise. The pruning task is a three step process:

- Generate a set of "good" pruned trees.
- Obtain reliable estimates of the generalization error of these trees.
- Choose one of these trees according to the estimates.

Thus, a sequence of pruned trees is generated and the tree with the best performance estimated by the *holdout* method on an independent set called *pruning set* PS (a holdout set), or by the *cross-validation* method (explained below in section 2.1.7), is chosen.

The measure that counts for the "goodness" of a given subtree, is called *quality* in Wehenkel's decision trees [Weh92a], *error-complexity* in CART regression trees, or *cost-complexity* in CART classification trees. It is a compromise between a component that measures the goodness of fit of the model to the learning data in regression or the precision with which the tree is able to classify seen objects in classification, and a component that penalizes model complexity (i.e stresses on simplicity)

$$Q \propto err_{LS} - \beta K, \tag{2.29}$$

where err_{LS} is the (resubstitution) error estimate of the tree, or the total information provided by the tree, or the total variance reduction provided by a regression tree, all computed on the

learning set, K is the number of terminal nodes and β is a user defined complexity parameter viewed as a cost of one terminal node. The interesting and necessary property of this quality measure is the *additivity* property, with respect to the decomposition of a tree into subtrees: for any such decomposition, the total quality of the overall tree is equal to the sum of the qualities of its subtrees.

The CART pruning algorithm is called an *optimal pruning algorithm* since their authors proved that each tree from the sequence is optimal from the perspective of the error-complexity compromise within a certain interval of cost-per-terminal-node values (this still does not mean that each tree from the sequence is optimal from the perspective of the true prediction accuracy).

Concerning the choice of the best tree, *"One-standard-error-rule"* (1SE) is a popular method [BFOS84, Weh98]. This rule consists of evaluating the generalization error (e.g. Pe, MAE, MSE) of each tree in the sequence in some fashion, evaluating also the standard error estimate of this generalization error and then selecting among the trees not the one of minimal error but rather the smallest tree from the sequence of trees whose error is at most equal to the best error from the sequence plus its standard error:

$$error^* \leq error_{best} + S.E.(error_{best}) \tag{2.30}$$

where the standard error S.E. is defined below in section 2.1.7.

In the more general "Standard-error-rule" there is a *multiplier* σ which multiplies the standard error in formula 2.30. If $\sigma = 0$ the rule chooses the tree with the best error, if $\sigma = 2$, the rule chooses the smallest tree with the error not larger than the best error plus 2 times its standard error, and so on.

2. Method based on estimates computed directly on the learning set. The *error-based pruning* algorithm of C4.5 uses the learning set not only for building the complete tree but also for pruning it. The novelty here is that the algorithm allows a subtree to be replaced by one of each branches, and not uniquely by a single terminal node. In this way it is possible to remove "intermediate" tests which appears useless. The method is based on estimates of the error rate of each pruned tree on unseen cases, estimates computed however on the learning set not on a pruning set or a test set. These error estimates for terminal nodes and subtrees are computed assuming that they were used to classify a set of unseen cases of the same size as the learning set. The pruning rationale is then: start from the bottom of the tree and examine each test node. If replacement of this subtree with a terminal node, or with its most frequently used branch, would lead to a lower error rate estimate, then prune the tree accordingly. Since the error rate for the whole tree decreases as the error rate of any of its subtrees is reduced, this process will lead to a tree whose estimation of the error rate is minimal.

3. A more general method of nested pruning algorithms. Some pruning strategies (including the one proposed by our FDT method in chapter 3) generate a sequence of trees where each tree is obtained by pruning the previous tree in the sequence at some node. No quality measure is explicitly involved. The procedure can be summarized as follows:

- Establish a list of critical nodes by sorting all the test nodes in increasing order of their "relevance".

- Generate the sequence of subtrees, by contracting nodes following the previous established list of critical nodes.

- Choose one of these trees.

Thus, these methods strongly rely on the strategy to choose the next node to prune, thus on the way the nodes to be pruned are sorted at the beginning. And in general, this strategy does not guaranty that the obtained sequence of trees includes the best possible subtrees of the complete tree.

Other pertinent pruning criteria are based on Rissanen's work on Minimum Description Length,

whose idea behind is that "the simplest explanation of an observed phenomena is most likely to be the correct one", or based on Bayesian methods so as to obtain reliable error err_{LS} estimates.

[Sch93] advocate the idea that "any overfitting avoidance strategy amounts to a form of bias and, as such, may degrade performance instead of improving it". [Min89a] and [EMS97] make a review and an empirical comparison of the most known pruning methods for trees. [Min89a] shows that "there are significant differences between the methods". [EMS97] conclusions that "pruning does not generally decrease the predictive accuracy of the final trees", that some methods have a marked tendency to overprune (e.g. pruning in C4.5) or to underprune (e.g. CART decision tree pruning with 1SE), and that "setting aside some data for pruning only is not generally the best strategy". [Tor99, Tor98a, Tor98b] perform such comparative studies in the special case of regression trees. Their conclusions are that "significant differences in terms of accuracy and tree size were observed by using different error estimation methods" (when pruning regression trees), that "CART regression tree pruning is biased towards smaller trees", and that "there are data sets where pruning is clearly beneficial, there are others where is not so evident".

Aggregation methods

The general idea behind aggregation techniques is that we aggregate multiple unstable (thus bad) models in order to get a stabilized (thus good) one. Here are some three classes of current perturbations (manipulations) applied to the learning algorithms in these types of methods:

- manipulation of the learning set: samples (*bagging*, *boosting*, *arcing* [Bre98, FS98]), inputs, or the output y of a model (*error-correcting output coding* [KD95])
- manipulation of the learning algorithm, like injecting randomness [Die00b]
- manipulation of the test set (*dual perturb and combine*)

These methods are generally valid independently of the learning algorithms on which they are applied. They are easy to implement and give often spectacular results in terms of variance reduction. Albeit the complexity of these methods is high (the complexity of the final model is the sum of the complexities of the intermediate models), the final prediction is prevented from being overfitted. Voting and averaging entail the loss of interpretability and efficiency in tree-models.

We are mostly concerned with these techniques from the tree-based methods perspective. [BK99] and [Die00b] are valuable empirical comparisons of averaging methods applied to decision trees. We discuss below in particular a few aggregation techniques, that are of interest for us, either because they are the most known and effective methods in this class (bagging and boosting), or because they will be compared latter on with results of our method of fuzzy decision trees (dual perturb and combine, and extra-tree averaging). We only give a general view for an intuitive comprehension.

Bagging

The *bagging* (Bootstrap aggregating) algorithm by Breiman [Bre96] votes classifiers or averages regression models generated by different bootstrap samples (replicates). A **bootstrap sample** is generated by uniformly sampling *with replacement* (bootstrap sampling) N objects[8] from a pool set of size N. q bootstrap samples $LS_1, LS_2, \dots LS_q$, are generated and a model m_{LS_i} is built from each bootstrap sample LS_i. A final model m^* is built from $m_{LS_1}, m_{LS_2}, \dots m_{LS_q}$ whose output is the class predicted most often by the classifiers (majority vote):

$$m^*(\mathbf{a}(o_j)) = \arg \max_{c \in \{C_1, C_2, \dots, C_n\}} \sum_{i=1}^{q} I(m_{LS_i}(\mathbf{a}(o_j)), c) \tag{2.31}$$

[8] In this sample an object may have multiple occurrences.

or the average model of all the predictors (used typically in regression)

$$m^*(\mathbf{a}(o_j)) = \frac{1}{q}\sum_{i=1}^{q} m_{LS_i}(\mathbf{a}(o_j)), \tag{2.32}$$

The idea of bagging is to find based on LS an estimation of the average model of equation (2.20), which average model has a zero variance and the same bias as the original algorithm. Bagging is a pure variance reduction technique generally applied to unpruned decision trees which have smaller bias than pruned ones. In parallel environments, bagging has the strong advantage of building the models in parallel. Applied to tree-based methods, bagging destroys the tree interpretability.

For a given bootstrap sample, an instance from the learning set has probability $1-(1-1/N)^N$ of being selected at least once. For large N this is about $1-1/e = 0.632$, which means that each boostrap sample contains about 63.2% of the instances from the learning set [BK99].

Boosting

The *boosting* algorithm by Schapire [FS96, FS99] was designed as a method for boosting the performance of a *weak* learning algorithm. It is considered as one of the most powerful learning ideas introduced in the last ten years. Like bagging, the AdaBoost.M1 algorithm, the most popular boosting algorithm, *sequentially* generates q **weighted** learning samples $LS_1, LS_2, \ldots LS_q$ and q models $m_{LS_1}, m_{LS_2}, \ldots m_{LS_q}$ are built. In each weighted learning sample, each object has a weight allocated based on the previously built model in the sequence (from here comes the interest in building the models sequentially, unlike bagging). The incorrect classified objects are weighted by a factor inversely proportional to the error on the learning set. A final model is formed by using a weighted majority vote, where the role of weights is to give higher influence to the more accurate models in the sequence.

While the perturbations on a given learning set introduced by bagging are random and independent, the perturbations introduced by boosting are chosen deterministically and serially, the nth perturbation depending strongly on all of the previously generated models. Empirical results show that among the averaging methods, boosting (as well as error-correcting output coding) can equally reduce model bias along with the correction in variance. The main problem with boosting seems to be the robustness to noise.

Dual perturb and combine

The *dual perturb and combine* algorithm by Geurts [Geu01] builds one single model with a given learning algorithm and learning sample and then at the prediction stage some random variations are introduced. Multiple predictions are obtained which are then aggregated. It is a particular perturb and combine algorithm, which by Breiman's definition is an algorithm that aggregate (combine) multiple models obtained by perturbing the learning procedure with a set of random parameters. Here the perturbation intervenes at the prediction stage in the form of random noise added at each attribute and obtained from a diagonal Gaussian distribution normalized according to the variance of each input attribute. Perturbations of the model's predictions at each point from the test sample $o_j \in TS$ are obtained by

$$m_{\epsilon_i}(\mathbf{a}(o_j)) = m(\mathbf{a}(o_j) + \epsilon_i), \tag{2.33}$$

The final prediction of the model at point $o_j \in TS$ is

$$m^*(\mathbf{a}(o_j)) = \arg \max_{c \in \{C_1, C_2, \ldots, C_n\}} \sum_{i=1}^{q} I(m_{\epsilon_i}(\mathbf{a}(o_j)), c) \tag{2.34}$$

in classification, or

$$m^*(\mathbf{a}(o_j)) = \frac{1}{q}\sum_{i=1}^{q} m_{\epsilon_i}(\mathbf{a}(o_j)) \tag{2.35}$$

in regression. The amount of added noise ϵ_i, $i = 1, \ldots, q$ is regulated by an user defined parameter λ that seems to be very dependent on the particular induced model and learning task, but it may be automatically optimized. Identifying the optimal value of parameter λ produces an overwork at the learning stage with respect to classical decision trees. Adding Gaussian noise to the attribute vector is more or less equivalent to adding Gaussian noise to the discretization threshold at tree nodes (it is strictly equivalent if there is only one test on the same attribute along each branch of the tree). With respect to other averaging techniques, this method does not require building several models and therefore, preserves the interpretability and efficiency of the original learning model. Variance is reduced although less impressively than in bagging.

Extra-tree averaging

The *extra-tree averaging* algorithm by Geurts [Geu02] averages a set of *extremely randomized trees* (called extra-trees). An extra tree is a classical tree where the splitting rule is special: the splitting attribute is first selected at random, then a threshold is drawn from a Gaussian distribution with mean and standard deviation given by the values of the selected attribute in the local learning set. In order to avoid very bad tests, the chosen attribute is rejected if it produces a value of the (classical) score less than a threshold a priori defined. Therefore, in a node of an extra tree, the discretization is really a random process. A node is terminal only when it is pure. The effect is that we get much more complex trees than with the classical decision tree algorithm, which once averaged produce better results than bagging. This is explained by the reduction in bias together with the reduction in variance. A nice property of this technique is the impressive decrease in computational time. To have an idea, to build 10 such random trees is still more efficient than to build a single classical one. Since the extra-trees can be built in parallel, this method can cope well with *very* large data sets in data mining environments. References [Geu03, MGW03] apply extra-tree averaging to image classification data, task defined by an impressive number of attributes (up to 49.000 attributes in their studies). The time advantage of this method increases with the number of attributes of the analysed data, the method being very well suited to databases with large number of attributes.

2.1.7 Evaluating learning algorithms

There are four important criteria for evaluating an automatic learning algorithm in the context of a data mining approach.

- The most commonly used criterion for evaluation is the *predictive accuracy* called also the generalization accuracy or test set accuracy. We measure/estimate in regression the (mean) absolute/squared error applying eqs. (2.4) or (2.5) (how "far" are the predictions from the real numerical output values) and in classification the error rate of eq. (2.8) (the proportion of incorrect class predictions that a learning algorithm makes on a test data). Since the accuracy can not be measured on non-deterministic learning problems, a lot of work is paid in literature so as to obtain a *reliable estimation* of this accuracy. There are three common ways of estimating the accuracy of a learning algorithm that we sketch in the section below. A more close-up view in classification would reflect also the proportions of predictions for the different classes in data, since two incorrect classifications may have different importance in reality (e.g. a missed alert in the operation of a complex system due to an incorrect result of the learning algorithm may be more serious or costly than a false

alarm). Thus, a more general measure of the performance of an algorithm, would take into account also *the cost of making wrong decisions*.

- The *interpretability* (called also understandability or comprehensibility) of a learning algorithm helps understanding and validating the resulted model with respect to prior knowledge about the field. Moreover, once the model is validated, it enlarges the knowledge about the studied problem.

- Another criterion is the *computational complexity* of the algorithm as a form of a price necessary to be paid for the activity of learning. One or more of the following parameters are influencing usually (linearly, quadratically, or logarithmically) the computational complexity of a model: the size of the learning set, the number of the attributes/inputs, the model complexity, the number of iterations until convergence in an iterative optimization process, etc. Relative CPU learning (training) times are a good measure of the computational burden when comparing algorithms. A very sought-after property of an algorithm is its *scalability*. An algorithm is said to be scalable if "its runtime increases linearly with the number of records (objects) in the input database" [GGR99].

- The *autonomy* of an algorithm is a valuable characteristic. One prefers generally a method that can be directly applied to the data without requiring much preprocessing in terms of data transformation or inputs selection and learning algorithm parameters tuning. Subsequent criteria are here: ability to deal with irrelevant inputs, ability to deal with a mixture of attribute types, ability to handle missing values in data, the cost of acquiring the data in the format that the learning algorithm is able to treat it, or the cost of human intervention in the learning process.

The complexity of a model (e.g. size of tree) is also a relevant property. The Occam's Razor principle - the fewer terms in a model the better - is generally accepted. Thus, between two fitted models presenting the same accuracy, the simpler is preferred. The complexity of a model has also an impact on the model interpretability since the more complex a model the less comprehensible.

Other criteria linked to the concept of data mining are: robustness to noise, robustness to outliers, stability of the algorithm or stability of its parameters (how sensitive the parameters are to small changes in the data), the cost of prediction errors, all these in the context of very large databases. Thus, learning algorithms that cope with all these aspects are preferred. Studies that compare different learning algorithms [LLS00, HTF01, MST94] show that none of the present learning methods does simultaneously act well in conformity with *all* the specified criteria. As a remark, a lot of studies have compared in the literature neural networks with decision trees (see [ACM+90, DHB90, FM88, SMT91, WK89]), since these two learning methods reassemble opposite criteria: a decision tree does not have the very good accuracy of a neural network and a neural network does not congregate interpretability, autonomy and computational cost criteria as a decision tree does.

Estimating models accuracy

There are three generic ways of estimating predictive accuracy of learning methods. They are used both to evaluate different hypotheses in the process of building the model, and to compare different models produced by automatic learning methods.

Resubstitution estimate. The accuracy estimation of eqs. (2.4) or (2.5) and (2.8) is directly computed on the learning sample of objects, the one used for learning. This is a biased estimate since it may give overly optimistic view about the accuracy of a model. As we have seen in figure 2.4 learning error is not a good estimate of the test error. Learning error always decreases

with the model complexity, amounting to zero for sufficient large model complexity. However, this does not mean that the model will generalize well. This estimator is used mostly during the model building procedure.

Hold-out estimate. Called also *test sample estimate* divides data into two subsets: one is the learning set LS used to build the model, the other one is a *hold-out* set H used to test the model and to extract an unbiased estimate. The necessary condition is that the objects in the hold-out set should be drawn from the same distribution as the cases in the learning set and should be independent of them. The hold-out estimate is the average prediction error of the model on the cases in the hold-out set:

$$\hat{E}_H = \frac{1}{N_H} \sum_{i=1}^{N_H} err_i \tag{2.36}$$

where N_H is the number of cases in the hold-out sample, and $err_i = err(\hat{f}_{LS}(\mathbf{a}(o_i)))$ is the prediction error of the model $\hat{f}_{LS}$ build on LS, for object $o_i \in H$.

The standard error of the holdout estimate is given by

$$S.E.(\hat{E}_H) = \sqrt{\frac{\sigma^2_{E_H}}{N_H}} \tag{2.37}$$

where $\sigma^2_{E_H}$ is the variance of the error err.

Examples of standard error estimates are:

$$S.E.(\hat{Pe}) = \sqrt{\frac{\hat{Pe}(1-\hat{Pe})}{N_H}} \tag{2.38}$$

for the error rate in classification and

$$S.E.(\hat{AE}) = \sqrt{\hat{SE} - \frac{\hat{AE}^2}{N_H}} \tag{2.39}$$

for the absolute error in regression, where $\hat{Pe}$, $\hat{AE}$ and $\hat{SE}$ denotes the estimates of the error rate, absolute error and squared error. They are both used when "One-standard-error-rule" is employed in pruning.

This hold-out estimation is employed when we have at disposal sufficient data since for small datasets it may reduce significantly the data available for training. The result is more honest than the resubstitution estimate. Generally, the more test data, the more accurate the error estimate. There is a tradeoff between the size of the two samples LS and H so as to not harm the quality of the learned models and still to get an un-optimistic estimate. Common practice set H as one third of the data, or at least not less than 1/10 of data. In the case of pruning, in order to select the best pruned model among multiple possible models, accuracy is evaluated for each model on a pruning set independent of the learning set and both these sets are separately from the so called test set used to honestly evaluate the final chosen model and to compare it with other models (tested possibly on the same test set).

V-fold cross-validation. It is preferred when we have at disposal very few data and the hold-out method thus cannot be applied. It consists in dividing the whole available data set into V roughly equally sized parts. For each kth set of objects, a model is built using the other $V-1$ sets of data and then tested on this set. The V-fold cross-validation estimator is the average prediction error over the V models and if V is not too small, it is a good estimation of the true prediction error.

Usual choices for V are 5 or 10. With respect to hold-out estimate here we are confronted with a higher computational burden since V models are built instead of one.

When $V = N$, N being the number of all learning states, the method becomes the *leave-one-out* estimate, preferred only in problems with very few cases. Here the test set contains only one single object. The advantage is the maximal use of training data, since we train on $N - 1$ objects. The procedure is deterministic, no sampling being involved. It is unfeasible for large data sets from the point of view of computational cost, as a large number of runs are required.

In classification, the split into learning and test set, either in the case of hold-out estimate or V-fold cross-validation, might be unrepresentative, e.g. a certain class is not represented in the learning set, thus the model will not learn to classify it. The solution is to use *stratified hold-out* or *stratified cross-validation* where one samples so as to maintain in each learning and test set the same proportion of classes (e.g. in a two class problem, each kth set of objects in a 10-fold cross-validation, contains 10% of class A objects and 10% of class B).

Comparing accuracy of different models

Assume we want to compare the performance of two classification learning algorithms A and B on the same data set. We can use cross-validation to determine the error rates of A and B and then compute the difference. The problem is that sampling being involved in cross-validation, there is variance for the error rates. Common sense tells us to be cautious of small samples and highly variable results. The solution is to determine whether the observed difference between the error rates of two fitted models is *statistically significant*.

Differences between two learning algorithms may come from the next four sources of variation [Die98]:

- the random variation in the selection of the *test data* that is used to evaluate the learning algorithms; this source of variation increases with the decrease of the test set size.

- the random classification error which is in fact the Bayes error of the test data, the minimum error one can achieve given the test data (if a fixed fraction η of the test data points are randomly mislabeled, then no learning algorithm can achieve an error rate of less than η).

- the random variation from the selection of the *learning data*. This is what we called model instability in section 2.1.4.

- the internal randomness in the learning algorithm; the algorithms depending on an initial random starting state like neural networks are for example, tend to perform different on identical test and learning data.

A good statistical test should not count for these sources of variation. To diminish the first two sources (test data variation and random classification error) the statistical procedure must consider the size of the test set and the consequences of changes in the test set. To diminish the last two sources (learning set variation and internal randomness) the statistical procedure must execute the learning algorithms multiple times and measure the variation in accuracy of the resulting classifiers.

[Die98] compares five statistical tests for determining whether one learning algorithm outperforms another on a particular learning task. The paper measures experimentally the *error of type I* that these tests are approving, i.e. their probability of incorrectly detecting a difference between the algorithms when no difference exists and also the *power* of the tests, i.e the ability of detecting algorithm differences when they do exist. The studied tests are: McNemar's test, a test for the difference of two proportions, the resampled paired *t*-test, the V-fold cross-validated paired *t*-test and the 5 times 2 cross-validated (5x2cv) paired *t*-test. The conclusions of this study are that the V-fold cross-validated paired *t*-test is the most powerful and that the resampled paired *t*-test has very high probability of type I error, and should never be employed, the results being untrusted.

And a general conclusion says that all the tests described there should be seen as approximate, heuristic tests, rather than as rigorously correct statistical methods. [Alp99] is another paper that proposes a variant, the combined 5x2cvF test, that combines multiple statistics to get a more robust test with lower type I error and higher power.

The most popular statistical test in the automatic learning literature is the V-fold cross-validated paired *t*-test which we have chosen to employ in our simulations.

The V-fold cross-validated paired *t*-test

A *t*-test is a statistical test used to compare the means of two variables in order to verify a statistical hypothesis. These variables are in our case the performances of algorithms A and B. i.e the observed proportion of test objects misclassified by the algorithms in a given trial. We have at disposal only a few samples of these two variables, since we usually perform a few trials from the possible ones given also the fact that we have only a limited sample of data when comparing methods. We want to find if the *null hypothesis* **Ho:** $\bar{A} = \bar{B}$ is true, i.e if the two learning algorithms have in average the same performances, over the whole population of possible objects.

A paired *t*-test consists in running both learning algorithms A and B on the same set of problems and record the difference in performances between A and B, $D = A - B$. Then calculate the mean of these differences $\bar{D}$ and their standard deviation σ_D. A V-fold cross-validated paired *t*-test, randomly divides the data sample into V disjoint sets of roughly equal size and then V trials are conducted, in each trial the test set being one set and the rest V-1 sets being the learning set, the algorithms A and B using the same random division of the data (i.e. the same folds). If we assume that the V differences D_i were drawn independently from a normal distribution, then we can apply Student's *t*-test, by computing the statistic

$$t = \frac{\bar{D}}{\sqrt{\frac{\sigma_D^2}{V}}}, \tag{2.40}$$

where

$$\sigma_D = \sqrt{\frac{\sum_{i=1}^{V} (D_i - \bar{D})^2}{V - 1}} \tag{2.41}$$

computes the standard deviation of the sampling distribution of the mean, often called the standard error of the mean [9].

Under the null hypothesis **Ho**, this statistic has a *t* distribution with V-1 degrees of freedom. The next step is to compare the computed score *t* with the *t* distribution. A *t* distribution is a bell-shaped distribution, looks a lot like a normal distribution but has heavier tails than the normal, more of the mass of the distribution being in the tails. The *t* distribution approaches the normal distribution as V approaches infinity. Table 2.1 gives a fragment of a table of a *t* distribution (a more complete table may be found in [Sap90]). Given the confidence level p and the degree of freedom $V - 1$ one may find in this kind of table the value $t_{1-p,V-1}$. If $|t| > t_{1-p,V-1}$ then the null hypothesis **Ho can be rejected** and there is statistical evidence that *alternative hypothesis* to be true, $\bar{A} \neq \bar{B}$, thus that one learning algorithm out-performs the other.

The null hypothesis **Ho** is rejected "at the 0.05 level", meaning we acknowledge a 5 percent chance of being wrong. Usually, 5 percent is considered as adequate evidence for rejection, 2.5 percent is a "good evidence" and 0.1 percent is considered strong evidence. By convention, one usually does not reject the null hypothesis unless $p > .05$. This is the largest bound on p

[9] The correct estimation of a standard deviation σ^2 is not $\frac{\sum(x-\bar{x})^2}{N}$ but $\frac{\sum(x-\bar{x})^2}{N-1}$. It is true that for large N (large sample size) $\frac{\sum(x-\bar{x})^2}{N-1} \approx \frac{\sum(x-\bar{x})^2}{N}$, but at small N (which is the case here), the $\frac{\sum(x-\bar{x})^2}{N}$ estimation leads to a systematic under-estimation of the standard deviation [PTVF94, Ves58].

Table 2.1: A fragment of a table of *t* distributions (two-tailed)

Degrees of freedom	Confidence level p		
	0.1	0.05	0.01
1	6.314	12.706	63.657
2	2.920	4.303	9.925
5	2.015	2.571	4.032
9	1.833	2.262	3.250
inf	1.645	1.960	2.576

that most researchers will call statistically significant. For example, if $t > 2.262$ in a 10-fold cross-validation, there is a statistical significant difference between the two algorithms, with 95% probability of being truth. Furthermore, if *t* is even larger, say $t > 3.250$ this probability increases to 99% of being confident.

The V-fold cross-validated paired *t*-test has the advantage that each test set is independent of the others and the disadvantage that the learning sets overlap (in a 10-fold cross-validation, each pair of learning sets shares 80% of the objects). This introduces variability in the test as we saw earlier. Thus unfortunately, variance due to learning data variation is still intercepted by the statistical test.

By contrast with parameter estimation where samples should be as large as you can afford, for hypothesis testing, samples should be no larger than required to show an effect. Indeed, in hypothesis testing, the only thing that changes when we increase sample size is our confidence in the conclusion. Any real statistical effect, can be boosted to significance by increasing V. This is because the standard error of any statistic is reduced by increasing V.

The *t*-test lies on the important assumption that the distribution of the sample differences D_i is normal. From here comes the necessity of a minimum size of the folders in a V-fold cross-validation, so as the sample mean of the error rate to follow a Normal distribution. This size should be 30 due to the Central Limit Theorem, fact which is some times violated in practical experiments.

[Coh95] and [Mit97] are two insightful references that treat the subject of statistical evidence when comparing two learning algorithms. *t*-test is treated of course in all books of statistics [MDIS97].

Other pertinent ideas [Elk00] concerning the experiments design in comparing algorithms are: do not compare tuned versions of an algorithm with untuned versions of another, compare a new algorithm against the state-of-the-art, not against an old method, given a test set of a fixed size, performance differences below a certain threshold are statistically meaningless, an average without normalization over many datasets is often dominated by the result on a few datasets.

2.1.8 Automatic Learning applications

Automatic Learning is the engine of the Data Mining field. Data mining approach has a major advantage from the point of view of its applicability: almost all the domains of human activity may benefit from it, both the ones where a lot of data is already available and the ones where the data have to be simulated in order to extract some more profitable knowledge concerning the field. Data mining helps one system, company or process to be more competitive, cheaper, (function) better, more client oriented, or more secure. We mention further some particular broad domains of interest in present data mining applications [OW99].

Market basket analysis refers to the process of examining point-of-sale data to identify affinities

between products and services purchased by a customer. Data mining must deal in these applications with large volumes of transactional and spread data and must be performed in a time interval that will allow an organization to respond to market opportunity before competition does. Important buying patterns, types of consumers exhibiting such patterns, customer characteristics that may be correlated to the consumers choices are all automatically identified by automatic learning techniques.

Customer segmentation is the process of analyzing data about customers or general consumers to identify characteristics and behaviors that can be exploited in the market place. One tries to reduce the customer attrition phenomenon, i.e. the loss of customers (one searches for customers that exhibit characteristics typical of someone who is likely to leave for a competing company), or to attract other customers, to identify the risk associated with insurance, etc.

Fraud detection. Data mining applications have demonstrated their benefits in the areas where many actions (transactions) are undertaken, making the respective system vulnerable to fraud: credit card services, telecommunications, computer systems ...

Detection of patterns in text, image, on the world wide web are broadly extensive areas of data mining applications due to the impressive amount of information available: finding associations amongst the keywords labeling items in a collection of textual documents, recognizing actions in video image sequences, helping users locate desired information in the web, protecting their e-mail from spam phenomenon, etc.

Medical diagnosis through means of data mining are intended to be helpful tools that can improve the physicians performance and make the diagnosis process more objective and more reliable. From the descriptions of the patients treated in the past for which the final diagnosis were verified, diagnosis rules may be automatically derived, although the technology is not widely accepted in medical practice, encountering a resistance of the physicians to new diagnostic technology (not so much because of technophobia, but because of "black boxes").

Quality control and industrial process reformation is a domain of interest since any industrial system or production process may be improved by means of data mining analysis. Different types of faults and spoilages may be reduced by making use of the knowledge extracted by automatic learning techniques, thus the process quality being improved.

Automatic Learning in power systems

Electrical power systems are essentially large, complex nonlinear systems that have grown significantly in size and importance during the last 50 years. In a power system there are some aspects which make automatic learning very welcomed [OGW99]: large scale character of power systems (thousands of state variables), temporal (from milliseconds to minutes, hours, weeks, years) and statistical nature of data, existence of a discrete (e.g. events such as topology changes or protection arming) and continuous (analog state variables) information mixture, on-line operation time restrictions for fast decision making, existence of uncertainty (noise, outliers, missing/partial information), existence of human factors, physical and socioeconomic influences difficult to take in account, continuous change/evolution of the system.

Starting with the late 1970s, many pioneering papers were proposing solutions to power systems that integrated Artificial Intelligence techniques. In the last decade, the maturity of automatic learning techniques per se has increased also, thus making more interesting and trusty their application in power systems, besides the existence of analytical tools.

Ref. [Weh00] make a good review of the huge possibilities of automatic learning application to power systems. They include: dynamic security assessment [Weh94, WJ97], controller and protection design, modeling, estimation, identification, forecasting [PK99], planning, and monitoring [McG01, MI99].

***

As we discussed in this first part of the chapter 2 some notions of interest for us concerning the automatic learning framework, let us introduce you the second frame that governs the fuzzy decision tree approaches, namely the fuzzy logic theory framework.

2.2 Fuzzy sets and fuzzy logic

Fuzzy theory is a mathematical theory aimed to deal with *vagueness* and to capture *imprecision* by providing a *quantitative* description, in a way similar to the probability theory.

The knowledge we have at disposal about a given situation is generally imperfect, either because we have a doubt concerning its validity, and we speak about **uncertainty** in this case, or because we have a difficulty of clearly expressing it, and we speak in this case about **vagueness** or **imprecision**. Thus the real world appears as sometimes imprecise and uncertain. To study or manage a (complex) real system, we need therefore to be able to deal with: vague data ("high pressure", "small house", "young men", "much older"), imprecise data ("about 20 meters large", "about between 20 and 25 years old"), data affected by errors ("200 inches with 5% possibility of being wrong"), ill-defined data ("a lot of pain"), data whose validity is not absolute ("in 85% of cases"), affected by an uncertain fact data ("very probable"). The human being has a natural feeling of cooping with all these types of data. Antagonistically, computers "reasoning" is based only on rigid conventional mathematics and conventional quantitative techniques of systems' analysis. Concerning *uncertainty*, it was already approached by the mediation of the *probability theory* in the seventeenth century by Pascal and Fermat. Problems posed by *imprecision or vagueness* were taken in consideration by the fuzzy sets theory. Fuzzy logic departs from the tradition of precision in scientific analysis, and offers a larger ability to deal with imprecision than conventional mathematics. The father of fuzzy sets theory is considered L. A. Zadeh, professor at University of Berkeley, California whose article "Fuzzy sets" [Zad65] put the bases of this theory in 1965.

Another facet of the real world and human thinking is concerning the ability to summarize information. In some situations, an approximate characterization of a collection of data is sufficient because a high degree of precision in the execution of a task for example is useless. The human brain has a *tolerance for imprecision* which allows him to take decisions based on "task-relevant" information. "The closer one looks at a real-world problem, the fuzzier becomes its solution" is a corollary principle to the *principle of incompatibility* enunciated by Zadeh, which states that:

"As the complexity of a system increases, our ability to make precise and yet significant statements about its behavior diminishes until a threshold is reached beyond which *precision* and *significance* become almost mutually exclusive characteristics."

It is in this sense that precise quantitative analyses of the behavior of humanistic systems are not likely to have much relevance to certain real-world problems and that fuzzy logic brings in useful tools for dealing with this tolerance to imprecision character that one would like to "introduce" in a computer based activity, being a control system or an automatic learning tool.

Fuzzy logic aims at providing a model for approximate reasoning rather than precise reasoning. The *approximate reasoning* is the process by which a possibly imprecise conclusion is deduced from a collection of imprecise premises. Fuzzy logic is an extension of the Boolean logic. Fuzzy logic uses graded statements rather than strictly true or false ones. Statements can be simultaneously true and false and this, to a measurable extent.

Fuzzy theory holds that all things are matters of degree [Kos92]. Mathematically, fuzziness means multivaluedness or multivalence. Three-valued fuzziness corresponds to truth, falsity, and

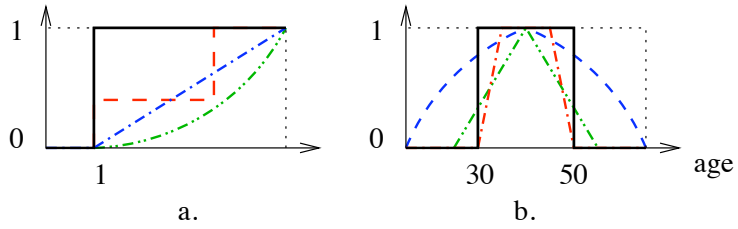


Figure 2.5: Example of fuzzy sets defined by fuzzy membership functions

indetermination, or to presence, absence and ambiguity.

2.2.1 Fuzzy sets

A *fuzzy set* is a "class" with a continuum of grades of membership. There is no $\{0,1\}$ Boolean sharp-defined class membership, but rather an infinity of possible grades of membership from 0 to 1.

Let U denote the universe of all possible points (objects) for a given problem. A fuzzy set $A \subset U$ is characterized by a *membership function*

$$\mu_A : U \rightarrow [0,1]$$

which associates with each point x of U a real number $\mu_A(x)$ in the interval $[0,1]$ representing the "grade of membership" of x in A. Traditionally, the grade of membership **1** is assigned to the objects that completely belong to A, conversely to objects that do not belong at all to A having assigned a membership value **0**. The more a point belongs to A the closer to **1** is its membership grade. The membership degree reflects an '*ordering* of the objects in the universe, and when this ordering exists, it is more important than the membership values themselves.

Examples of fuzzy sets:

1. The fuzzy set of numbers much greater than 1, with a membership degree that reflects in a certain given quantifying manner how far each number in our set is from 1. This membership function could take the shape of a sigmoidal curve, or a piecewise-linear trapezoidal or a triangular curve, or any other hand-defined function. The corresponding crisp Boolean set would be here the set of numbers greater than 1. A few membership function shapes are exemplified in figure 2.5a where in bold is represented the crisp one.

2. The fuzzy set of middle age members of a club, where 30 and 50 are taken as reference age values (instead of crisp set of members between 30 and 50 years old). Membership functions are exemplified in figure 2.5b.

As we may observe from these examples, the biggest difference between crisp and fuzzy sets is that the crisp sets always have *unique* membership functions, whereas every fuzzy set has an *infinite* number of membership functions that may represent it. Thus uniqueness is spent with a concomitant gain in flexibility.

2.2.2 Fuzziness is not probability

The grade of membership is not a probability. It is a measure of the compatibility of an object with the concept represented by a fuzzy set. For example 0.2 is the compatibility of Larry to the definition of the fuzzy set of middle age members of a club, 0.2 is not the probability that Larry is a middle age member of the club. Fuzzy memberships represent similarities of objects to imprecisely defined properties, whereas probabilities convey information about relative frequencies. Probability addresses the question *whether* an event will occur or has occurred, while fuzziness

quantifies *to what extent* it does. Probabilities deal with *chances*, fuzzy logic with *truth values*. Zadeh pointed out that "Intuitively, fuzziness degree relates to our perception of the degree of feasibility or ease of attainment whereas probability is associated with a degree of likelihood, belief, frequency or proportion".

An example in [Bez93] points out the difference between fuzzy degree and probability. "Suppose you had been in the desert for a week without drink and you come upon two bottles, A and B, A with the membership degree of being potable of 0.91, B with the probability of being potable of 0.91. Confronted with this pair of bottles and given that you must drink from the one that you choose, which would you choose to drink from first? [...] We may immediately see that while A could contain, say, swamp water, it would not contain liquids such as a hydrochloric acid. That is, a *membership* of 0.91 means that the contents of A are fairly similar to perfectly potable liquids (pure water). On the other hand, the *probability* that B is potable = 0.91 means that over a long run of experiments, the contents of B are expected to be potable about 91% of the trials; and the other 9%? In these cases the contents will be unsavory (indeed, possibly deadly) - about one chance in ten. Thus most subjects will opt for a chance to drink swamp water, and will choose bottle A. [...] Now suppose that we examine the contents of A and B, and discover that A contains beer, while B contains hydrochloric acid. *After* observation then, the membership value for A will be unchanged, whilst the probability value for B clearly drops from 0.91 to 0.0."

In other words, a probabilistic description reflects either the observer (partial) knowledge of the process parameters or a very random nature of the process, on the contrary to a membership degree, that states an intrinsic property of the process parameters, irrespective of the observer knowledge about the process.

2.2.3 Operators and operations in fuzzy sets

In the frame of fuzzy sets theory [BM93, Kau73, DP80], sets, numbers, operations, operators, distances, constrains, equations, matrices, relations, functions, events, models, systems, languages, grammars, algorithms, instructions, control are all re-defined in fuzzy version.

Set operations

Suppose two fuzzy sets A and B with their membership functions given by μ_A and μ_B. We present in what follows several definitions involving fuzzy sets in the way Zadeh defined them in [Zad65], which are extensions of the corresponding definitions for ordinary sets.

Empty set. A fuzzy set A is empty if and only if its membership function is identically zero on U, thus

$$\forall x \in U, \mu_A(x) = 0. \tag{2.42}$$

Equality. Two fuzzy sets A and B are equal, written as $A = B$ if and only if

$$\forall x \in U, \mu_A(x) = \mu_B(x). \tag{2.43}$$

Complement. The complement of fuzzy set A is noted $\bar{A}$ and is defined by

$$\mu_{\bar{A}(x)} = 1 - \mu_A(x). \tag{2.44}$$

Inclusion. A is contained in B, or A is a subset of B, or A is smaller than or equal to B, thus $A \subset B$ if and only if

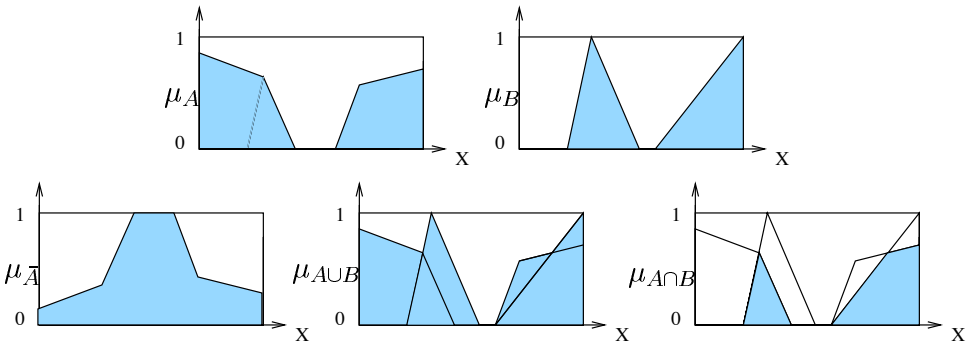


Figure 2.6: Extended Venn Diagram of the complement, union and intersection of fuzzy sets, adapted from [DP80]

$$\forall x \in U, \mu_A(x) \leq \mu_B(x). \tag{2.45}$$

Union. The union of two fuzzy sets A and B, written as $A \cup B$ is defined by

$$\forall x \in U, \mu_{A\cup B}(x) = max(\mu_A(x), \mu_B(x)). \tag{2.46}$$

A more intuitive appealing definition of the union is the following: the union of two fuzzy sets is the smallest fuzzy set containing both fuzzy sets.

Intersection. The intersection of two fuzzy sets A and B, written as $A \cap B$ is defined by

$$\forall x \in U, \mu_{A\cap B}(x) = min(\mu_A(x), \mu_B(x)). \tag{2.47}$$

As in the case of union: the intersection of two fuzzy sets is the largest fuzzy set which is contained in both fuzzy sets.

Cardinality. The cardinality of a fuzzy set A, noted $|A|$ is defined by

$$|A| = \sum_{x\in U} \mu_A(x) \tag{2.48}$$

in the case of a countable fuzzy set A and

$$|A| = \int_{x\in U} \mu_A(x)dx \tag{2.49}$$

in the case of an uncountable fuzzy set A.

Figure 2.6 exemplifies the three most useful fuzzy sets operations: the complement of a fuzzy set, the union and the intersection of two fuzzy sets. For example, if in our figure, A is the fuzzy set of "nice cars" and B is the fuzzy set of "expensive cars", the union shows the fuzzy set of "either expensive or nice cars or both", and the intersection shows the fuzzy set of "expensive (and) nice cars". Union, intersection and complement follow the same properties as in Boolean mathematics (commutativity, associativity, distributivity, idempotency, De Morgan's laws, etc), excepting the excluded-middle law which is not longer true: $A \cap \bar{A} \neq \emptyset$, $A \cup \bar{A} \neq U$, since sets A and $\bar{A}$ overlap. This overlap region is always limited to 0.5. For example, if U is the set of colored objects, and A is the subset of red ones, μ_A measures the degree of redness. A pink object has a membership value close to 0.5 and thus belongs nearly equally to both A and $\bar{A}$.

Table 2.2: Principal dual t-norms and t-conorms

$\top(\mu_A, \mu_B)$ t-norms		$\perp(\mu_A, \mu_B)$ t-conorms	
intersection: $\wedge$, AND		union: $\vee$, OR	
$\mu_{A\cap B} = min(\mu_A, \mu_B)$	Zadeh	$\mu_{A\cup B} = max(\mu_A, \mu_B)$	Zadeh
$\mu_{A\cdot B} = \mu_A\mu_B$	product	$\mu_{A\hat{+}B} = \mu_A + \mu_B - \mu_A\mu_B$	probabilistic sum
$max(0, \mu_A + \mu_B - 1)$	bold $\cap$ Lukasiewicz	$min(1, \mu_A + \mu_B)$	bounded sum
$f(x) = \begin{cases} \mu_A & \text{if } \mu_B = 1 \\ \mu_B & \text{if } \mu_A = 1 \\ 0 & \text{otherwise} \end{cases}$	Weber	$f(x) = \begin{cases} \mu_A & \text{if } \mu_B = 0 \\ \mu_B & \text{if } \mu_A = 0 \\ 1 & \text{otherwise} \end{cases}$	Weber

Logical operators

Apart minimum and maximum operators, there exists multiple intersection and union operators. The functions that fulfill a minimum of requirements for an intersection operator are called **t-norms**. The functions that fulfill a minimum of requirements for a union operator are called **t-conorms**. Here are the definitions of these operators.

T-norm. A function $\top : [0,1] \times [0,1] \to [0,1]$ is called *t-norm* if it fulfills the conditions:

$$\begin{array}{ll} \top(a,1) = a & \text{(unit element 1)} \\ \top(a,c) \leq \top(b,c) \quad \text{if} \quad a \leq b & \text{(monotonocity)} \\ \top(a,b) = \top(b,a) & \text{(commutativity)} \\ \top(a,\top(b,c)) = \top(\top(a,b),c) & \text{(associativity).} \end{array} \tag{2.50}$$

T-conorm. A function $\perp : [0,1] \times [0,1] \to [0,1]$ is called *t-conorm* if and only if

$$\begin{array}{ll} \perp(a,0) = a & \text{(unit element 0)} \\ \perp(a,c) \leq \perp(b,c) \quad \text{if} \quad a \leq b & \text{(monotonocity)} \\ \perp(a,b) = \perp(b,a) & \text{(commutativity)} \\ \perp(a,\perp(b,c)) = \perp(\perp(a,b),c) & \text{(associativity).} \end{array} \tag{2.51}$$

T-norm and t-conorm are dual concepts. We can derive a t-conorm from a t-norm and inversely through a negation operation. Table 2.2 displays some of the t-norms and t-conorms operators defined in the literature [BM93]. The t-norms are employed for defining conjunctions (AND operations) in approximate reasoning, while the t-conorms serve the same role for disjunctions (OR operations). In the majority of fuzzy systems the *min* and *max* operators are used.

2.2.4 Fuzzy systems

Fuzzy controller

A *fuzzy logic controller* [Lee90] consists of a collection of control laws whose inputs and outputs are both fuzzy values and an inference mechanism so as to determine the control action for a given process state. A fuzzy logic controller processes information as follows (see figure 2.7). First, the **fuzzification** process converts a crisp input value (coming from a sensor for example) into a fuzzy value. Then the **fuzzy inference** is responsible of drawing conclusions from a given base of *fuzzy rules* given the fuzzified inputs. Finally, the **defuzzification** process converts the fuzzy control actions into a crisp control action output.

The *fuzzy knowledge base* or simply, the *rule base* contains a set of fuzzy (control) rules of type

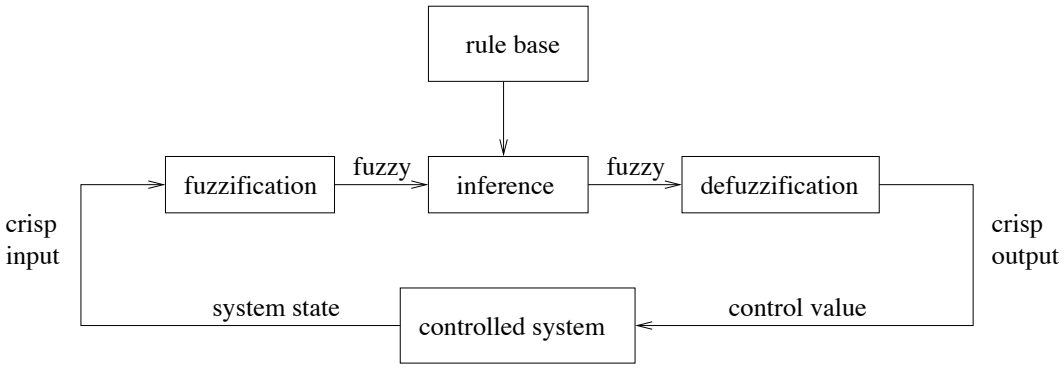


Figure 2.7: Architecture of a fuzzy logic controller

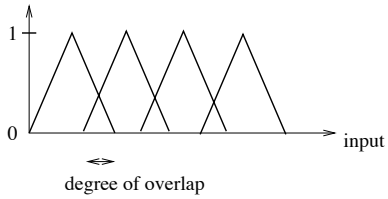


Figure 2.8: Equally overlapped fuzzy sets

$$Rule\ j: \quad \textbf{IF}\ X\ is\ A\ \textbf{THEN}\ Y\ is\ B \tag{2.52}$$

where A and B are fuzzy sets, $j = 1 \ldots R$, R is the number of rules, X is the vector of inputs, Y is the output. "*X is A*" is called the *antecedent* of the rule, "*Y is B*" is called the *consequent* of the rule.

Fuzzification

At this step of the fuzzy process, fuzzy sets are defined based on the crisp inputs. Thus mappings between the observed inputs and the membership degrees to fuzzy sets are established, i.e. from $U \subset \Re^m$ to the unit hypercube $[0,1]^{m*R}$, where m is the number of inputs. Usually, the shape of the membership function is a priori chosen. The triangular membership function is the most frequently used function and the most practical, but other shapes are also used. One is the trapezoid which contains more information than the triangular shape. A fuzzy set can also be represented by a quadratic equation to produce a continuous curve. Additional shapes which may appear are: the S-curve (sigmoidal), the pi-curve, bell-shaped (Gaussian), and linear. Usually, fuzzy sets are described with convex functions. Often, the fuzzy sets are series of fuzzy sets of identical shape (see figure 2.8), overlapped at a certain degree, called *degree of overlap*, degree that needs to be a priori defined also, or there are procedures that select the appropriate degree for the problem at hand. Also the number of membership functions has to be fixed a priori. If the fuzzy sets cover the whole universe of discourse we speak about completeness. There are cases where the membership functions are integrally setup by hand and bestowed in databases, or cases where the membership functions are optimized (tuned) in order to fit some available data (called data-driven membership functions).

Fuzzy inference

The inference process is a decision making logic which determines fuzzy outputs corresponding to fuzzified inputs, with respect to the fuzzy rules in the rule base. Fuzzy systems "reason" with parallel associative inference. When asked a question or given an input, a fuzzy system fires each rule in parallel, but to different degree, to infer a conclusion or output.

The inference process of a fuzzy system may be accomplished in three substeps as follows:

1. Determination of the truth degree for each fuzzy rule of the satisfaction of its antecedent. Since it is a conjunction operation, t-norms are used.

2. Determination of the degree of the satisfaction of the consequent for each fuzzy rule, by propagation of the satisfaction of the antecedent to the consequent. This step is done by means of a *fuzzy implication*, like for example a *min* or *product* operation (ref. [BM93] gives a non-exhaustive list of principal fuzzy implications).

3. Conflict resolution from multiple consequents, thus aggregation of all the rules consequents. Since it is a disjunction operation, t-conorms are used.

Let us consider a fuzzy rule base with two fuzzy rules as a particular case of the equation (2.52). Thus we have the following multiple fuzzy reasoning form:

$$\begin{array}{lll} R_1: & \textbf{IF } X_1 \textit{ is } A_{11} \textbf{ AND } X_2 \textit{ is } A_{12} \textbf{ THEN } Y \textit{ is } B_1 & \\ R_2: & \textbf{IF } X_1 \textit{ is } A_{21} \textbf{ AND } X_2 \textit{ is } A_{22} \textbf{ THEN } Y \textit{ is } B_2 & \\ Fact: & x_{10} \text{ and } x_{20} & \\ \hline Consequence: & y \text{ is } B' & \end{array} \tag{2.53}$$

where A_{11}, A_{12}, A_{21}, A_{22} are fuzzy sets defined on the input X, B_1 and B_2 are fuzzy sets defined on the output Y and $x_{10} \in X_1$ and $x_{20} \in X_2$ are the crisp input values for which we have to infer a correspondent output fuzzy set B'.

One of the most used fuzzy inference methods in control systems is the *min-max* method. We express the steps of this type of inference process on the considered example:

1. Truth degree of each fuzzy rule i by means of a *min* ($\wedge$) operation:

$$\alpha_i = \mu_{A_{i1}}(x_{10}) \wedge \mu_{A_{i2}}(x_{20}). \tag{2.54}$$

2. Degree of the satisfaction of the consequent for each fuzzy rule i by means of a *min* operation:

$$\mu_{B'_i}(y) = \alpha_i \wedge \mu_{B_i}(y). \tag{2.55}$$

3. Aggregation of the rules consequents by means of a *max* ($\vee$) operation:

$$\mu_{B'}(y) = \mu_{B'_1}(y) \vee \mu_{B'_2}(y). \tag{2.56}$$

This example if illustrated also graphically in figure 2.9, where the induced output fuzzy set $\mu_{B'}$ is also defuzzified so as to obtain a crisp output value y_0.

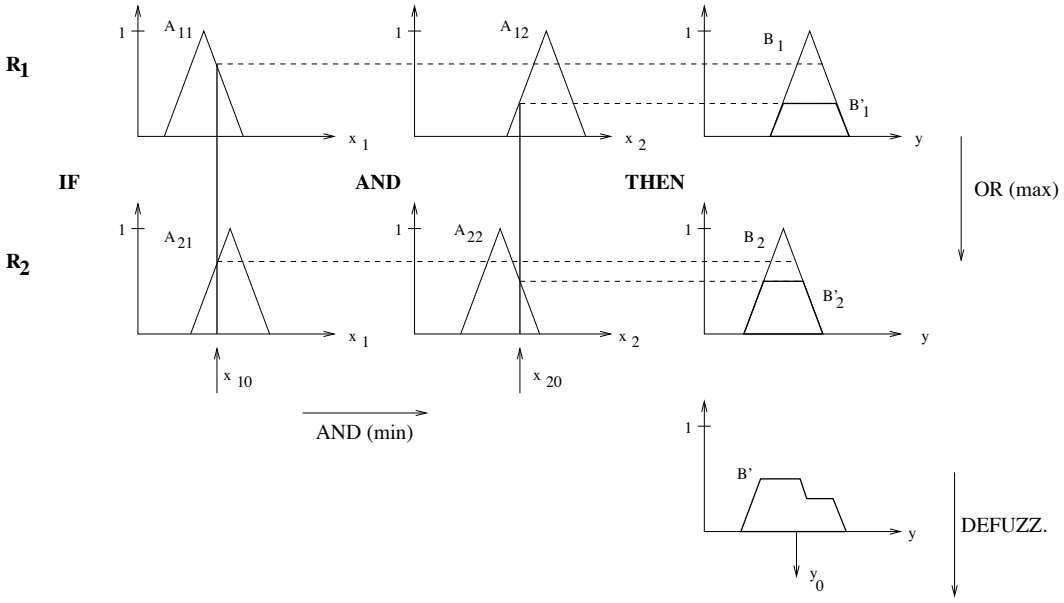


Figure 2.9: An example of evaluation of two fuzzy rules (min-max inference plus defuzzification)

Defuzzification

Using the rules of a fuzzy inference, the evaluation of a new input produces one output fuzzy set. Defuzzification involves finding an (abscissa) value that best represents the information contained in this fuzzy set. In any defuzzification algorithm there is a tradeoff between the need to find a single point result and the loss of information such a process entails. There are multiple defuzzification strategies in order to transform a fuzzy set into a crisp value. Here are some of them:

- *Max-criterion* that finds the domain point which presents the maximum truth.

- *Mean-of-maximum (MOM)* method where the output is the average of all abscissa values where the fuzzy set assumes a maximum membership degree.

- *Center-of-gravity (COG)* or *center-of-aria (COA)* method where all possible output values are weighted by the degree of membership of the fuzzy set and added up, then the result is divided by the sum of all the membership degrees. It is in fact an weighted mean of the fuzzy region, or in other words, a point representing the center of gravity of the fuzzy set.

- *Average of the nonzero region* that takes the average of the support set of the output fuzzy region (values for which the membership degree is not null).

- *Center-of-maximums* technique that finds the highest plateau and then the next highest plateau. The midpoint between the centers of these plateaus is selected.

2.2.5 Hybrid Fuzzy Automatic Learning techniques

Neurofuzzy systems

For some problems, a fuzzy rule base can be clearly obtained by using human knowledge. In this case, a key problem is how to tune membership functions to fit knowledge efficiently. Since this tuning task can be viewed as an optimization problem, neural networks offer a possibility to solve this problem. The term **neurofuzzy** stems from the use of heuristic learning strategies derived from the domain of neural network theory to support the development of a fuzzy system. A neurofuzzy system takes over the capabilities of neural nets to learn and generalize, as well as their massively

parallel computing character; it inherits also their "universal approximation capabilities". On the other hand, it may take advantage of fuzzy logic to represent, manipulate and utilize information that possesses non statistical uncertainty. More importantly, neurofuzzy systems aim at expressing a function in terms of interpretable linguistic rules as an alternative to pure neural network "black boxes", in which the function learned can only be observed through the input/output relationship. Neurofuzzy systems have been studied in detail by [NKK97].

We make the next classification of systems merging fuzzy and neural networks techniques:

(i) neural networks that incorporate the imprecision and linguistic data handling abilities of fuzzy systems; in such approaches a conventional neural network architecture is fuzzified in order to be able to process fuzzy inputs; fuzzy techniques are used to create or enhance neural networks; we call these approaches *fuzzy (arithmetic based) neural networks*.

(ii) approaches where neural networks are used to provide inputs or to change outputs of a fuzzy system and where parameters of this fuzzy system are not modified by a learning process; we prefer to call them *neural/fuzzy combinations*.

(iii) fuzzy inference systems that incorporate the learning and generalization abilities of neural networks; neural networks are used as tools in fuzzy systems; a fuzzy system is derived from data or enhanced by learning from examples; we consider these *neurofuzzy* systems.

A more subtle classification of neurofuzzy systems is performed in [NM00], where 11 possible types are identified.

Fuzzy decision trees

For some other problems, no fuzzy rule base is available and a decision tree approach may be an effective way to determine it, extracting the fuzzy rules directly from quantitative data available about the problem. Thus the **fuzzy decision tree** approach is born, as an evolution of the inductive learning strategies toward the development of fuzzy decision rules. Next chapters are treating this type of approach. Ref. [OW00] achieves a parallel between neurofuzzy and fuzzy decision tree approaches. Chapter 5 makes a short review of the major related literature to fuzzy decision trees and points out different aspects of existent fuzzy decision tree methods.

Chapter 3

Proposed soft decision tree induction

This chapter gives first an intuitive view of our fuzzy (or soft) decision tree approach through means of an illustrative example: a parallel is done between the fuzzy decision tree approach and the regression tree approach, so as to allow the reader to perceive at an intuitive level how the model works in practice. Then the proposed fuzzy decision tree induction method is presented[1]*. Growing, pruning, refitting and backfitting steps are explained herein in detail.*

3.1 First glance of the fuzzy decision tree approach

We present intuitively the formal representation of a fuzzy decision tree by explaining first the regression tree (RT) type of induction. Regression trees and fuzzy decision trees are extensions of the decision tree induction technique, predicting a numerical output, rather than a discrete class[2]. Both types of 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. Figure 3.1 shows a crisp regression tree (left part (a)) and a fuzzy decision tree (right part (b)). Both were built for an illustrative electric power system security problem (the so-called OMIB database described in section B.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) [3] of the system state (see right part of figure 3.2). The trees predict the membership degree of instances to this fuzzy class.

In a crisp decision (see left part of figure 3.2), the system could be considered "insecure" if the CCT is smaller than 155 msec, "secure" otherwise (classification problem). In a fuzzy decision (see right part of figure 3.2), there is a transition region of 110 msec (as we defined it) in which the system is neither "secure" nor "insecure" (regression problem).

Next, significant differences between fuzzy and crisp trees are pointed out on the OMIB example.

[1] The FDT method proposed here has been published in [OW03]

[2] Appendix A gives a parallel between CART regression trees and our fuzzy decision tree method, summarized in formulas.

[3] This CCT parameter is a security margin used in transient stability studies.

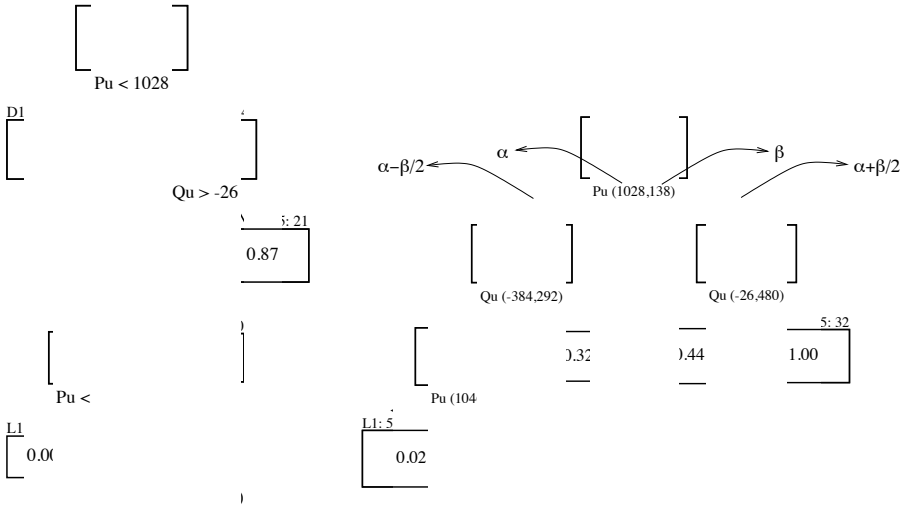


Figure 3.1: Regression tree versus fuzzy decision tree (OMIB data)

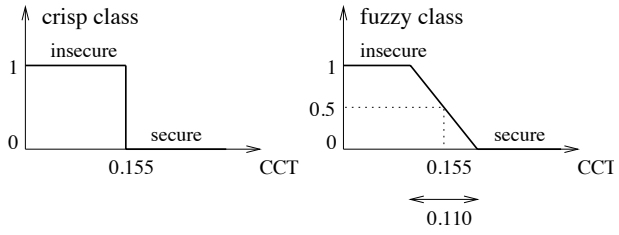


Figure 3.2: Example of a crisp class and a fuzzy class for OMIB data

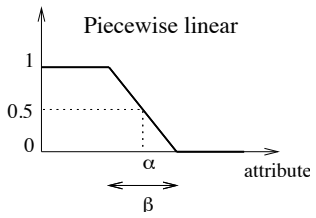


Figure 3.3: Example of discriminator function: piecewise linear

3.1.1 Crisp regression tree

The CART regression tree in figure 3.1(a) has four test nodes and five terminal nodes. Each node box is stamped with the local estimation of the output[4]. 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 figure 3.2): since degree 0.87 corresponds to a CCT value smaller than 155msec, 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 $Pu \geq 1028MW$ and $Qu \leq -26MVar$ Then $degree$ 0.87.

3.1.2 Fuzzy decision tree

The fuzzy decision tree in figure 3.1(b) has also four test nodes and five terminal nodes. Each node is also marked with its local estimation of the output[5]. 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 test node. A widely used shape of discriminator function is the piecewise linear one (see figure 3.3). 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 partition) 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.

[4]The local estimation of the output in a node of a CART regression tree is given by the average of output values of all the objects in the local learning set.

[5]The precise meaning of these local estimations (called also labels) and how they are computed are explained latter in section 3.4.2.

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, 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 averaged 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 figure 3.1(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 "insecure" 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 155msec. The rules fired by our instance correspond to the two identified paths:

If $Pu \geq 959MW$ and $Qu \geq -266MVar$ Then $degree$ 0.44

If $Pu \geq 959MW$ and $Qu \leq 214MVar$ Then $degree$ 1.00.

It is worth noting in figure 3.1 that both regression and fuzzy decision trees present at their root node the same chosen attribute Pu and also identical cut-points (1028MW). This is not a coincidence and the procedure we adopted for fuzzy partitioning will explain why this happens when the learning set to be split in a node is identical for the two methods.

3.2 Hypothesis space of a fuzzy decision tree

A fuzzy decision tree is a method able to partition the input space into a set of rectangles and then approximate the output in each rectangle by a smooth curve, instead of a constant or a class like in the case of crisp tree-based methods (see the right part of figure 3.4, corresponding to the estimation of the output approximated by the fuzzy decision tree in the left part of the figure; compare figure 3.4 with figure 2.3 of section 2.1.3).

A fuzzy tree is an approximation structure to compute the degree of membership of objects to a particular class (or concept) or to compute a numerical output of objects, as a function of the attribute values of these objects. The goal is to recursively split the input space into (overlapping) subregions of objects which have the same membership degree to the target class (in the case of classification problems) or the same output value (in the case of regression problems).

As in any tree-based approach, the hypothesis space of a fuzzy decision tree model is a family of structures. A fuzzy decision tree structure is determined by the graph of the tree and by the attributes attached to the its test nodes. The discretization thresholds (α) and width (β) values of all these attributes (thresholds α_1 and α_2 and widths β_1 and β_2 in figure 3.4) together with the labels of all the terminal nodes represent the parameters of the tree-based model. There is a search over both structure and parameter spaces so as to learn a model from experience.

3.3 Building a fuzzy decision tree

Figure 3.5 presents an overview of the complete procedure for building a fuzzy 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

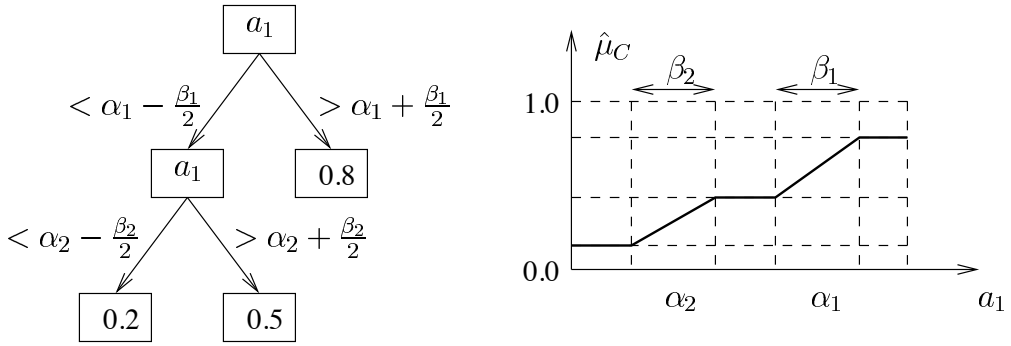


Figure 3.4: Example of a fuzzy decision tree

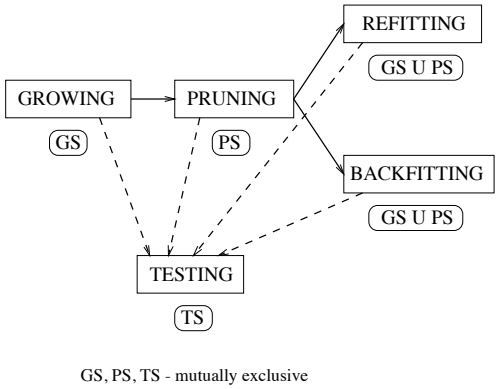


Figure 3.5: The procedure of building a fuzzy decision tree

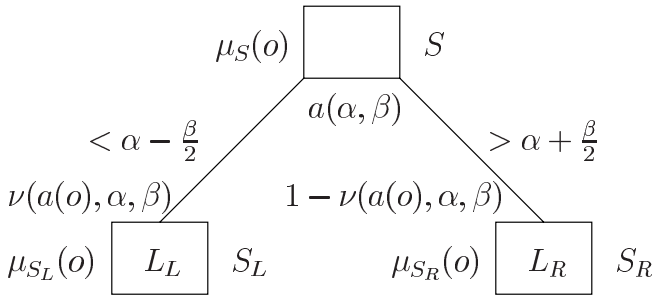


Figure 3.6: Fuzzy partitioning of a node in a fuzzy decision tree

the grown tree is *pruned* in a bottom-up fashion to remove its irrelevant parts. At this stage, the hold-out 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 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.

3.4 Growing

By analogy with the CART method of decision and regression tree building [BFOS84], the procedure for growing a fuzzy 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.

3.4.1 Fuzzy tree semantics

Let U denote the universe of all possible objects for a given problem. Let us remind that 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.

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].

Figure 3.6 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 $\nu(a(o),\alpha,\beta,\gamma\ldots) \to [0\ldots1]$ 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 figure 3.3); these are defined[6] 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:

$$\mu_{S_L}(o) = \mu_S(o)\nu(a(o), \alpha, \beta). \tag{3.1}$$

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 $\nu(a(o), \alpha, \beta) = 0$ for all objects for which $a(o) > \alpha + \frac{\beta}{2}$, only those objects of S such that $a(o) \leq \alpha + \frac{\beta}{2}$ go towards the left successor S_L. Similarly, the objects directed towards the right successor S_R are weighted by the complement of the discriminator function value:

$$\mu_{S_R}(o) = \mu_S(o)(1 - \nu(a(o), \alpha, \beta)), \tag{3.2}$$

and correspond to the condition $a(o) \geq \alpha - \frac{\beta}{2}$.

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[7] 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, $\mu_{S_{L_j}}(o)$:

$$\hat{\mu}_C(o) = \frac{\sum_{j \in leaves} \mu_{S_{L_j}}(o) L_j}{\sum_{j \in leaves} \mu_{S_{L_j}}(o)}, \tag{3.3}$$

where $leaves$ denotes the set of indexes 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 manuscript we will make this assumption so as to simplify our notations.

Analogy with a fuzzy system

A fuzzy decision tree may be seen as a fuzzy system, like the one represented in section 2.2.4, figure 2.7. The attributes chosen by the tree constitute the input of the fuzzy system, the target class or the numerical output represent the output of the system. Lets consider a new object is dropped in the tree. The fuzzification step consists in fuzzy partitioning the inputs corresponding to this new object in conformity with the parameters of the discriminator function in every node of the tree. The fuzzy inference step draws a conclusion based on these fuzzified inputs as follows. The truth degree for each fuzzy rule is determined by a t-norm *product* operator (see equations (3.1) and (3.2)). The degree of the satisfaction of the consequent for each fuzzy rule is determined by a *product* fuzzy implication (see the numerator of the equation (3.3)). The aggregation of all the rules consequents is done by a *weighted-sum* operation (see equation (3.3)).

[6] Appendix A gives the formula for a piecewise discriminator function.

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

3.4.2 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, fuzzy decision tree growing implies the *automatic* generation of a 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 fuzzy decision tree, find attribute $a(\cdot)$, threshold α and width β (parameters defining the discriminator function ν) together with successors labels L_L and L_R, so as to minimize the squared error function

$$E_S = \sum_{o \in S} \mu_S(o)[\mu_C(o) - \hat{\mu'}_C(o)]^2 \tag{3.4}$$

where

$$\hat{\mu'}_C(o) = \nu(a(o), \alpha, \beta)L_L + (1 - \nu(a(o), \alpha, \beta))L_R, \tag{3.5}$$

error depending nonlinearly 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 [BW95b, Mar98b, Ram94, SL99] and as [BW95b] proves is the case of convex measures. The squared error of equation (3.4) called *score measure* or *discriminating measure* is in our case employed *both* for measuring the "power" of each attribute to split the input space in subsets of the same output, *and* for fuzzifing each attribute.

Strategy. The local search is decomposed as follows (next sections give 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 previous 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. (3.4) in the particular case of a FDT 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 [Ola98]. Figure 3.7 presents a typical error surface in terms of two of the four parameters (α and β), error expressed as

$$E'_S(\alpha, \beta) = \min_{L_L, L_R} E_S(\alpha, \beta, L_L, L_R). \tag{3.6}$$

The two successor labels, L_L and L_R, are optimized for each value of α and β. The graphic scans values of $\alpha \in (0.0, 1.0)$ and values of $\beta \in [0.0, min(2\alpha, 2(1-\alpha))]$ [8] with small enough step size and a piecewise linear discriminator.

One can notice from figure 3.7 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

[8] The superior limit of β here comes from the following reasoning: when $\alpha < 0.5$, β is maximum 2α (see figure 3.3) but no more than 1.0; and when $\alpha \geq 0.5$, β is maximum $2(1-\alpha)$ but no more than 1.0. Therefore, β cannot be more than $min(2\alpha, 2(1-\alpha))$.

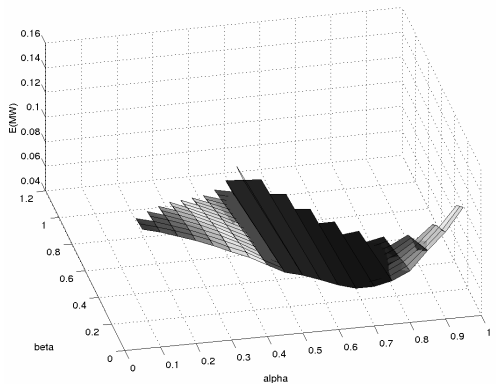


Figure 3.7: $E'_S(\alpha, \beta)$ error surface for the fuzzy class of OMIB database

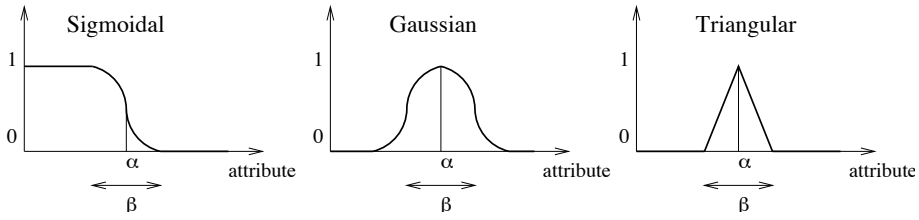


Figure 3.8: Example of other discriminator functions

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 nonlinear optimization technique were used for simultaneous search of all the parameters.

Reference [Ola98] shows that similar surfaces of the error function are found if the sigmoidal discriminator is used (see figure 3.8) instead of the piecewise linear one. Unfortunately, it is not the case of discriminators of Gaussian or triangular shape, in which case the surface presents multiple minima. This led us to not further consider non-monotonic discriminator functions. Note also that in the case of ordered attributes, monotonic discriminator templates are preferred since they yield trees which are easier to interpret.

3.4.3 Searching for the split location α in a FDT, when $\beta = 0$

As long as the split in a fuzzy 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 weighting by membership degrees. The objects have to be *a priori sorted*: for each attribute, the local learning sample is first sorted by increasing order of the attribute values.

Crisp splitting rule. Given a node S of a fuzzy decision tree (or equivalently, given its subset S of learning objects), the best crisp partition for S maximizes over all the possible partitions of all attributes the normalized squared error reduction

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

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 figure 3.6 are mutually exclusive, the piecewise linear discriminator $\nu(\cdot)$ of figure 3.3 becomes crisp, and the error function E_S of eq. (3.4) can be written based on eq. (3.5)

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

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

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

$$\begin{aligned} -2\textstyle\sum_{o\in S_L} \mu_S(o)[\mu_C(o) - L_L] &= 0 \\ -2\textstyle\sum_{o\in S_R} \mu_S(o)[\mu_C(o) - L_R] &= 0, \end{aligned}$$

and finally, the two successors labels are estimated as

$$L_L = \frac{\sum_{o\in S_L} \mu_S(o)\mu_C(o)}{\sum_{o\in S_L} \mu_S(o)}, \qquad L_R = \frac{\sum_{o\in S_R} \mu_S(o)\mu_C(o)}{\sum_{o\in S_R} \mu_S(o)}, \tag{3.7}$$

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:

$$E_A = \sum_{o\in A} \mu_S(o)\mu_C^2(o) - \frac{(\sum_{o\in A} \mu_S(o)\mu_C(o))^2}{\sum_{o\in A} \mu_S(o)} \tag{3.8}$$

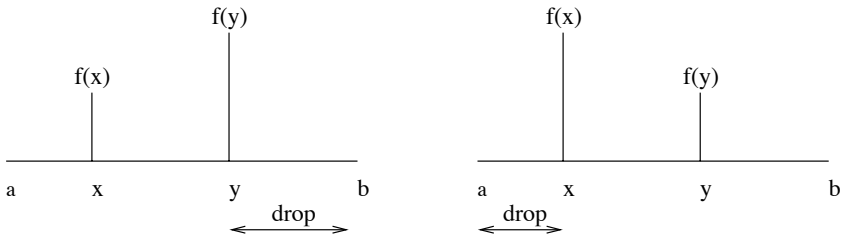
where A stands for S_L or S_R. All the sums $\sum_{o\in S_L}$ and $\sum_{o\in S_R}$ that intervene in the preceding formulas are updated once an object changes the part of the split, as it is explained next. We should note that the update formulas (3.8) may bring round-off errors [PTVF94].

Strategy. The optimal value of α is determined in the following way: objects are sorted by increasing values of the considered attribute, and let v_i and v_{i+1} be two successive different values found in this list of attribute values. A candidate value $\alpha = \frac{v_i+v_{i+1}}{2}$ is considered for each $i = 1, ..., N-1$, its score is computed (the normalized squared error reduction) and the α corresponding to the best score is the result obtained. Since the search is done by increasing values of i, at each step, only one or a small number of objects migrate from S_R to S_L and incremental formulas are used to update sums $\sum_{o\in S_L}$ and $\sum_{o\in S_R}$ (these objects that migrate are deleted from the right sum $\sum_{o\in S_R}$ and added to the left sum $\sum_{o\in S_L}$). This avoids computing the two sums from scratch every time α changes.

If in formulas (3.8) we consider that all the membership degrees $\mu_S(o)$ equal unity, we get the regression tree formulas. This explains why a regression tree and a non-backfitted fuzzy tree have in this case in their root nodes identical chosen attributes and α thresholds (see figure 3.1).

3.4.4 FIBONACCI search for β width

FIBONACCI search [Gre03] is a univariate optimization strategy that finds the minimum of a function on an interval only by function evaluations at certain points in the interval, without making use of any function derivatives. It finds the minimum of a function on an interval $[a, b]$, by evaluating points placed according to a FIBONACCI sequence F_N, numbers satisfying $F_{(n+2)} = F_{(n+1)} + F_n$, with initial conditions $F_0 = F_1 = 1$. If there are F_N points in the interval, only N evaluations of the function to be minimized are needed. One begins with the interval of uncertainty $[a, b]$ and its length is finally reduced to $(b-a)/F_N$. N is the number of evaluations needed to reduce the length of an interval of uncertainty to $1/F_N$.



$f(x) < f(x)$ | $f(x) \geq f(x)$

$b = y$ | $a = x$

$y = x$ | $x = y$

$x = a + \frac{1-g_n}{b-a}$ | $y = a + \frac{g_n}{b-a}$

Figure 3.9: Two possible cases in Fibonacci search

The ratio $g_n = F_{(n-1)}/F_n$ is the key to the placements of the points in which the function to be minimized is evaluated. For very large N, this placement ratio approaches the golden mean (approximately 0.618) and the FIBONACCI method approaches the Golden Section search. With N evaluations, FIBONACCI search can guarantee finding the minimum on a set of F_N points. In this respect, FIBONACCI search reduces the maximum number of evaluations needed to find the minimum of a function on a given finite set of points with respect to other Line Search methods like Dichotomous search or Golden Section search [Gre03].

Intuitively, the algorithm brackets the minimum, by splitting the current interval $[a, b]$ corresponding to FIBONACCI numbers, instead of splitting it in the middle (like a bisection method does), or based on golden mean (like a Golden Section search does). A quadruple (a, x, y, b) characterizes at every step of the algorithm the state of the search, where $[a, b]$ interval is narrowed at each new step. This interval brackets the abscissa whose ordinate is the best minimum achieved so far. Figure 3.9 shows the two possible cases given the current state (a, x, y, b): a sub-interval is dropped according to which of the values $f(x)$ and $f(y)$ is higher, resulting in a new state (a, x, y, b). The algorithm starts with a given maximal number of evaluations N and stops when they are all accomplished. The final minimum is one of the values x or y.

Here is the algorithm for minimizing a function $f(\cdot)$, given the starting interval $[a, b]$ and the maximal number of evaluations N:

1 set $n = N$
2 let $x = a + \frac{1-g_n}{b-a}, y = a + \frac{g_n}{b-a}$
3 evaluate $f(x)$ and $f(y)$
4 while $(n > 1)$ **do**
  if $f(x) < f(y)$
  then
    set $b = y, n = n - 1, y = x, f(y) = f(x), x = a + \frac{1-g_n}{b-a}$
    compute $f(x)$
  else
    set $a = x, n = n - 1, x = y, f(x) = f(y), y = a + \frac{g_n}{b-a}$
    compute $f(y)$
5 set $x^* = \arg\min(f(x), f(y))$

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: figure 3.10 presents the function to be minimized of figure 3.7 $E_S = E_S(\beta)$ in terms of β once the optimal α location has been settled, in this case $\alpha = 0.65$, for OMIB database.

The experiments show that $N = 5$ is a good compromise computation time - accuracy (see section 4.2.6 for a little experiment to prove this). Note that this value corresponds to a reduction of an interval to 0.125 of its original length.

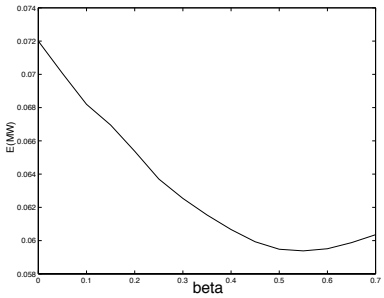


Figure 3.10: $E_S = E_S(\beta)$ for the optimal value of α ($\alpha = 0.65$). OMIB database

3.4.5 Optimizing the successor labels at fixed α and β

In order to minimize the error function of eq. (3.4), we settle

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

In conformity with eq. (3.5)

$$\begin{array}{rcl} -2\sum_{o\in S}\mu_S(o)\nu(a(o))\{\mu_C(o) - [\nu(a(o))L_L + (1-\nu(a(o)))L_R]\} & = & 0 \\ -2\sum_{o\in S}\mu_S(o)(1-\nu(a(o)))\{\mu_C(o) - [\nu(a(o))L_L + (1-\nu(a(o)))L_R]\} & = & 0. \end{array}$$

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

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

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 $\nu(a(o))$:

$$\begin{array}{ll} a(o) = \sum_{o\in S}\mu_S(o)\nu(a(o))^2 & b(o) = \sum_{o\in S}\mu_S(o)\nu(a(o))(1-\nu(a(o))) \\ c(o) = \sum_{o\in S}\mu_S(o)(1-\nu(a(o)))^2 & d(o) = -\sum_{o\in S}\mu_S(o)\mu_C(o)\nu(a(o)) \\ e(o) = -\sum_{o\in S}\mu_S(o)\mu_C(o)(1-\nu(a(o))). & \end{array}$$

We could have settled the labels directly by formulas of type (3.7) 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. [BW99] already observed that using weighted average leads to over-smoothing.

3.4.6 Computational complexity of tree growing

The complexity of FDT 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.

3.4.7 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 should 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.

3.4.8 Multiple classes

The fuzzy decision tree approach is suited for having a single output class C, the result for the complementary class being deduced as

$$\hat{\mu}_{\bar{C}}(o) = 1 - \hat{\mu}_C(o) \tag{3.9}$$

(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 $FDTs$ is built, each tree being dedicated to the recognition of a single class against the union of all the other classes [MBM97]. 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:

$$C^* = \arg\max_{C_i} \hat{\mu}_{C_i}(o). \tag{3.10}$$

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.

3.5 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. Pruning should be regarded as a preference bias over simpler models. The pruning algorithm which is an important part of our FDT method is further detailed below.

Objective. Given a complete FDT and a pruning sample of objects PS, find

$$\arg \min_{\text{subtrees of FDT}} MAE_{PS} \tag{3.11}$$

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

A subtree of an FDT 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 in the worst case 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 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 3.5.1).

- *Best subtree selection*. We use the "One-standard-error-rule" [BFOS84, Weh98] 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 $\min\{MAE\} + S.E.(MAE)$, where $S.E.(MAE)$ is the standard error of MAE (see below its definition) estimated from the PS. Notice that, if the PS size is very large this rule tends to select the tree of minimal MAE (since $S.E.(MAE) \to 0$ with pruning sample size). On the other hand, for small size of PS, this rule tends to slightly over-prune the tree.

3.5.1 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. (3.3).

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 to 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.

3.5.2 Best subtree selection

"One-standard-error-rule" chooses the smallest tree from the sequence of trees whose MAE^* is at most equal to the best MAE from the sequence plus its standard error:

$$MAE^* \leq MAE_{best} + S.E.(MAE_{best}) \tag{3.12}$$

where

$$S.E.(MAE) = \frac{1}{M}\sqrt{SE - \frac{AE^2}{M}} \tag{3.13}$$

(see eq. (2.39)) and M is the size of the pruning set, $SE = \sum_{o \in PS}(\mu_C(o) - \hat{\mu}_C(o))^2$ is the squared error and $AE = \sum_{o \in PS} |\mu_C(o) - \hat{\mu}_C(o)|$ is the absolute error.

3.5.3 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 $\mathcal{O}(K \cdot ||GS|| + \cdot K \cdot \log K)$, where K is the complete tree complexity and $||GS||$ denotes the growing 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, where $||PS||$ denotes the pruning set size. The tree selection is only proportional to the number of pruned trees obtained in the sequence which is upper bounded by K.

3.6 Parameter 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 nonlinear optimization technique, backfitting optimizes all model free parameters, thus being more time consuming and in principle more accurate.

3.6.1 Tree refitting

Let us consider the vector q of all the labels of the terminal nodes of the tree, such that $q_j = L_j$, with $j = 1 \ldots K+1$, 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 define vector $\hat{Y}$ by $\hat{Y}_i = \hat{\mu}_C(o_i)$, vector Y by $Y_i = \mu_C(o_i)$ and matrix M by $M_{ij} = \mu_{S_{L_j}}(o_i)$ with $i = 1 \ldots ||RS||$, $j = 1 \ldots K+1$, where RS represents the set of objects used for the refitting stage.

Objective. Given FDT and a refitting sample of objects RS, find vector q^* of labels so as to minimize the squared error function

$$E(q) = ||Y - \hat{Y}||^2, \tag{3.14}$$

where $\hat{Y} = Mq$ in conformity with eq. (3.3).

Solution. This optimization problem may be solved by matrix inversion. According to equation (2.12) (see section 2.1.3), the solution is

$$q^* = [M^T M]^{-1} M^T Y. \tag{3.15}$$

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 $M^T M$ 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 $M^T M$ matrix, which is only quadratic with respect to the tree complexity.

3.6.2 Tree backfitting

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

$$E(q) = \sum_{o \in BS} (\mu_C(o) - \hat{\mu}_C(o, q))^2, \tag{3.16}$$

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

Strategy. According to eq. (3.3), 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 the LEVENBERG-MARQUARDT nonlinear optimization technique, considered the standard of nonlinear least squares minimization technique [PTVF94]. 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.

3.6.3 LEVENBERG-MARQUARDT optimization

Suppose we are currently at point q_0 in the parameter space and we move to point $q_0 + \delta q$, where δq represents the increment of vector 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:

$$\nabla_q E(q)|_{q_0+\delta q} = \nabla_q E(q)|_{q_0} + \nabla_q^2 E(q)|_{q_0} \delta q. \tag{3.17}$$

$\nabla_q E(q)|_{q_0+\delta q} = 0$ if $q_0 + \delta q$ is the point corresponding to the minimum of E function. By noting

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

eq. (3.17) becomes

$$A\ \delta q = B. \tag{3.18}$$

On the other hand, as (3.17) may be a poor approximation, the Steepest Descent method proposes to take a step down the gradient and to jump with the increment

$$\delta q = constant * B. \tag{3.19}$$

Combining eqs. (3.18) and (3.19), 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 $p = 1 \ldots 3K + 1$

$$A_{pp} = A_{pp}(1 + \lambda), \tag{3.20}$$

where $\lambda = 0$ corresponds to the Newton method and $\lambda \gg 1$ corresponds to the gradient descent method.

3.6.4 Deducing involved matrices

The two matrices A and B are defined in the following way:

$$B_p = \sum_{o \in BS} [\mu_C(o) - \hat{\mu}_C(o)] \frac{\partial \hat{\mu}_C(o)}{\partial q_p}, \quad p = 1 \ldots 3K + 1, \tag{3.21}$$

$$A_{pl} = -\frac{\partial B_p}{\partial q_l} = -\sum_{o \in BS} [\mu_C(o) - \hat{\mu}_C(o)] \frac{\partial^2 \hat{\mu}_C(o)}{\partial q_p \partial q_l} + \sum_{o \in BS} \frac{\partial \hat{\mu}_C(o)}{\partial q_p} \frac{\partial \hat{\mu}_C(o)}{\partial q_l}. \tag{3.22}$$

Considering that close to the solution the first term of A_{pl} is close to zero, the LEVENBERG-MARQUARDT method uses instead, the approximate hessian matrix defined by

$$\tilde{A}_{pl} = \sum_{o \in BS} \frac{\partial \hat{\mu}_C(o)}{\partial q_p} \frac{\partial \hat{\mu}_C(o)}{\partial q_l}, \quad p, l = 1 \ldots 3K + 1, \tag{3.23}$$

which needs only first derivatives to be computed. These first derivatives are:

$$\frac{\partial \hat{\mu}_C(o)}{\partial L_j}, \quad \frac{\partial \hat{\mu}_C(o)}{\partial \alpha_i} \quad \text{and} \quad \frac{\partial \hat{\mu}_C(o)}{\partial \beta_i}. \tag{3.24}$$

The first one may be derived from eq. (3.3) as

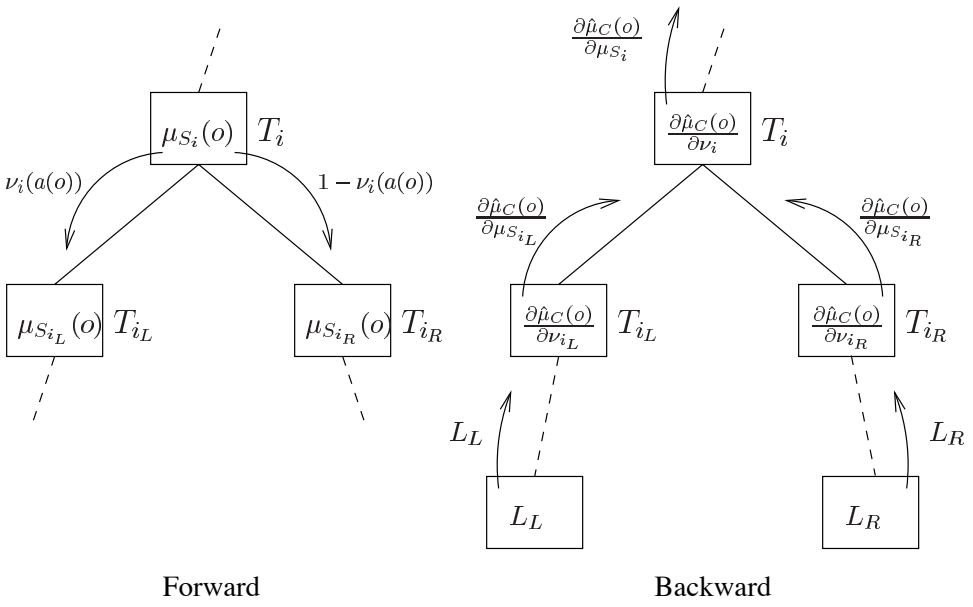


Figure 3.11: Computation of partial derivatives

$$\frac{\partial \hat{\mu}_C(o)}{\partial L_j} = \mu_{S_{L_j}}(o), \qquad j = 1 \dots K+1. \tag{3.25}$$

For the other two first derivatives we notice that $\hat{\mu}_C(o)$ depends on α_i and β_i only through the discriminator function $\nu_i(\alpha_i, \beta_i, a(o))$. For example

$$\left.\frac{\partial \hat{\mu}_C(o)}{\partial \alpha_i}\right|_{\alpha_i,\beta_i,a(o)} = \left.\frac{\partial \hat{\mu}_C(o)}{\partial \nu_i}\right|_{\alpha_i,\beta_i,a(o)} * \left.\frac{\partial \nu_i(o)}{\partial \alpha_i}\right|_{\alpha_i,\beta_i,a(o)}. \tag{3.26}$$

The crux is to compute efficiently the derivative $\frac{\partial \hat{\mu}_C(o)}{\partial \nu_i}$ for all the nodes of the tree. This is achieved using an ad hoc variant of the back-propagation algorithm described below.

3.6.5 Partial derivatives computation

The most delicate part of the optimization procedure is the computation of the partial derivatives $\frac{\partial \hat{\mu}_C(o)}{\partial \nu_i}$, $i = 1 \dots K$. Figure 3.11 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 $\nu_i(a(o))$ based on current parameters q. We send this information to the two successors, so as to compute $\mu_{S_{i_L}}(o) = \nu_i(a(o))\mu_{S_i}(o)$ at left successor T_{i_L}, and $\mu_{S_{i_R}}(o) = [1 - \nu_i(a(o))]\mu_{S_i}(o)$ 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 $\frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_L}}}$ from its left successor T_{i_L} (which equals label L_L if this successor is a terminal node) and $\frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_R}}}$ from its right successor T_{i_R} (which equals label L_R if this successor is a terminal node). T_i node also has at disposal $\mu_{S_i}(o)$ and $\nu_i(a(o))$ computed at forward stage. Thus, we may calculate the partial derivative we searched for at node T_i:

$$\frac{\partial \hat{\mu}_C(o)}{\partial \nu_i} = \mu_{S_i}(o)[\frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_L}}} - \frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_R}}}], \quad and \tag{3.27}$$

$$\frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_i}} = \nu_i(a(o)) \frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_L}}} + [1 - \nu_i(a(o))] \frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_R}}}. \tag{3.28}$$

Eqs. (3.28) 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.

Given object $o \in BS$ and the fuzzy tree FDT, the algorithm for computing all $\frac{\partial \hat{\mu}_C(o)}{\partial \nu_i}$ for all test nodes T_i of $FDT, i = 1 \ldots K$ is as follows:

for (every node T_i of FDT) do in a backward manner (i.e. $i = K \ldots 1$)
1 compute $\frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_L}}}$ and $\frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_R}}}$
Note: **if** $T_{i_{L,R}}$ is a terminal node **then** $\frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_{L,R}}}} = L_{L,R}$
2 compute $\frac{\partial \hat{\mu}_C(o)}{\partial \nu_i}$ with eq. (3.27)
3 if (T_i is not the root node) (i.e. $i \neq 1$) **then** compute $\frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_i}}$ with eq. (3.28).

When we say "backward manner" we mean "in the inverse order of the nodes creation". The nodes have been created in a depth first procedure, which means that a successor of the current node has an index i greater than the index of the current node.

3.6.6 Deducing partial derivatives formulas

For a test node T_i, next two formulas are true:

$$\frac{\partial \hat{\mu}_C(o)}{\partial \nu_i} = \frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_L}}} \frac{\partial \mu_{S_{i_L}}(o)}{\partial \nu_i} + \frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_R}}} \frac{\partial \mu_{S_{i_R}}(o)}{\partial \nu_i} \tag{3.29}$$

$$\frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_i}} = \frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_L}}} \frac{\partial \mu_{S_{i_L}}(o)}{\partial \mu_{S_i}} + \frac{\partial \hat{\mu}_C(o)}{\partial \mu_{S_{i_R}}} \frac{\partial \mu_{S_{i_R}}(o)}{\partial \mu_{S_i}} \tag{3.30}$$

where $\mu_{S_{i_L}}(o)$ and $\mu_{S_{i_R}}(o)$ are the membership degrees at the left and respectively the right descendants of node T_i for object o.

We may write also that

$$\mu_{S_{i_L}}(o) = \mu_{S_i}(o)\nu_i(o) \tag{3.31}$$

$$\mu_{S_{i_R}}(o) = \mu_{S_i}(o)(1 - \nu_i(o)) \tag{3.32}$$

By derivating these last relations we get

$$\frac{\partial \mu_{S_{i_L}}(o)}{\partial \nu_i} = \mu_{S_i}(o); \quad \frac{\partial \mu_{S_{i_R}}(o)}{\partial \nu_i} = -\mu_{S_i}(o) \tag{3.33}$$

$$\frac{\partial \mu_{S_{i_L}}(o)}{\partial \mu_{S_i}} = \nu_i(o); \quad \frac{\partial \mu_{S_{i_R}}(o)}{\partial \mu_{S_i}} = 1 - \nu_i(o) \tag{3.34}$$

Replacing the derivatives (3.33) in formula (3.29) we get equation (3.27). Replacing the derivatives (3.34) in formula (3.30) we get equation (3.28).

3.6.7 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 ΔE_{min}. Also, if the algorithm did not converge in a certain number of cycles to the imposed ΔE_{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 $FDT(q)$ corresponding to set q of parameters, the threshold value for the error gain ΔE_{min}, the starting value for the algorithm factor λ_{init} and its step for adjustment λ_{step} is:

1 set $\lambda = \lambda_{init}$
2 Forward: compute $E(q)$ with eq. (3.16)
3 Backward: compute first derivatives $\frac{\partial \hat{\mu}_C(o,q)}{\partial q_p}$ with eqs. (3.25) and (3.26) and
set matrices A and B with eqs. (3.23) and (3.21)
4 alter matrix A with eq (3.20)
5 solve linear system $A\ \delta q = B$ by Gauss elimination
6 Forward: compute $E(q + \delta q)$ with eq. (3.16)
7 compute gain $\Delta E(q) = E(q) - E(q + \delta q)$
8 if $0.0 \leq \Delta E(q) \leq \Delta E_{min}$ **then** $q^* = q + \delta q$; go to **11** (end)
9 if $\Delta E(q) < 0.0$ **then** $\lambda = \lambda * \lambda_{step}$; go to **4** (alter)
10 if $\Delta E(q) > \Delta E_{min}$ **then** $\lambda = \frac{\lambda}{\lambda_{step}}$; $q = q + \delta q$; go to **3** (backward)
11 Update all others FDT parameters (tests in nodes, ...)

3.7 Variance and bias studies

The bias-variance decomposition of the error rate (see eq. (2.18) in section 2.1.4) is a useful tool to understand the behavior of learning algorithms.

Objectives. From this point of view, the behavior of our FDT algorithm has been studied in detail. The experiments included two different variance and bias studies: i) a global variance and bias study, in order to see how the variance changes in the FDT case and to compare it to the standard crisp regression and decision trees case and to the aggregation methods 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 FDT.

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 FDT built from a random learning set LS of size N at a point $o \in U$ of the universe, leaving the fact that actually $\hat{\mu}_{C,LS}(o) = \hat{\mu}_{C,LS}(a_1(o), \ldots, a_m(o))$ 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 FDT learning algorithm can be written as

$$variance_{FDT} = E_U\{E_{LS}\{(\hat{\mu}_{C,LS}(o) - E_{LS}\{\hat{\mu}_{C,LS}(o)\})^2\}\} \tag{3.35}$$

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

The *global bias* of the FDT algorithm reflects the systematic error that the FDT is having in average with respect to the Bayes model when trained on learning sets of the same finite size. Since for the analysed problems (datasets), the Bayes model is not known a priori, the bias of the FDT algorithm cannot be calculated separately from the residual error only from data. However, in order to study the global bias relative evolution, we compute the sum of the residual error and bias terms, since the residual error is a constant term:

$$bias^2_{FDT} + residual\_error = E_U\{(\mu_C(o) - E_{LS}\{\hat{\mu}_{C,LS}(o)\})^2\}. \tag{3.36}$$

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

$$bias_\alpha = \alpha_{asymptotic} - E_{LS}\{\alpha_{LS}\}, \quad variance_\alpha = E_{LS}\{(\alpha_{LS} - E_{LS}\{\alpha_{LS}\})^2\} \tag{3.37}$$

where $\alpha_{asymptotic}$ 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 $FDTs$ 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 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. In this case, formula (3.35) becomes eq. (2.22), formula (3.36) becomes eq. (2.24), and formulas (3.37) become eqs. (2.26) and (2.27) (see section 2.1.4).

3.8 Variants of the FDT algorithm

We collected in this last section the variants of our FDT method. They are tested in some of our simulations in chapter 4.

3.8.1 Variant of FIBONACCI search for β width

We tried in some of our simulations a variant of searching for β width. Since there is a link between the number N of function evaluations needed to find the minimum and the number of the points in the interval (here the number of the attribute values of objects in the interval), we decided empirically to perform a certain number of evaluations when the size of the local growing set of the node is "small" and a *larger* number of function evaluations when the node is "large". We defined "small" and "large" as:

$$\begin{cases} \text{large node} & \text{if } ||GS||_{local} \geq \frac{2}{3}||GS|| \\ \text{small node} & \text{otherwise} \end{cases} \tag{3.38}$$

where $||GS||$ represents the size of the whole growing set size (the number of its objects) and $||GS||_{local}$ - the size of the local growing set in the node. In these experiments we have chosen to make 10 evaluations of the error function in the case of large nodes and 5 evaluations in the case of small nodes.

3.8.2 Variant for pruning

Global error of the subtrees as relevance measure

We noticed that after pruning, the model accuracy does not seem to improve for certain data. Indeed, the relevance measure we use for pruning does not guaranty that the obtained sequence of trees includes the best possible sub-trees of the complete tree. As a nested pruning algorithm, our pruning may miss some good trees found by an optimal pruning algorithm. The main difference between nested pruning algorithms lies in the methods used for choosing the next node to prune.

We thus tried another relevance measure: we compute at the growing stage for each test node the estimate of the *global error of the subtree rooted at this node*. And we sort these nodes by increasing order of their relevance. The theoretical advantage of this measure is the global character it has in opposition to the local character of the previous squared error relevance measure: it measures the error over the whole subtree of a node instead of the error of that node.

Suppose a current test node S_i with its left successor S_{i_L} and right successor S_{i_R}. The global error of the subtree rooted at node S_i is

$$E_{S_i} = \sum_{o \in S_i} \mu_{S_i}(o)[\mu_C(o) - \hat{\mu}_{C_{S_i}}(o)]^2 \tag{3.39}$$

where $\mu_{S_i}(o)$ represents the membership degree of object o to fuzzy set of node S_i and $\hat{\mu}_{C_{S_i}}(o)$ - the output approximated by the subtree rooted at node S_i calculated as a function of the outputs approximated by the subtrees rooted in the successors of the node: the left subtree has the estimation of the output $\hat{\mu}_{C_{S_{i_L}}}(o)$ and the right subtree has the estimation of the output $\hat{\mu}_{C_{S_{i_R}}}(o)$. Thus

$$\hat{\mu}_{C_{S_i}}(o) = \nu_{S_i}(o)\hat{\mu}_{C_{S_{i_L}}}(o) + (1 - \nu_{S_i}(o))\hat{\mu}_{C_{S_{i_R}}}(o) \tag{3.40}$$

where $\nu_{S_i}(\cdot)$ represents the discriminator at node S_i. If the quantities of equations (3.39) and (3.40) are computed in a backward manner (in the inverse order of the nodes creation), starting from the terminal nodes of the complete tree, we obtain in a single pass, the errors we are interested for at all subtrees of the complete tree. Every non-terminal node S_i thus receives $\hat{\mu}_{C_{S_{i_L}}}(o)$ from its left successor S_{i_L} (which equals label L_L if this successor is a terminal node) and $\hat{\mu}_{C_{S_{i_R}}}(o)$ from its right successor S_{i_R} (which equals label L_R if this successor is a terminal node). In this way, the time complexity of this computation is bounded by $\mathcal{O}(K \cdot ||GS||)$, thus it is linearly in the size of the growing set and in the tree complexity.

Given the tree FDT, the algorithm for computing E_{S_i} for all test nodes S_i of FDT, $i = 1 \dots K$ is like follows:

for (every node S_i of FDT) compute in a backward manner (i.e. $i = K \dots 1$)
1 for (all $o \in S_i$)
$\hat{\mu}_{C_{S_i}}(o) = \nu_{S_i}(o)\hat{\mu}_{C_{S_{i_L}}}(o) + (1 - \nu_{S_i}(o))\hat{\mu}_{C_{S_{i_R}}}(o)$
Note: **if** $S_{i_{L,R}}$ is a terminal node **then** $\hat{\mu}_{C_{S_{i_{L,R}}}}(o) = L_{L,R}$
2 $E_{S_i} = \sum_{o \in S_i} \mu_{S_i}(o)[\mu_C(o) - \hat{\mu}_{C_{S_i}}(o)]^2$.

3.8.3 Refitting during pruning approach

This approach combines pruning and refitting so as to obtain a more accurate fuzzy decision tree. It consists in refitting *every* tree from the sequence of pruned trees, *before* choosing the best pruned one. This approach has been tested in chapter 4, but a more rigorous and systematic validation still has to be done in the future. Also, further work should be done in terms of computational

complexity, so as to reduce the amount of computations which then becomes cubic in terms of model complexity before pruning.

Chapter 4

Experimental evaluation of the proposed FDT method

In this chapter, the new introduced fuzzy decision tree method is validated by experimental trials on 18 datasets. The results herein attempt to respond to questions like what is the place of the fuzzy decision tree methods amongst different crisp and aggregation learning methods, what is the impact of each stage of building on the fuzzy decision tree results, and to study the bias-variance tradeoff of multiple methods, various FDT behaviors. Finally, the last part of the chapter concerns the FDT visualization.

In order to consistently validate our method, we report here two complementary protocols of results. Part I studies the behavior of the proposed FDT method on 7 large databases of several thousand objects since in data mining analyses we are often confronted with large amounts of data. Model accuracies are estimated by averaging multiple holdout estimates. Global model variance and bias, and parameter variance and bias studies are also done on these datasets. Part II applies FDT 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. In this second part, the goal is to compare FDT with standard tree-structured interpretable methods on standard datasets, using a standard protocol.

4.1 The automatic learning methods used in comparisons

We compare the fuzzy decision tree with two packages of methods:

- Firstly, with *three standard decision and regression tree methods*, thus three interpretable tree-structured-inductive methods: standard C4.5 [Qui93] decision trees of Quinlan, standard CART [BFOS84] regression trees of Breiman and Wehenkel's method of decision trees published in [WP93, Weh98], called in what follows ULG method. The last two methods are implemented at the University of Liege, Belgium and the first one is C4.5 Release 8, taken from the *http* address [1]: www.cse.unsw.edu.au/∼quinlan/.

- Secondly, with *five aggregation methods*, thus methods to reduce variance: dual perturb and combine [Geu01] (noted dual P&C), bagging [Bre96], boosting [FS96], extra-tree averaging [Geu02] and a combination of bagging with dual perturb and combine algorithm [Geu02] (called bag.+dual P&C further on). All the results corresponding to these algorithms are taken from the PhD thesis of Pierre Geurts [Geu02]. We followed in purpose a protocol

[1] The command used is: *c4.5 -f database_name -u*.

of experiments identical to the one employed in this reference and identical data bases, in order to be able to compare our FDT method with its results and to situate better our method among the most accurate automatic learning methods. Moreover, this author has also done bias and variance studies on its methods, so we could be able to compare our method with these ones also from this point of view, fact which completes our analysis. We have succinctly described all these methods in section 2.1.6 of chapter 2.

The simulations in the first package of methods have been carried out by us, together with the simulation of the FDT method. CART and ULG trees were completely grown and then pruned (using a multiplier $\sigma = 1$ in the one-standard-error-rule; the meaning of this multiplier is given in section 2.1.6 of chapter 2). ULG method uses an optimal backward post-pruning in conformity with [Weh92b]. CART and FDT were trained for a regression goal. Afterwords, 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 FDT were built. The compared methods including the FDT one were all run on the same computer and programming language (Common 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 (so called pessimistic) estimates of the error rate computed on growing set GS (see [Qui93] and section 2.1.6 of chapter 2).

The second package of methods has as a starting point the ULG method of decision trees. Bagging, boosting, extra tree averaging and bagging + dual perturb and combine methods amount to build several ULG standard trees (a number of 25 trees has been considered by [Geu02]). In the case of bagging, the trees were fully developed but grown on the basis of a boostrap sample. In the case of boosting (AdaBoost.M1 algorithm), the trees were pre-pruned and built on the basis of the full learning sample with changing weights. The strategy for pre-pruning the ULG tree was a stop-splitting criterion based on the G^2 statistic with $\alpha = 0.005$ (see [WP93]). Concerning the extra-tree averaging, the method is based on extremely randomized trees, i.e. trees with a very permissive splitting rule which selects at random a pair attribute-threshold. In order to avoid very bad tests, the chosen attribute is rejected in an extra-tree if it produces a value of the normalized score less than a threshold a priori defined: [Geu02] considered this threshold 0.1. Dual perturb and combine and bagging + dual perturb and combine methods add some noise to the attributes at the prediction level, noise regulated by a parameter λ dependent on the learning task. In their studies, the optimal value of this parameter is automatically determined on a holdout estimate (the pruning set). When speaking about pruned ULG, reference [Geu02] refers to a multiplier $\sigma = 0$ in the one-standard-error-rule. This is because all his methods are methods to reduce variance, and thus for an improved accuracy, it is better to start from decision trees with as less as possible bias, thus from fully grown trees or pruned trees with $\sigma = 0$ instead from pruned trees with $\sigma = 1$. All the methods were trained on the crisp classification goal.

4.2 Part I

4.2.1 The experimental methodology

Datasets used

Table 4.1 describes the datasets used for method validation in Part I of experiments. The column $||Samples||$ displays the whole number of learning instances in each database, column $||Pool\ set||$ refers to the set from which random growing sets GS of the same size are drawn by sampling *without* replacement in order to build multiple models for each dataset (see paragraph "Bias and variance estimation" of section 2.1.4, chapter 2), $||PS||$ is the number of the learning instances in the *fixed* pruning set PS and $||TS||$ is the number of test instances in the *fixed* test set

Table 4.1: Datasets - Part I

Dataset	$\|\|Attributes\|\|$	$\|\|Classes\|\|$	$\|\|Samples\|\|$	$\|\|Pool\ set\|\|$	$\|\|PS\|\|$	$\|\|TS\|\|$	$\sigma^2_C(\%)$
Gaussian	2	2	20000	14000	2000	4000	11.5
Twonorm	20	2	10000	7000	1000	2000	2.3
Omib	6	2	20000	14000	2000	4000	0.0
Waveform	21	3	5000	3000	1000	1000	14.0
Satellite	36	6	6435	3435	1000	2000	n/a
Pendigits	16	10	10992	5494	2000	3498	n/a
Dig44	16	10	15000	9000	2000	4000	n/a

TS. The last column (σ^2_C) gives whenever available, the Bayes error rate, i.e the minimal error one can hope to obtain given the learning task (see section 2.1.4 for its definition). The attributes in datasets are all numerical. The seven datasets investigated and the associated *classification* tasks are detailed in section B.1 of the appendix.

Protocol of experiments

Because of the instability of the learning algorithms and especially of the tree-based methods, any comparison of two different algorithms must involve multiple runs of the algorithms instead of a single run, in order to diminish the learning set variation and the algorithm internal randomness as sources of variation. That is why all the results simulated for this part I of the experiments are computed as averages over a number of $q = 5$, $q = 20$ or $q = 25$ of trees. The variance coming from the test set is also diminished by choosing sufficiently large test sets for all experimented datasets.

Concerning the first package of methods, three different strategies have been adopted:

- In the context of global bias and variance studies, the reported results (error rate, complexity, mean squared error, global variance and global bias) were obtained by averaging over a number of $q = 25$ trees built on (GS, PS) pairs, where GS is randomly chosen from the pool set. Here, there is a must in having a sufficiently large number of trials in order to compute an average, as we stress in section 2.1.4. We considered $q = 25$ as a sufficiently large number of trials in order to reflect the real variance and bias tradeoff.

- In the context of parameter bias and variance studies, the reported results were obtained by averaging over a number of $q = 20$ trees.

- For other complementary studies judged interesting concerning the FDT algorithm behavior (as the evolution of results with the growing set size, or different variants of the proposed method) the reported results were obtained by averaging over a number of $q = 5$ trees.

Concerning the second package of methods, the results taken from [Geu02] are obtained by averaging over a number of $q = 50$ learning models[2]. Bagging, boosting, bagging + dual perturb and combine and extra tree averaging necessitate, for each randomly selected learning set, the building of 25 standard ULG decision trees. We considered the same pool set, pruning set and test set as in [Geu02].

In the refitting and backfitting steps, we generally use the whole learning set : $LS = GS \cup PS$, since all these data are available. However, refitting and/or backfitting sets may consist uniquely of

[2]When computing and comparing bias and variance quantities, it does not matter how large is the number of trials, provided it is sufficiently large. Also, average complexities and error rates can be compared if the number of trials is sufficiently large.

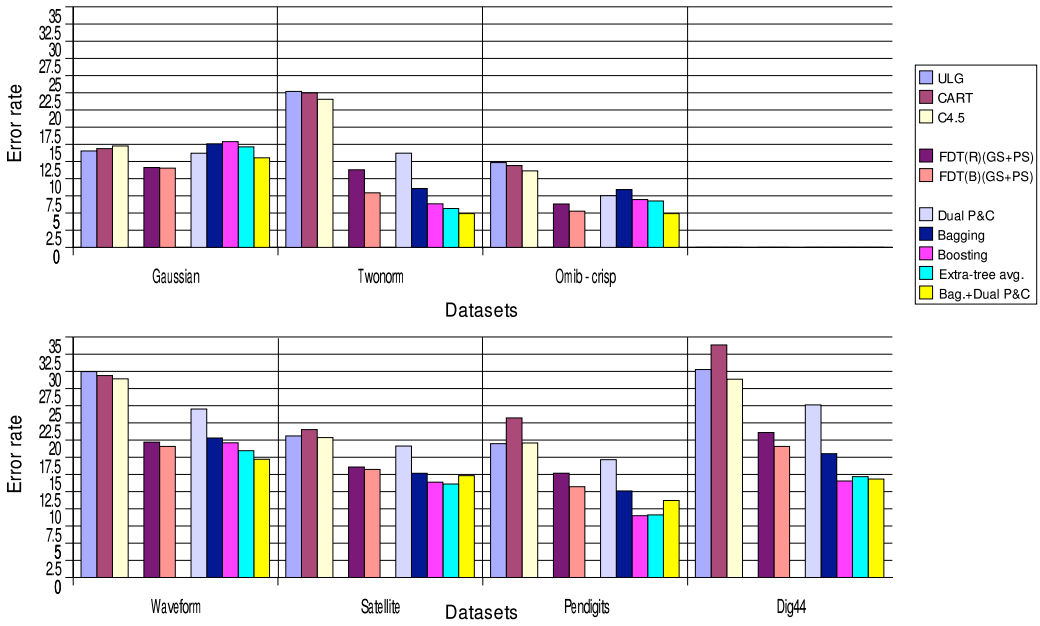


Figure 4.1: Comparison of methods in terms of average error rates

the growing set GS in some of our experiments, basically in the case of variance and bias studies, for the fairness of comparison with other methods in terms of bias and variance quantities[3].

4.2.2 Comparing FDT with different automatic learning methods

As support for the following discussions, tables D.1, D.2, D.3 and D.4 are providing full results (results of this sub-section have been synthesized from these tables of the appendix).

Accuracy

Table 4.2 presents a summarized comparison of the average error rates of all the investigated algorithms. For each data base a single GS size has been considered here. Only for the omib database, two classifications have been tried: one having a crisp output and one having a fuzzy output: here, results for crisp output definition are indicated[4]. This was possible because we were able to define the fuzzy output, which was not the case of the other databases. Two discriminators have been tried: piecewise linear and sigmoidal [5]. Figure 4.1 shows graphically the values of table 4.2. In these graphics, for each database, three clusters of methods are distinguished. Each cluster represents one of the classes of methods: crisp trees, fuzzy decision trees and aggregation methods.

Firstly, fuzzy decision trees, in all its variants (full, pruned, refitted or backfitted, piecewise linear discriminator or sigmoidal), are more accurate (sometimes much more) than standard decision and regression trees: ULG, C4.5 and CART. Secondly, FDT refitting and backfitting results on $GS + PS$ are more accurate than dual perturb and combine results for all 7 datasets. Thirdly,

[3]Throughout the thesis, when refitting or backfitting results do not have specified if they are obtained on GS or $GS \cup PS$, the reader should consider them by default obtained on $GS \cup PS$.

[4]Throughout the thesis, when omib results do not have specified if they are obtained on the fuzzy or the crisp class, the reader should consider them by default obtained on the fuzzy class.

[5]Throughout the thesis, when results do not have specified if they are obtained on the piecewise linear or sigmoidal discriminator, the reader should consider them by default obtained on the piecewise linear discriminator.

Table 4.2: Summarized comparison: average error rate of different algorithms on 7 databases

Method	Gaussian GS=100	Twonorm GS=300	Omib-crisp GS=500	Waveform GS=300	Satellite GS=500	Pendigits GS=500	Dig44 GS=500
q = 25							
Pruned ULG ($\sigma = 1$)	14.04	22.70	12.34	29.97	20.60	19.48	30.27
Pruned CART	14.38	22.48	11.90	29.39	21.52	23.22	33.84
Pruned C4.5	14.76	21.55	11.13	28.91	20.36	19.58	28.86
q = 25, Piecewise linear discriminator							
Full FDT	13.08	14.47	9.84	23.34	16.43	17.01	25.85
Pruned FDT	12.22	14.52	10.25	22.06	16.99	16.98	24.31
Ref. FDT (GS+PS)	11.62	11.29	6.32	19.70	16.08	15.18	21.10
Backf. FDT (GS+PS)	11.53	7.94	5.27	19.08	15.73	13.22	19.08
q = 25, Sigmoidal Discriminator							
Full FDT	13.70	15.36	9.96	22.99	16.60	16.95	27.03
Pruned FDT	12.25	15.65	10.75	21.28	16.33	17.30	25.22
Ref. FDT (GS+PS)	11.72	11.31	6.51	19.23	15.80	15.66	21.71
Backf. FDT (GS+PS)	11.64	11.98	5.49	19.45	15.54	14.67	21.13
q = 50, (taken from [Geu02])							
Full ULG	17.57	21.59	12.13	29.87	20.78	19.03	30.30
Pruned ULG ($\sigma = 0$)	13.72	21.43	12.23	28.73	19.86	18.95	29.65
Dual P&C	13.71	13.72	7.52	24.51	19.13	17.15	25.11
Bagging	15.07	8.57	8.42	20.30	15.17	12.60	18.02
Boosting	15.39	6.34	6.96	19.60	13.88	9.00	14.05
Extra-tree avg.	14.63	5.66	6.76	18.45	13.61	9.12	14.68
Bag. + dual P&C	13.03	4.92	4.92	17.21	14.82	11.21	14.33
Bayes	11.5	2.3	0.0	14.0	n/a	n/a	n/a

FDTs are less effective than the best combination of ensemble methods (boosting, extra-tree averaging or bagging + dual perturb and combine), with one exception (gaussian data).

Generally, FDT results with piecewise linear discriminator are slightly better than sigmoidal discriminator results [6].

Table 4.3: Error gain (%) of FDT versus C4.5

Gaussian GS=100	Twonorm GS=300	Omib-crisp GS=500	Waveform GS=300	Satellite GS=500	Pendigits GS=500	Dig44 GS=500	Avg. effect
R ($GS + PS$) versus C4.5							
21.3	47.6	43.2	31.9	21.0	22.5	26.9	30.6
B ($GS + PS$) versus C4.5							
21.9	63.2	52.7	34.0	22.7	32.5	33.9	37.3

Table 4.3 quantifies the error gain [7] of refitted (R) and backfitted (B) versions of FDT over C4.5 accuracy. There is an average error gain on the 7 datasets of 30.6% of refitted FDT with respect to C4.5 and 37.3% of backfitted FDT with respect to C4.5 respectively.

[6] Each time we will further compare FDT results with others, we take as reference the best FDT results, i.e. FDT with piecewise linear discriminator results.

[7] We computed the error gain (%) of FDT versus C4.5 as $\frac{Pe_{C4.5} - Pe_{FDT}}{Pe_{C4.5}} 100$.

Table 4.4: Summarized comparison: average complexity of different algorithms on 7 databases

	Gaussian GS=100	Twonorm GS=300	Omib-crisp GS=500	Waveform GS=300	Satellite GS=500	Pendigits GS=500	Dig44 GS=500
q = 25							
Pruned ULG ($\sigma = 1$)	3.3	15.1	20.5	12.6	11.1	32.1	38.7
Pruned CART	5.5	18.4	25.4	37.8	52.8	85.0	104.0
Pruned C4.5	2.4	18.4	22.9	26.1	36.0	32.0	55.4
q = 25, Piecewise linear discriminator							
Full FDT	90.6	191.2	141.0	398.1	446.4	294.0	663.0
Pruned FDT	22.4	71.5	48.5	90.0	68.4	123.0	175.0
q = 25, Sigmoidal Discriminator							
Full FDT	89.7	184.4	143.8	400.2	422.4	271.0	651.0
Pruned FDT	18.1	73.9	46.3	113.1	73.2	122.0	167.0
q = 50, (taken from [Geu02])							
Full ULG	15	25	37	38	58	48	90
Pruned ULG ($\sigma = 0$)	4	23	27	19	23	44	56
Dual P&C	15	25	37	38	58	48	90
Bagging	248	444	681	662	1012	953	1622
Boosting	25	262	451	404	729	724	1047
Extra-tree avg.	813	2158	2713	3341	4673	5402	8030
Bag. + dual P&C	39	450	690	664	1003	940	1621

Complexity

Table 4.4 shows the average complexities of the trees of table 4.2. Let us recall that tree complexity is defined as the number of the test nodes. In the case of multi-class problems, for all the methods, excepting ULG and C4.5 methods, the complexities displayed are the *total* number of test nodes of the models over the forest.

We observe that pruned (and a fortiori fully grown) FDT models are generally more complex than all decision and regression tree models: ULG, C4.5 and CART, for all datasets. They are also more complex than dual perturb and combine models for all datasets. Bagging, boosting, extra-tree averaging and bagging + dual perturb and combine are much more complex models, since they are obtained by averaging a number of 25 trees in each case.

Piecewise linear and sigmoidal discriminators give slightly the same complexity of FDT models.

CPU time

Table 4.5 displays a single typical run CPU time for FDT, ULG and CART models. We may compare their CPU times since they are obtained on the same computer, a Linux Pentium III, 1.0GHz, 512MB machine and they are programmed all in Common Lisp programming language[8]. For multi-class problems the time displayed is the time to build the whole forest of CART and FDT models.

The CPU times reflect a big gap between FDTs and the other tree-structured models. In average, refitted FDT on GS+PS is approximately 20 times and backfitted FDT on GS+PS is 50 times less fast than pruned ULG decision trees.

This is the price paid for having a more accurate and still interpretable tool in the same time. Since Lisp is not the fastest language in terms of computational efficiency, hopefully CPU times

[8] All the CPU times displayed are CPU times excluding garbage-collecting CPU times.

Table 4.5: Summarized comparison: a single typical run CPU time (in seconds) of different algorithms on 7 databases

Method	Gaussian GS=100	Twonorm GS=300	Omib-crisp GS=500	Waveform GS=300	Satellite GS=500	Pendigits GS=500	Dig44 GS=500
Pruned ULG	5	5	6	3	8	7	11
Pruned CART	3	2	6	4	12	21	32
Piecewise linear discriminator							
Full FDT	32	54	41	44	107	95	292
Pruned FDT	34	61	50	52	115	116	337
Ref. FDT (GS)	38	67	65	60	121	132	406
Backf. FDT (GS)	50	406	122	79	165	206	493
Ref. FDT (GS+PS)	63	101	78	51	129	156	484
Backf. FDT (GS+PS)	90	598	340	115	205	545	844

would be much improved if the FDT algorithms would be programmed in C programming language for example.

Interpretability

From the point of view of interpretability, all standard decision and regression trees have primacy, among the studied algorithms. They are the most interpretable, since they are the less complex. Dual perturb and combine method and fuzzy decision tree are also interpretable methods, since the former builds a single model which is then perturbed at the prediction stage and the later preserves interpretability with the amendment that multiple paths of a tree must be aggregated in order to obtain the final answer. The other aggregation methods (bagging, boosting, extra-tree averaging and bagging + dual perturb and combine) are not at all interpretable methods, since they are based integrally on averaging.

Table 4.6 sketches this characteristic for all the investigated methods, and also other features like accuracy, rapidity and autonomy in our view. It is a qualitative not a quantitative description, a minus (-) meaning the method does not present the feature, a plus (+) meaning it presents the feature, a (++) - it presents the feature more strongly.

Table 4.6: Summarized qualitative comparison between learning algorithms

	FDT(R) FDT(B)	ULG CART C4.5	Dual P&C	The other aggregation methods
Accurate	+	–	-	++
Rapid	-	++	+	-
Autonomous	+	+	+	+
Interpretable	+	+	+	-

Fuzzy versus crisp output definition

We recall the discussion of section 3.1 explaining the difference between a fuzzy and a non fuzzy (a crisp 0/1) output. In practice, there are a lot of problems where we can define fuzzy outputs: on a continuous index we define a transition region based on the information about the problem. Thus

we may take profit of this supplementary information. We illustrate here the difference in results with the omib database example, where we have been able to define a fuzzy output. Until now, we discussed the results on the omib data with the crisp output. Table 4.7 puts side by side the results for crisp and fuzzy outputs. We may notice that the fuzzy output highly outperforms crisp output results in terms of accuracy. The pruned models are smaller in the crisp case. We should stress on the fact that a fuzzy output contains a richer information about the problem (the process) than the crisp output. This is one cause of the improved results. The other case lies on the fuzzy decision tree method itself and in its power to operate such kind of information with good results.

Table 4.7: Comparing error rate and complexity between crisp and fuzzy output for omib data, $q = 25, GS = 500$, piecewise linear discriminator

Method	fuzzy		crisp	
	C	Pe(%)	C	Pe(%)
Pruned CART	67.0	11.91	25.4	11.90
Full FDT	125.9	8.75	141.0	9.84
Pruned FDT	59.9	9.58	48.5	10.25
Ref. FDT (GS)	59.9	5.24	48.5	7.83
Backf. FDT (GS)	59.9	4.47	48.5	8.86
Ref. FDT (GS+PS)	59.9	4.62	48.5	6.32
Backf. FDT (GS+PS)	59.9	3.01	48.5	5.27

What about large growing set sizes?

The previous study has been done for a single growing set size experiment and this size has not been chosen very large, so as to reduce the time required to carry out the experiments. Table 4.8 shows results for a larger GS size ($GS = 2000$), for FDTs in comparison with C4.5, ULG and CART results ($q = 5$ trees).

We may observe that at large growing set size, refitting and backfitting on $GS + PS$ FDTs are more accurate than C4.5, CART or ULG methods. Also, FDTs just after pruning, are more accurate than C4.5, CART or ULG for gaussian, twonorm, waveform and dig44 datasets. For gaussian data, FDT results in all its forms are about near the boundary of the Bayes error, which is around 11%. From the complexity point of view, FDT models are approximately as complex as CART models, and generally more complex than ULG models.

4.2.3 The impact of the four stages on the FDT results

After comparing FDT results with other methods results, let us analyse the effects of each stage of the FDT building (pruning, refitting and backfitting) on the model results, in terms of accuracy, complexity, CPU time or interpretability.

Effect of pruning

Figure 4.2 displays error rates for full (F), pruned (P), refitted (R) and backfitted (B) fuzzy decision trees, corresponding to the table 4.2. We may observe that pruning has a positive effect on the error rate for gaussian, waveform and dig44 and has no significant effect for twonorm and pendigits. For omib and satellite datasets, pruning worsens the accuracy.

Table 4.9 shows the effect of pruning on the complexity reduction[9], computed on the results

[9] We computed the complexity reduction (%) of pruning as $\frac{C_{before} - C_{after}}{C_{before}} 100$.

Table 4.8: Summarized comparison at large growing set size ($GS = 2000$): average error rate and complexity of different algorithms on 7 databases (q = 5)

Method	Gaussian	Twonorm	Omib-crisp	Waveform	Satellite	Pendigits	Dig44
Average error rate							
Pruned ULG	12.11	18.42	9.22	26.12	16.80	12.77	22.32
Pruned CART	12.76	17.69	8.44	24.06	16.65	13.95	23.74
Pruned C4.5	12.06	17.30	8.30	25.46	16.12	11.46	19.92
FDT with Piecewise linear discriminator							
Full FDT	11.75	7.38	7.65	17.12	14.15	10.31	16.07
Pruned FDT	11.49	7.98	8.95	17.96	16.14	11.41	15.81
Ref. FDT (GS+PS)	11.38	8.92	5.29	17.44	14.93	10.37	14.70
Backf. FDT (GS+PS)	11.52	3.59	4.66	17.40	14.43	9.62	14.03
Average complexity							
Pruned ULG	8.6	71.0	53.8	40.4	36.4	69.2	74.8
Pruned CART	37.6	83.2	64.8	165.6	128.4	189.0	256.0
Pruned C4.5	5.6	98.4	61.8	136.0	112.2	72.4	156.0
FDT with Piecewise linear discriminator							
Full FDT	114.4	239.6	145.2	561.9	687.0	508.0	1219.0
Pruned FDT	38.0	90.2	52.4	133.8	82.2	171.0	230.0

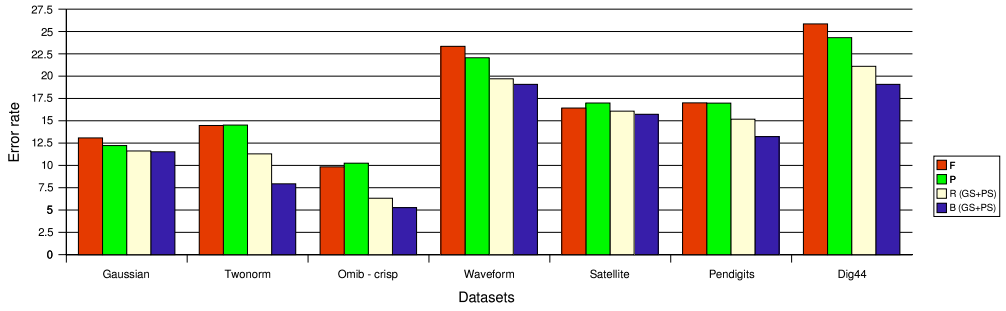


Figure 4.2: The impact of the four stages of the FDT on the error rate

of tables 4.4 and 4.8. Thus, in average over the 7 datasets, pruning reduces complexity with about 70%. 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. This drastic reduction of tree complexity after pruning has of course a positive effect on the model interpretability.

Table 4.9: Effect of FDT pruning: complexity reduction (%) on 7 databases

	Gaussian GS=100	Twonorm GS=300	Omib-crisp GS=500	Waveform GS=300	Satellite GS=500	Pendigits GS=500	Dig44 GS=500	Avg. effect
q = 25								
pw	75.3	62.6	65.6	77.4	84.7	58.2	73.6	71.1
sig	79.8	59.9	67.8	71.7	82.7	55.0	74.4	70.2
GS=2000, q = 5								
pw	66.8	62.4	63.9	76.2	88.0	66.3	81.1	72.1

Effect of refitting

Analysing again figure 4.2, we see that refitting on $GS + PS$ has a positive effect on the error rate for all 7 datasets. This is quantified also by table 4.10 which reflects an average error gain[10] of refitting over pruning results for the 7 datasets of 15.0%.

Of course, during the refitting and backfitting tree stages the fuzzy tree complexity remains frozen, since the model structure is not adjusted anymore at this stage.

Table 4.10: Error gain (%) of refitting over pruning results

Gaussian GS=100	Twonorm GS=300	Omib-crisp GS=500	Waveform GS=300	Satellite GS=500	Pendigits GS=500	Dig44 GS=500	Avg. effect
4.9	22.2	38.3	10.7	5.4	10.6	13.2	15.0

Effect of backfitting

The improvement in accuracy of backfitting with respect to refitting results is quantified in table 4.11 as a percentage of 10.7% error gain [11] in average over the 7 datasets. In the case of twonorm, omib, pendigits and dig44 data, backfitting optimization clearly contribute to the accuracy improvement with respect to refitting. However, backfitting does not pay back the overload in CPU time in the case of gaussian, waveform and satellite datasets. Table 4.12 displays the increase in CPU time [12] for a backfitted FDT versus a refitted FDT and shows that a backfitted FDT needs 2 times more CPU time in average than a refitted FDT.

We will see further in Part II of experiments that differences in accuracy between backfitted and refitted FDTs are not statistically significant in average for "small" datasets.

[10] We computed the error gain (%) of refitting over pruning results as $\frac{Pe_{pruning} - Pe_{refitting}}{Pe_{pruning}} 100$.

[11] We computed the error gain (%) of backfitting over refitting as $\frac{Pe_{refitting} - Pe_{backfitting}}{Pe_{refitting}} 100$.

[12] We computed the increase in CPU time (%) for a backfitted FDT versus a refitted FDT as $\frac{CPU_{FDT(B)} - CPU_{FDT(R)}}{CPU_{FDT(R)}} 100$.

Table 4.11: Error gain (%) of backfitting over refitting results

Gaussian GS=100	Twonorm GS=300	Omib-crisp GS=500	Waveform GS=300	Satellite GS=500	Pendigits GS=500	Dig44 GS=500	Avg. effect
0.8	29.7	16.6	3.2	2.2	12.9	9.6	10.7

Table 4.12: CPU time increase (%) for a backfitted FDT versus a refitted FDT (on $GS + PS$)

Gaussian GS=100	Twonorm GS=300	Omib-crisp GS=500	Waveform GS=300	Satellite GS=500	Pendigits GS=500	Dig44 GS=500	Avg. effect
43	492	336	125	59	249	74	197

4.2.4 Global variance and bias study

Following the variance and bias estimation procedure with $q = 25$ learning algorithms, described in sections 2.1.4 and 3.7, we computed the two quantities as $variance$ and $bias^2 + residual\_error$, for CART and FDT models. Of course, mean squared error is the sum of the two quantities. So, what we will call next bias is in fact $bias^2 + residual\_error$. In this way, we may observe at least the relative evolution of this quantity, since $residual\_error$ is a constant term (residual error equals the last column $\sigma_C^2(\%)$ of table 4.1).

Comparing FDT bias-variance tradeoff with other methods tradeoff

Figure 4.3 shows the mean squared error (MSE) for all the methods, divided in its two parts: bias and variance (table D.6 from appendix corresponds to this graphic). Refitting and backfitting are carried out only on the GS, in order to not favor FDTs within comparison. Table 4.13 presents the error rates when refitting and backfitting on GS only, as a complement to the error results on refitting and backfitting on $GS + PS$ of tables 4.2 and 4.7.

We notice firstly, that variance of all versions of FDT, fully grown (F), pruned (P), refitted (R) or backfitted (B) is much smaller than variance of (pruned) CART regression trees and ULG decision trees. Secondly, when a fuzzy decision tree presents better accuracy than one or more of the aggregation methods, this is due generally to a lower prediction bias.

FDT pruning succeeds to leave constant or to slightly diminish bias or variance quantities, so as to obtain the same or a slightly improved mean squared error compared with the fully grown tree. Refitting and backfitting increase the variance of the pruned version, but reduce the bias.

Fuzzy versus crisp output definition in the bias-variance tradeoff. Figure 4.4 compares the bias-variance tradeoff between a fuzzy and a crisp output definition on the omib data (table D.5 in the appendix corresponds to this graphic). We may observe that all the quantities, MSE, variance

Table 4.13: Refitting and backfitting on GS error rates

Method	Gaussian	Twonorm	Omib fuzzy	Omib crisp
$q = 25$				
Ref. FDT (GS)	13.01	14.57	5.24	7.83
Backf. FDT (GS)	13.25	16.03	4.47	8.86

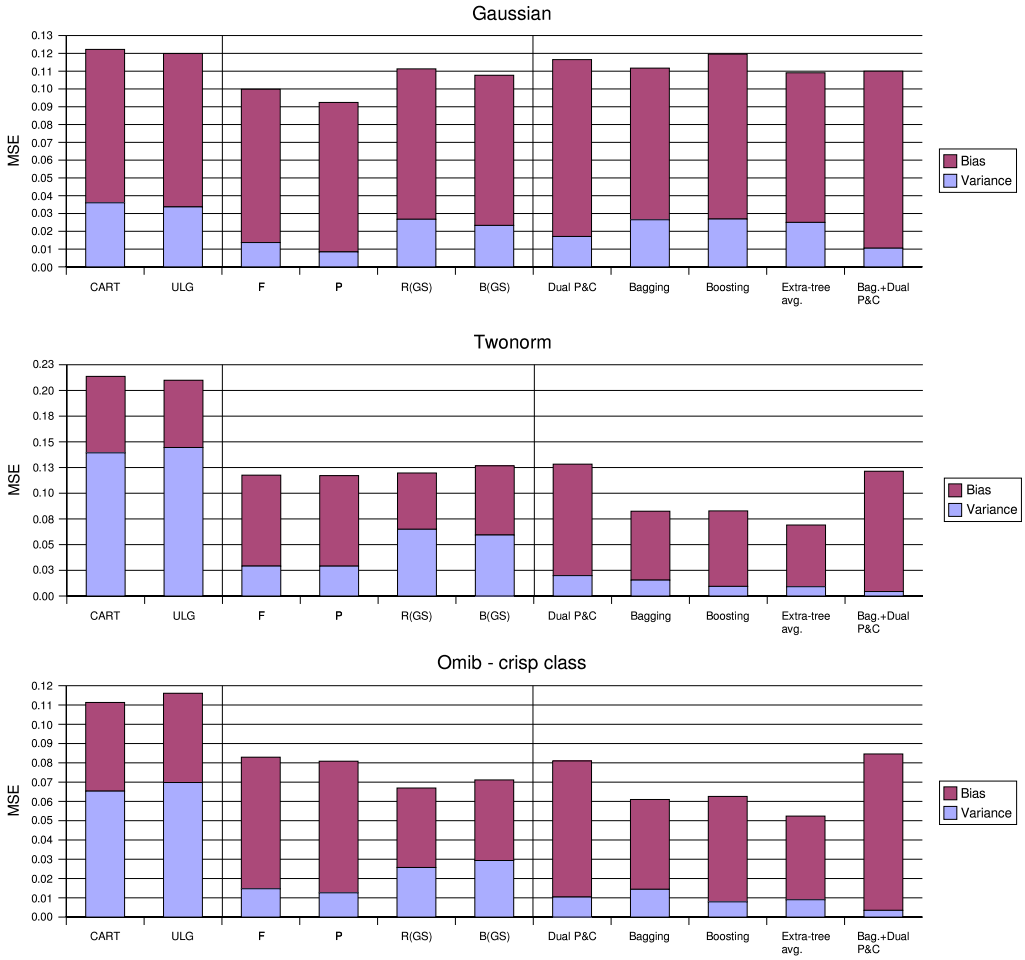


Figure 4.3: The proportion of variance and bias in the mean squared error

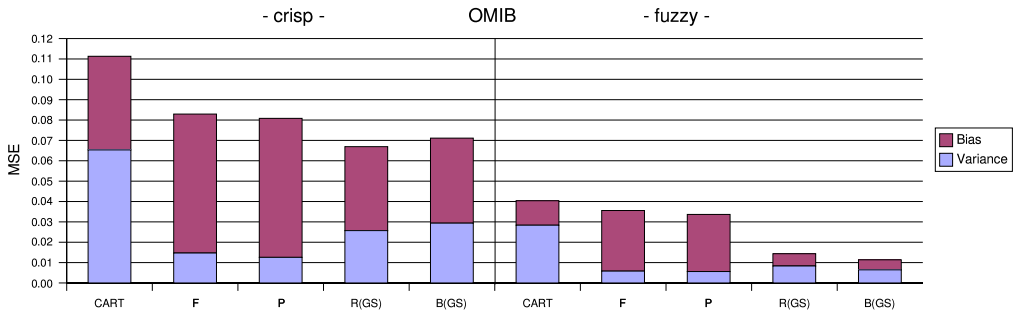


Figure 4.4: Crisp (left part) versus fuzzy (right part) class on FDT results and omib database

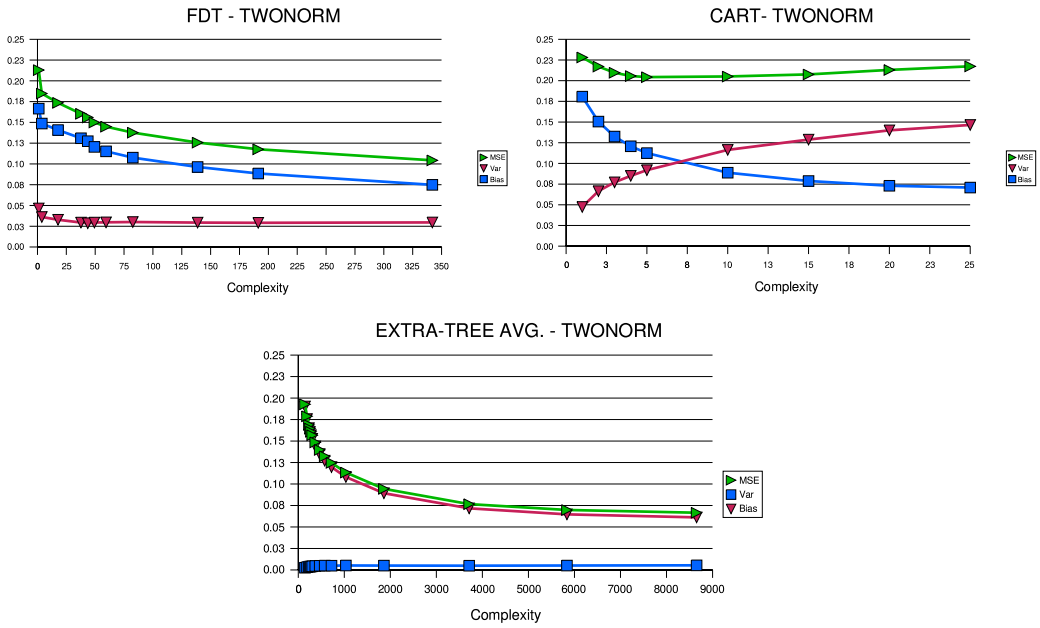


Figure 4.5: Evolution of MSE, variance and bias with the model complexity on twonorm dataset

and bias, are much decreased by the use of the fuzzy output, for all algorithms.

Piecewise linear versus sigmoidal discriminator. We also carried out comparisons of the bias-variance tradeoff between piecewise linear and sigmoidal discriminators, but differences are not significant (table D.7 from appendix shows this comparison).

In conclusion, fuzzy decision trees improve accuracy with respect to classical CART regression trees and ULG decision trees essentially by decreasing variance. FDTs do not reduce variance as much as aggregation methods do. The possibility to define a fuzzy output decreases both bias and variance of the fuzzy decision trees.

Variance and bias as function of tree complexity

Figures 4.5 and 4.6 display the evolution of MSE, variance and bias with the model complexity on twonorm and omib datasets, respectively. Tables D.8 and D.9 in section D.2 of the appendix display detailed results of those shown in these graphics.

In order to obtain such an evolution for a FDT, the experimental protocol was to use one of the stopping conditions: by changing the cardinality threshold (the maximum cardinality of a terminal node), we indirectly influence the tree complexity. Each complexity considered in these figures is an average complexity over the $q = 25$ trees, at a fixed cardinality threshold. For a CART regression tree, we used a best-first strategy of tree growing, and a threshold on the maximal complexity.

Figure 4.5 has one graphic for fully grown FDT, one for fully grown CART and one for extra-tree averaging [13] (100 extra-trees, $q = 25$). We observe firstly, the "classical" evolution of the bias and variance tradeoff for CART regression trees: bias decreases with the model complexity, variance increases with the model complexity and MSE shape presents an optimum of the bias-variance tradeoff, which gives also the optimal complexity for CART in this given problem.

[13] Dr. Geurts provided us with the graphic for extra-tree averaging.

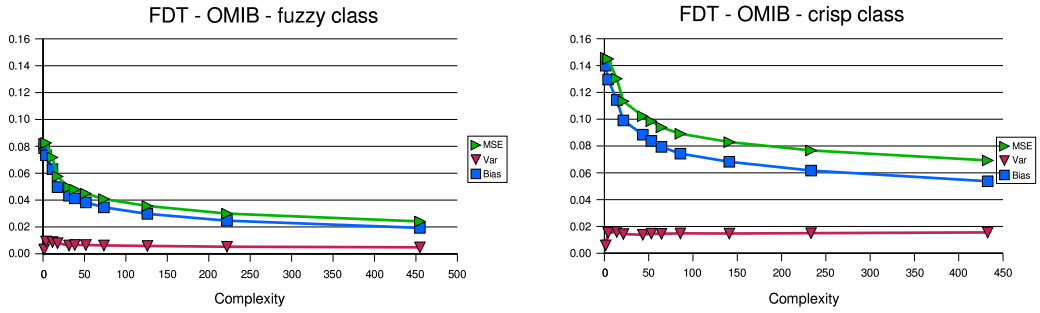


Figure 4.6: Evolution of MSE, variance and bias with the model complexity on omib dataset

This is a well known shape of bias-variance tradeoff in crisp trees, be it for classification of regression. Secondly, the evolution of the curves for the FDT method reveals a different picture. Bias evolution is not far from the CART bias evolution and decreases with the model complexity. But variance evolution has a different shape: variance decreases for small FDT models and then remains constant at larger model complexities. Variance amplitude is much lower than bias one. The effect of this two evolutions on MSE is that the error continues to decrease with the model complexity. Thus, the optimal MSE value is reached by the most complex FDT model. Thirdly, a similar evolution of the tradeoff has been evidenced by the extra-tree averaging method (see the bottom part of figure 4.5), where the variance remains constant with the model complexity and at very low level, even much lower than FDT variance (as figure 4.3 also shown it).

As variance for FDTs practically does not increase with the model complexity, this explains why the effect of pruning on FDT accuracy is mostly weak, since by pruning, variance remains the same and bias can only increase or remain constant. This is not the case of classical crisp trees, where a decrease in complexity may increase bias but reduce variance, so as the accuracy may even decrease.

Figures 4.6 repeats the same graphic for fully grown FDTs, for both fuzzy and crisp class outputs. At crisp class, omib evolution follows the same ideas than twonorm evolution, with the sole difference that variance increases at small complexities instead to decrease as for twonorm data. At fuzzy class, again variance remains constant and bias decreases, moreover, *both* variance and bias levels are lower. This confirms one more that the fuzzy definition of the output has a positive effect on both bias and variance.

Note that the evolution of the analysed tradeoffs changes if the FDT is after pruning, refitting or backfitting steps.

The behaviors we put in evidence here are maybe linked to only some of the datasets not to all of them. What we may add up is: we noticed (for twonorm and omib datasets) that if a FDT is not constrained by the stopping conditions (which are let as much relaxed as possible), it continues to develop itself and it does not ever stop. This because of the overlapping of the sets of objects in the left and right successors, which may determine at infinite the same objects in left and right successors of a current node as in the current node itself and even a perfect overlap. Thus, determining that a same cliche to run off continuously doing the same splitting, but increasing infinitely the complexity. We think that this is the same phenomenon as for aggregation techniques: *the larger the model complexity, the better the accuracy*.

Variance and bias as function of growing set size

Figure 4.7 display the evolution of MSE, variance and bias with the growing set size on omib dataset and backfitted (on GS) version of FDTs. Table D.10 in section D.2 of the appendix display

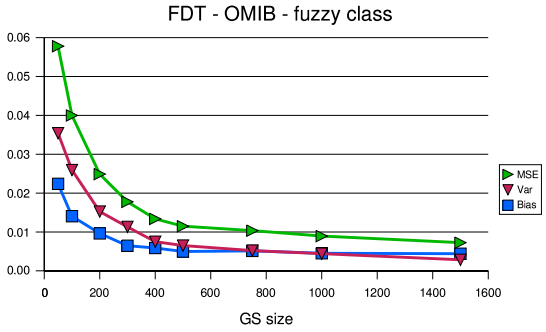


Figure 4.7: Evolution of MSE, variance and bias with the growing set size on omib dataset: backfitted on GS FDT

detailed results of those shown in this graphic. As expected, both bias and variance decrease with the growing set size. In this case, they are almost constant and equal quantities for medium and large sample sizes.

We did also an experiment where the structure of the FDT is fixed and the complexity is settled to 3 test nodes. The structure has been determined with an ULG decision tree built with the whole dataset in each case. The evolution of MSE, variance and bias with the growing set size of the backfitted FDT, for omib data (crisp and fuzzy output) and for gaussian data is displayed in figure 4.8. We remark here a less pronounced decrease of the error and variance with the growing set size and an almost constant bias. Also, bias is always the dominant source of error. This is normal, since by forcing the structure and the model complexity, we introduced bias. This is in concordance with the evolution of MSE, variance and bias quantities with the model complexity of figures 4.5 and 4.6, where bias is the principal source of error, especially at small model complexities. The refitted version of the FDTs gives the same allure of the curves as the backfitted FDT for all the datasets investigated.

4.2.5 Parameter variance and bias study

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. We sketched in section 2.1.5 the principal sources of variance in a tree-based method. Refs. [GOW01, GW00, Weh97] show that classical discretization methods of decision trees actually lead to very high variance of the cut point parameter, even for large learning sample sizes, and also that, there is a large variability of the decision tree structure, visible principally through the variation of the chosen attribute in a particular node. Our simulations reveal that the same two behaviors are encountered at backfitted FDT.

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 25 trees built on a small growing set of size 50, both ULG and FDT have chosen 19 times the same Pu attribute out of the 6 attributes in the root node. Thus this structure variability depends 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, we have done 2 studies: we computed variance (standard deviation) of the cutting point α parameter

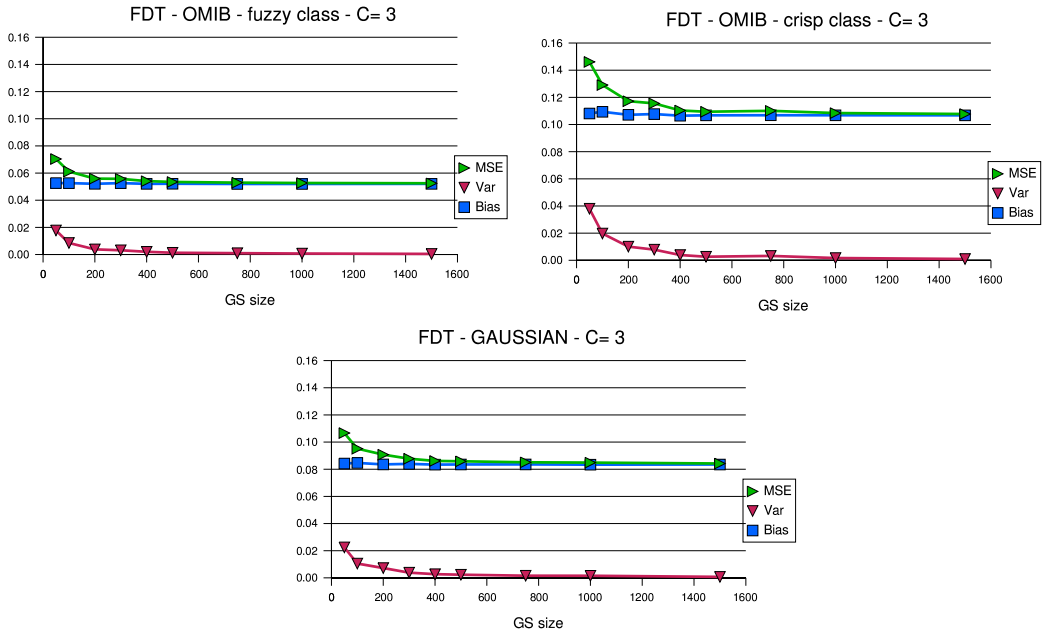


Figure 4.8: Evolution of MSE, variance and bias with the growing set size, at fixed structure with $C = 3$: backfitted on GS FDT

in the *root node* σ_α, together with its bias b_α, and we computed the same quantities in the *second level* of test nodes of the tree (depth 2), respectively, in the left and right successors of the root node. Both experiments have been done for ULG, CART and FDT methods, by following the parameter variance and bias estimation procedure described in sections 2.1.4 and 3.7.

Parameter variance and bias at the root node

The procedure is like follows. We build q trees (CART, FDT or ULG). For each of these trees we keep the α cutting point parameter of the root node of the tree given that it corresponds to the same attribute. Thus we obtain at most q parameters, since not necessarily all the trees will choose the same attribute in the root node. We compute statistics on these parameters: mean value, standard deviation (σ_α) and bias (b_α). To obtain the bias we compute the difference between an asymptotic value of the α parameter and the just computed mean value. The asymptotic value of the α threshold in the root node is determined a priori by an ULG decision tree built on the whole available dataset for each database.

Table 4.14 presents the variance σ_α of the cutting point α parameter in the root node and its bias b_α, for ULG, CART, refitted (R), backfitted on GS (B(GS)) and backfitted on $GS + PS$ (B(GS+PS)) FDTs for omib fuzzy output and gaussian datasets. These results are all linked to the results already discussed so far where $q = 25$ (thus to tables 4.2, 4.4 and D.6). For each database, the asymptotic value (e.g. $\text{Pu}_{asymptotic} = 1057$) and the standard deviation of the attribute (e.g. $\sigma_{Pu} = 171$) are displayed. For each case, the number of appearances of the selected attribute in the root node is displayed: for omib database, 25 times out of 25 trees we had attribute Pu in root node (we noted 25/25 in the table), whereas for gaussian database, 16 times out of 25 trees we obtained attribute $A1$ in root node (16/25).

To enrich the results, we did also a study of the bias-variance evolution for the parameter α in the root node with the growing set size, for omib dataset which is shown in table 4.15. Here we made an average over $q = 20$ trees. For these data, trees have chosen naturally attribute Pu in the root node. Figure 4.9 displays the evolution of the parameter variance in the root node with the

Table 4.14: Bias-variance tradeoff for the cutting point parameter α

class	σ_α					b_α				
	ULG	CART	R	B(GS)	B(GS+PS)	ULG	CART	R	B(GS)	B(GS+PS)
Omib, GS=500, $\text{Pu}_{asymptotic} = 1057, \sigma_{Pu} = 171$, 25/25										
fuzzy	55	38	38	42	46	-8	3	3	15	36
crisp	55	44	44	43	47	-8	-20	-20	15	4
Gaussian, GS=100, $\text{A1}_{asymptotic} = 1.23, \sigma_{A1} = 1.46$, 16/25										
crisp	0.47	0.27	0.27	0.43	0.45	0.06	0.39	0.39	0.50	0.48

Table 4.15: Bias-variance tradeoff evolution for the cutting point parameter α with the growing set size, omib data, $q = 20$.

GS size	$\sigma_\alpha\,[MW]$				$b_\alpha\,[MW]$			
	ULG	CART	R	B(GS)	ULG	CART	R	B(GS)
Omib fuzzy, $\text{Pu}_{asymptotic} = 1057, \sigma_{Pu} = 171$, 20/20								
100	83	52	52	56	-10	4	4	2
200	79	46	46	49	-13	-11	-11	-7
300	62	36	36	46	-12	-13	-13	-13
400	57	42	42	46	1	-2	-2	11
500	40	35	35	27	-1	-8	-8	6
750	59	41	41	51	-19	7	7	30
1000	34	23	23	35	7	-3	-3	15
1500	25	18	18	40	9	8	8	42

growing set size, using data of the table 4.15, for ULG, refitted FDT and CART (R) and backfitted FDT (B) methods.

Due to the adopted fuzzy partitioning approach, the chosen attribute in the root node of a non-backfitted fuzzy 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 fuzzy decision trees in the root node (see columns CART and R of tables 4.14 and 4.15). Once backfitted, a fuzzy decision tree changes its thresholds in all the tree nodes, and thus also its parameter variance and bias (see columns B(GS) and B(GS+PS) of table 4.14 and columns B(GS) of table 4.15).

One may observe that a non-backfitted fuzzy decision tree, identically to a CART regression

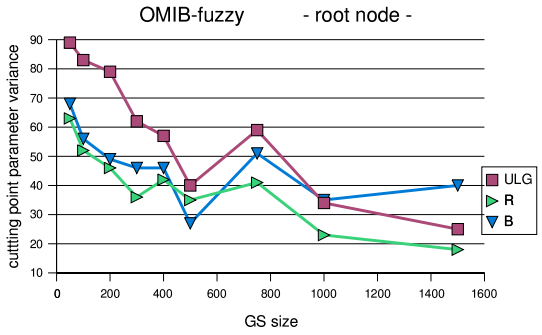


Figure 4.9: Evolution of the parameter variance with the growing set size at the root node

tree, presents less parameter variance in the root node (about 31% in average for omib and 43% for gaussian), and as much parameter bias in average as an ULG decision tree. By backfitting, parameter variance in a root node of a FDT increases with respect to the non-backfitted version, as figure 4.9 shows it. 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.

Parameter variance at the second level test nodes

By studying the parameter variance and bias in the root node of a fuzzy decision tree, we are not able to see the contribution of the fuzzy approach of splitting to the parameter variance and bias in a whatever internal test node. Thus, we did also an experiment where the structure of the FDT is fixed and the complexity is settled to 3 test nodes. The structure has been determined with an ULG decision tree built with the whole dataset in each case. We did the study for omib fuzzy output, omib crisp output and gaussian data. We watched also the evolution of the parameter variance with the growing set size. Figures 4.10 (omib fuzzy class data), 4.11 (omib crisp class data) and 4.12 (gaussian data) display the evolution of parameter variance at both the second level nodes (depth 2): right and left (tables D.11 (omib crisp), D.12 (omib fuzzy) and D.13 (gaussian) from appendix correspond to these graphics). In the omib case, the ULG method has chosen attribute Pu in root node and attribute Qu in both its successors. In the gaussian case, in the root node and in its "yes" successor (corresponding to the "left" successor in a FDT) attribute A1 has been chosen and in the "no" ("right") successor, attribute A2 has been chosen.

We may notice from these graphics, firstly, a decrease of the parameter variance with the growing set size for omib data, exactly as for the root node parameter variance of figure 4.9 and almost no decrease for gaussian data. One may conclude that this decreasing of the parameter variance with the growing set size depends on the problem and data analyzed. Secondly, CART, refitted FDT and backfitted FDT do not seem to show a significant difference in their parameter variance at the second level nodes, but they are definitely all more stable from this point of view than the ULG method, with a single exception, left node in gaussian problem, where backfitted FDT has the worse parameter variance among the four methods.

The conclusion of this study is that the reduction of prediction variance of FDTs (with respect to CART) is not correlated with a reduction of parameter variance (with respect to CART). Thus, the fuzzy partitioning does not improve the parameter stability with respect to a crisp partitioning. On the other hand, a by product of our study is that we found that parameter variance of CART (and also FDTs) is smaller than that of ULG.

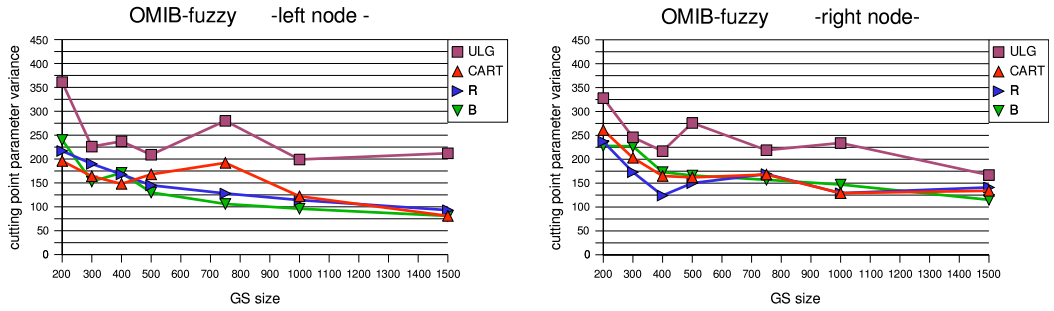


Figure 4.10: Evolution of the parameter variance with the growing set size at the second level nodes of a fuzzy decision tree (depth 2), for omib fuzzy output database

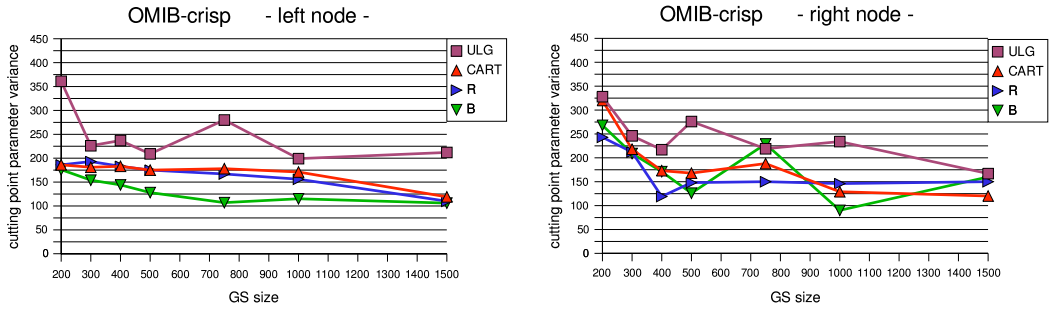


Figure 4.11: Evolution of the parameter variance with the growing set size at the second level nodes of a fuzzy decision tree (depth 2), for omib crisp output database

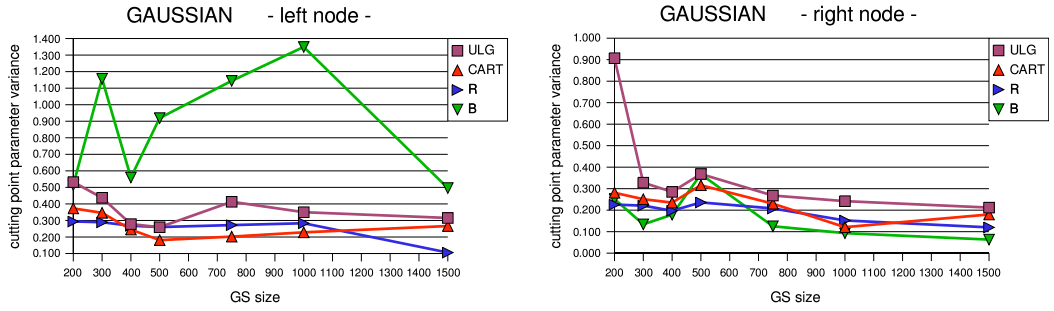


Figure 4.12: Evolution of the parameter variance with the growing set size at the second level nodes of a fuzzy decision tree (depth 2), for gaussian database

4.2.6 Various analyses of FDT behavior

Evolution of results function of the growing set size

Tables D.14 (gaussian), D.17 (twonorm), D.21 (omib), D.26 (waveform), D.27 (satellite), D.28 (pendigits) and D.29 (dig44) from section D.4 of the appendix give results for averages of $q = 5$ FDTs. For each database a number of 4 sizes of GS samples has been investigated.

Figure 4.13 shows the evolution of the FDT complexities after pruning with the growing set size for all 7 datasets. Here complexities for multi-class problems are the average of number of test nodes over the forest. Generally, complexities increase with the growing set size, more at small sizes, and less at larger sizes where they seem to stabilize. Figure 4.14 shows the evolution of the FDT error rates (after pruning in the upper graphic, after backfitting on GS+PS in the lower graphic) with the growing set size for all 7 datasets. Except for gaussian data whose evolution is almost constant, all the error rates are improving with the growing set size, as expected. This phenomenon of error rate decrease is more accentuated for pruned versions of FDT than for backfitted versions. This because, at any GS size, backfitting on GS+PS improves a lot the error, thus also at small GS sizes and this makes more uniform the evolution of the error after backfitting.

We also remark from the appendix tables that excepting pendigits data, FDTs built for medium growing sets (of 300 or 500 examples) give similar results or sometimes more accurate results than C4.5, ULG or CART built for the largest growing sets (of 2000 objects). This statement is true for the majority of refitting and backfitting on GS+PS results, but also for fully grown and pruned versions of trees in the case of gaussian, twonorm and waveform datasets.

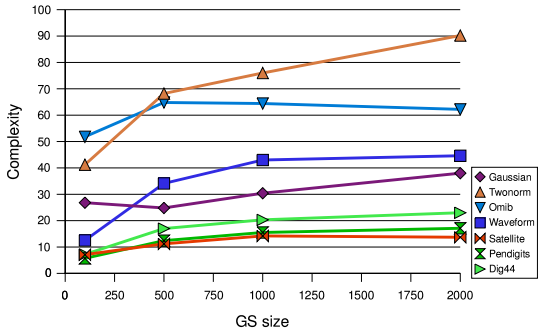


Figure 4.13: FDT complexities function of growing set size

Effect of refitting and backfitting on FDT parameters

By refitting a FDT, the only parameters of the model that change are the terminal nodes labels. We note by ΔL the quantification of the change in all the labels of a FDT between before and after refitting. By backfitting a FDT, all the model parameters change, thus, we note also by $\Delta\alpha$ the global change in value of the cutting point in each test node of the tree, and by $\Delta\beta$ the global change in value of the fuzziness degree in every test node, both changes quantified between the pruned (non-refitted) FDT and the backfitted FDT. Table 4.16 displays these quantities for twonorm and omib datasets [14].

We may observe that terminal nodes labels change a lot with refitting for omib data and less for twonorm. This is in concordance with the fact that error rates change a lot with refitting for omib and less impressive for twonorm. By backfitting, for omib, the parameters change is

[14] $\Delta param = \frac{1}{P}\sqrt{\sum_{i=1}^{P}(param_{\text{i, before}} - param_{\text{i,after}})^2}$, where P is the number of the parameters $param$ (the model complexity K in the case of α and β and $K+1$ in the case of L).

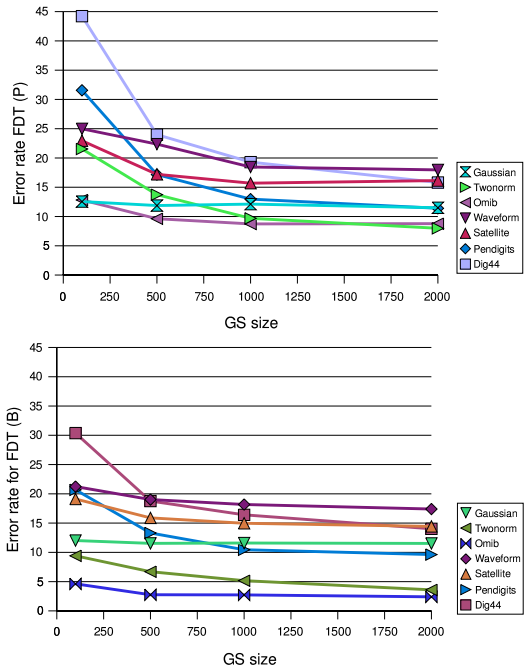


Figure 4.14: FDT error rates function of growing set size

distributed between $\Delta\alpha$, $\Delta\beta$ and ΔL and all these quantities are smaller then ΔL obtained by refitting. By backfitting, for twonorm, terminal nodes labels change more than by refitting. This is in concordance with the fact that error rates change a lot with backfitting for twonorm.

Generally, by backfitting, α displays the smallest change, then β follows, and terminal nodes present the most important change of all. This may explain why refitting as an optimization stage is so effective and comes near backfitting results: because α and β parameters do not need much change, and only by optimizing labels in terminal nodes we already get the wanted effect over the accuracy or at least a very good one.

We should make the remark that during backfitting, certain β values in the tree may become negative values. Suppose we have $\alpha \in (0,1)$ and $\beta_1 > 0$, which corresponds to a (piecewise linear or sigmoidal) discriminator $\nu(\alpha, \beta_1)$. When a negative β intervene, at $\alpha \in (0,1)$ and $\beta_2 = -\beta_1 < 0$ it corresponds discriminator $\nu(\alpha, \beta_2) = 1 - \nu(\alpha, \beta_1)$. Therefore, when in a test node, its discriminator presents a negative β value, we may switch its successors in order to obtain a discriminator with a positive β value and to preserve the interpretability of the tree.

Choice of algorithm parameters and their influence on the results

An important aspect of the learning algorithms is their autonomy. The results reported in this chapter are obtained for fuzzy decision trees using a single set of algorithm parameters, thus the same parameters for all datasets. We did not optimize by trial and error the results on each database by "playing" with the algorithm parameters. In this way, the autonomy of FDT is preserved: FDTs do not depend on a human expert in order to choose the best model parameters or complexity for obtaining an optimum result for each problem to be analysed.

The parameters for the stopping conditions of the FDT 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}.

Table 4.16: Effect of refitting and backfitting on FDT parameters

	refitting	backfitting		
GS size	ΔL	$\Delta\alpha$	$\Delta\beta$	ΔL
omib, fuzzy class, q = 5				
100	0.5824±0.4286	0.0528±0.0185	0.0993±0.0306	0.2490±0.1083
500	0.6006±0.1674	0.0614±0.0157	0.1311±0.0668	0.2353±0.0589
1000	0.4242±0.1550	0.0539±0.0192	0.0815±0.0245	0.2361±0.0509
2000	0.4817±0.1227	0.0524±0.0191	0.0986±0.0553	0.3155±0.2611
twonorm, q = 5				
100	0.7948±0.7022	0.0884±0.0374	0.1966±0.1183	0.4779±0.1367
300	0.1565±0.0516	0.0556±0.0358	0.1002±0.0547	0.2021±0.0911
1000	0.1127±0.0151	0.0316±0.0215	0.0672±0.0208	0.2786±0.0835
2000	0.1002±0.0303	0.0638±0.0158	0.1149±0.0255	0.3572±0.0948

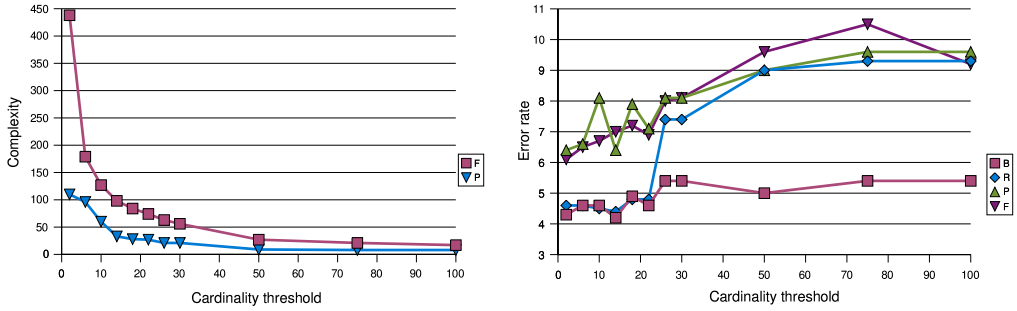


Figure 4.15: How accuracy and complexity differ with cardinality threshold parameter for OMIB fuzzy output, $||GS|| = 2000$.

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. Figure 4.15 shows the influence of this cardinality threshold parameter on accuracy and complexity, at fixed GS, PS, TS sets, with $||GS|| = 1000$ objects, for the fully grown (F), pruned (P), refitted (R) and backfitted (B) versions of the FDT 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. If we diminish the two thresholds we obtain more complex trees after growing, with an increase in CPU time, but after pruning we get the same pruned trees. Thus, we get useless larger trees at the beginning.

The number of evaluations in FIBONACCI search is 5. The parameters of the LEVENBERG-MARQUARDT algorithm are : 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 ΔE_{min}, 50 for the maximum allowed number of cycles cy_{max}. Increasing the number of evaluations in FIBONACCI search or decreasing ΔE_{min} in LEVENBERG-MARQUARDT algorithm would produce slower algorithms with only slightly improved results, whereas the opposite case could make the algorithms sub-optimal.

Table 4.17 displays the average number of cycles necessary to backfit the FDT. By cycle we denote every passing through the forward step of the optimization procedure. We observe that the backfitting does not need many iterations to converge toward a satisfactory solution. For gaussian

Table 4.17: The average number of cycles necessary to backfit the FDT for 3 datasets and $q = 5$

GS size	Gaussian	Omib-fuzzy	GS size	Twonorm
100	29	21	100	50
500	36	23	300	45
1000	32	23	1000	48
2000	30	22	2000	39

and omib data, the average number of cycles is much less than the imposed threshold $cy_{max} = 50$, whereas for twonorm data, among the 5 trees averages, there are some for which backfitting process stops because 50 cycles are done, not because the threshold for the error gain ΔE_{min} has been reached. To see what happens for this twonorm database if we settle $cy_{max} = 100$, we made the experiment of table 4.18. We observe that MSE and error rates are decreasing insignificantly whereas the CPU time increases sometimes significantly.

Table 4.18: Comparison in terms of cy_{max} parameter for twonorm dataset

GS size	Method	cy_{max}	Error rate (%)	MSE	$\bar{cy}$	total CPU time
q = 5						
100	Backf. FDT	50	9.41±2.30	0.085790±0.011055	50	14m
	Backf. FDT	100	9.20±2.28	0.084554±0.011208	65	17m
300	Backf. FDT	50	6.70±1.28	0.069981±0.004581	45	33m
	Backf. FDT	100	6.68±1.26	0.069650±0.004527	75	53m
1000	Backf. FDT	50	5.16±0.58	0.060791±0.003332	48	1h07
	Backf. FDT	100	5.14±0.51	0.060456±0.002814	57	1h18
2000	Backf. FDT	50	3.59±0.78	0.054419±0.001710	39	1h53
	Backf. FDT	100	3.59±0.78	0.054419±0.001710	39	1h53
q = 25						
300	Backf. FDT	50	7.94±1.59	0.075440±0.007156	45	2h57
	Backf. FDT	100	7.77±1.52	0.074828±0.006942	61	3h46

In order to see that a number of 5 evaluations in FIBONACCI search is enough in FDT growing, let us take a little problem of estimating the fuzzy class of omib based on the attribute CCT itself only, with $GS = 2000$, $PS = 2000$, $TS = 1000$ and one single test node as complexity. Table 4.19 shows α and β parameters obtained if the number of evaluations in FIBONACCI search is 5 or 10, and then optimized by backfitting, together with the obtained error rate and MSE computed on TS. We may observe that even if the mean squared error is improved for 10 evaluations, the error rate remains constant, thus only 5 evaluations on a set of 2000 objects is enough. It is true that with 10 evaluations the obtained β value is nearer of the real value (obtained by backfitting) than the one obtained with 5 evaluations, but the necessary CPU time is also 2 times larger.

We have tried to select a tree at pruning stage by a multiplier $\sigma = 0$. Results were slightly the same, slightly worse for larger GS and slightly better for smaller GS.

We have equally tried to select a tree at pruning stage by error rate Pe itself instead of MAE. We obtained comparable error rates for trees and more complex trees.

Piecewise linear versus sigmoidal discriminators

Table 4.20 makes a summarized comparison between results obtained with piecewise linear and sigmoidal discriminators, for different GS sizes. They are extracted from tables D.14 and D.15

Table 4.19: Comparison between 5 and 10 evaluations in FIBONACCI search

Method	α	β	$Pe(\%)$	MSE
5 evaluations	0.35397	0.26548	1.6	0.001353
10 evaluations	0.35397	0.23068	1.6	0.000998
after backfitting	0.33631	0.23976	0.0	0.000000

for gaussian data, tables D.17 and D.18 for twonorm data, tables D.21 and D.22 for omib data that may be found in section D.4 of the appendix. Overall, piecewise linear discriminator results are better than sigmoidal discriminator results. Sigmoidal discriminator models are slightly less complex. They are less accurate, in terms of both error rate and MSE, especially for twonorm and omib datasets.

Table 4.21 displays the CPU time for piecewise linear and sigmoidal discriminators (for multi-class problems is the time to build the whole forest of CART and FDT), for building a fully grown FDT. The CPU times are generally more important in the case of sigmoidal discriminator.

Variant for Fibonacci search

In section 3.8.1 of chapter 3 we presented a variant of the FIBONACCI search, which for "large" nodes performs 10 evaluations of the function to be minimized whereas for "small" nodes performs only 5 evaluations. Tables D.19 (twonorm) and D.24 (omib) in appendix display results of this variant. In the variant used so far (we will call it the normal one), 5 evaluations are done no matter the node size. Table 4.22 makes a summarized comparison between results obtained with the normal FIBONACCI search and results obtained with the variant, for $GS = 2000$ objects. We may observe that even if for fully grown and pruned versions, the variant for FIBONACCI search slightly improves the results in accuracy, the overall effect is insignificant, and backfitting results are slightly depreciated.

Variant for pruning

In section 3.8.2 of chapter 3 we presented a variant for pruning. Tables D.16 (gaussian) and D.23 (omib) in appendix display results of this variant. Table 4.23 makes a summarized comparison between pruned FDT results obtained with normal pruning and variant for pruning, for different GS sizes. The conclusion is that, at small growing sets the variant for pruning gives slightly better results after pruning (also after refitting and backfitting), but at large growing sets gives slightly less accurate results after pruning (also after refitting and backfitting). Overall, the two pruning variants give similar results.

Refitting during pruning

In section 3.8.3 of chapter 3 we presented a refitting during pruning approach. Tables D.20 (twonorm) and D.25 (omib) in appendix display results of this variant. Table 4.24 shows a summarized comparison between results obtained with normal pruning and with the variant of refitting during pruning, for $GS = 2000$ objects. We may see that pruned and refitted trees have close accuracies in the refitting during pruning variant, that both accuracies are improved with respect to the normal pruning variant and that they are not far from the backfitted tree accuracy. For omib data there is a clear improvement on all pruned, refitted and backfitted versions of FDT which comes also with an increase in the model complexity.

Table 4.20: Comparison between backfitted FDT obtained with piecewise linear and sigmoidal discriminators, $q = 5$

Database	GS size	Discr.	Complexity	$Pe(\%)$	MSE
Gaussian	100	pw	26.8	12.02	0.087480
		sig	25.2	11.61	0.084464
	500	pw	24.8	11.53	0.083682
		sig	26.6	11.56	0.083767
	1000	pw	30.4	11.59	0.084224
		sig	10.6	11.62	0.084137
	2000	pw	38.0	11.52	0.083661
		sig	24.8	11.53	0.083565
Twonorm	100	pw	41.2	9.41	0.085790
		sig	27.8	11.51	0.096017
	300	pw	68.2	6.70	0.069981
		sig	73.2	9.07	0.081849
	1000	pw	76.0	5.16	0.060791
		sig	87.2	8.41	0.072847
	2000	pw	90.2	3.59	0.054419
		sig	84.4	5.01	0.059127
Omib fuzzy	100	pw	51.8	4.62	0.014460
		sig	47.2	6.74	0.018126
	500	pw	64.8	2.76	0.006151
		sig	63.8	3.33	0.006940
	1000	pw	64.4	2.73	0.005746
		sig	47.6	3.86	0.009757
	2000	pw	62.2	2.40	0.005519
		sig	58.8	4.03	0.008783

Thus, this variant of refitting during pruning approach may be valuable since with only two stages, growing and pruning, we may reach a good accuracy, close to the one given by refitting stage. This could be a promising alternative provided that we succeed to improve the computational complexity of this approach, which for the moment is very costly in CPU time, a lot more costly than the normal pruning approach.

Table 4.21: A single typical run CPU time (in seconds) comparison between piecewise linear and sigmoidal discriminator of full FDTs on 7 databases

Discriminator	Gaussian GS=100	Twonorm GS=300	Omib GS=500	Waveform GS=300	Satellite GS=500	Pendigits GS=500	Dig44 GS=500
Piecewise linear	32	54	41	44	107	95	292
Sigmoidal	37	62	76	92	199	114	287

Table 4.22: Comparison between normal Fibonacci search and its variant, $q = 5$, $||GS|| = 2000$

Database	Method	Variant	Complexity	$Pe(\%)$	MSE
Twonorm	Full FDT	normal	239.6	7.38	0.121386
		variant	240.0	7.05	0.121595
	Pruned FDT	normal	90.2	7.98	0.115621
		variant	82.0	7.40	0.114333
	Refitted FDT	normal	90.2	8.92	0.064139
		variant	82.0	6.14	0.063949
	Backfitted FDT	normal	90.2	3.59	0.054419
		variant	82.0	3.72	0.054027
Omib fuzzy	Full FDT	normal	132.0	7.77	0.035325
		variant	137.5	7.39	0.034388
	Pruned FDT	normal	62.2	8.78	0.032424
		variant	63.4	8.83	0.032518
	Refitted FDT	normal	62.2	3.68	0.009506
		variant	63.4	3.92	0.010054
	Backfitted FDT	normal	62.2	2.40	0.005519
		variant	63.4	2.55	0.005433

Table 4.23: Comparison between pruned FDT obtained with normal pruning and with pruning variant, $q = 5$

Database	GS size	Variant	Complexity	$Pe(\%)$	MSE
Gaussian	100	normal	26.8	12.55	0.092943
		variant	23.2	12.13	0.090426
	500	normal	24.8	11.87	0.086268
		variant	34.2	12.09	0.087921
	1000	normal	30.4	12.12	0.086891
		variant	32.4	12.25	0.088639
	2000	normal	38.0	11.49	0.086229
		variant	26.2	11.76	0.086387
Omib fuzzy	100	normal	51.8	12.82	0.043770
		variant	69.8	12.36	0.043955
	500	normal	64.8	9.62	0.032965
		variant	60.0	8.87	0.032232
	1000	normal	64.4	8.73	0.032032
		variant	55.6	8.63	0.031760
	2000	normal	62.2	8.78	0.032424
		variant	49.2	9.38	0.033765

Table 4.24: Comparison between normal pruning and refitting during pruning variant, $q = 5$, $||GS|| = 2000$

Database	Method	Variant	Complexity	$Pe(\%)$	MSE
Twonorm	Full FDT	normal	239.6	7.38	0.121386
		variant	239.6	7.38	0.121386
	Pruned FDT	normal	90.2	7.98	0.115621
		variant	93.2	5.91	0.064054
	Refitted FDT	normal	90.2	8.92	0.064139
		variant	93.2	5.74	0.063158
	Backfitted FDT	normal	90.2	3.59	0.054419
		variant	93.2	4.29	0.055281
Omib fuzzy	Full FDT	normal	132.0	7.77	0.035325
		variant	132.0	7.77	0.035325
	Pruned FDT	normal	62.2	8.78	0.032424
		variant	112.0	2.94	0.002591
	Refitted FDT	normal	62.2	3.68	0.009506
		variant	112.0	2.81	0.006202
	Backfitted FDT	normal	62.2	2.40	0.005519
		variant	112.0	1.90	0.003713

4.3 Part II

4.3.1 The experimental methodology

Datasets used

The second part of experiments is designed to compare the predictive accuracy of fuzzy decision trees with respect to C4.5 and ULG decision trees and CART regression trees, and to see if the FDT gain in accuracy is statistically significant with respect to these standard crisp tree-methods. We have chosen in this purpose 11 datasets from the UCI Machine Learning Repository, well known standard datasets. They are summarized in table 4.25. Section B.2 of the appendix gives details concerning these datasets. All of them were previously used in other comparative studies. Their attributes are all numerical with no missing values. They are rather small datasets, with no more than a few hundred objects.

Table 4.25: 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

Protocol of experiments

The adopted strategy was to run *ten* times 10-fold non-stratified cross-validation tests and to apply paired two-sided t-test in order to verify the statistical significance of the differences in the error rate between FDT, C4.5, ULG and CART algorithms. This protocol has been explained in section 2.1.7 of chapter 2.

Pruning has been done by all methods directly on the growing set ($PS = GS$). Also refitting and backfitting has been done for FDT only on the growing set $RS = BS = GS$. Each method has been built and tested on identical pairs (GS,TS).

4.3.2 Discussion of results

Table 4.26 shows comparatively accuracy of C4.5 decision trees, ULG decision trees and CART regression trees, and refitting (R) and backfitting (B) versions of our FDT method. For each pair 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 with significance set at the 5% level. Relative to each algorithm and according to this significance test, a $+(-)$ sign in the left part of the column means that the error rate of this algorithm is significantly better (worse) than the error rate of the refitting (R) FDT, whereas a $+(-)$ sign in the

Table 4.26: Classification error rates and standard deviations for a 10 times 10-fold cross-validation

Database	C4.5	ULG	CART	FDT (R)	FDT (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

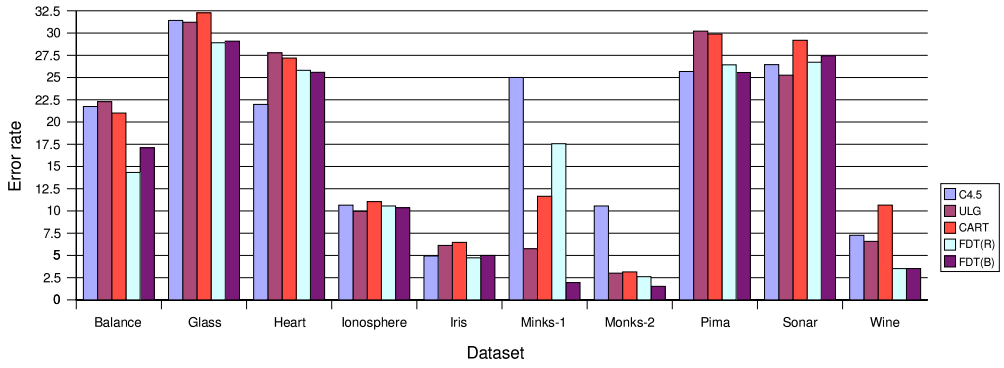


Figure 4.16: Comparison of methods in terms of error rates

right part of the column means that the error rate of this algorithm is significantly better (worse) than the error rate of the backfitted (B) FDT.

Figure 4.16 displays the error rates of the 5 methods and all datasets excepting monks-3 data where the errors are null everywhere.

Since databases are small this time, there is a risk of overfitting, which can be seen on certain datasets (balance, glass, iris, sonar) where backfitting slightly worsens accuracy with respect to refitting. In spite of this, both refitting and backfitting FDTs offer the best results overall. This time, refitting and backfitting results $RS = BS = GS$ are not anymore influenced by the fact that they use a learning set larger than the growing set, which was the case in Part I of this chapter where refitting and backfitting sets where $RS = BS = GS + PS$.

Table 4.27 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 fuzzy 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. According to the same significance test, backfitted FDT is significantly better than refitted FDT for 3 datasets (monks-1, monks-2 and pima) and significantly worse for one dataset (balance). 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

Table 4.27: 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
FDT (R)	5/1	6/1	7/1
FDT (B)	5/1	7/1	9/0

C4.5, CART or ULG ones.

4.4 FDT visualization

An important aspect of an interpretable learning method is the ability to visualize the model. The same as standard crisp decision or regression trees, our fuzzy decision tree method can be displayed. By visualizing the FDT one can observe how complex is the model, the attributes preferred by the model in its near-by-the-root nodes and generally all the attributes that contribute to the final answer for a given new instance dropped in the root node of the tree.

The software that builds fuzzy decision trees includes also a module for fuzzy tree visualization. We give here in the next figures of this chapter a few examples of fuzzy decision trees built in the context of previous validation studies. All the visualized trees are built exactly in the same conditions as presented in the two protocols of experiments of part I and II. All the attributes are normalized into the [0;1] interval (that is why they are called *norm-att*).

Figures from 4.17 to 4.19 constitute an example of how the FDT looks like in each of the intermediary stages: a pruned tree (figure 4.17), the refitted version (figure 4.18) and finally the backfitted model (figure 4.19). This example is done on omib fuzzy data, on a growing set of 500 objects, pruning set of 2000 objects, refitting and backfitting sets of 2500 objects and test set of 4000 objects. Model complexity, error rate and mean squared error are indicated for each tree in the figure caption.

The FDT trees should be "read" as already explained in section 3.1.2. Each rectangle corresponds to a node. Above the root node is marked how many test nodes, leaves and deadends contains the tree. For example, for pruned FDT of figure 4.17 we have T10+L8+D3 which means that there are 10 test nodes, 8 leaves and 3 deadends. Leaf and deadends are two types of terminal nodes. A leaf (L) is obtained if the splitting is stopped based on the first two stop splitting conditions explained in section 3.4.7 of chapter 3. A deadend (D) is obtained based on the third condition. After pruning, a pruned test node keeps its name (T) and it counts for a deadend.

Each node contains the label of the node, i.e the local estimation of the output. The test in a test node is formed by the attribute located under each test node together with the two conditions marked on the two branches that come out from this node and go to the left and right successor nodes. For example, the root node of figure 4.19 has chosen the test: if attribute Pu normalized Norm-Pu < 0.06 then go to the left successor with full membership, if Norm-Pu > 0.553 go to the right successor with full membership, if $0.06 \leq$ Norm-Pu ≤ 0.553 go to both successors with intermediate memberships. Or in other words, go to left if Norm-Pu ≤ 0.553 and go to right if Norm-Pu ≥ 0.06.

Above each node, the name of the node is written, then the number of objects passing by the node and the cardinality of the node (the sum of the memberships degrees of all the objects that pass by). For example notation "T2: 282/145.9" designate the node test T2, with 282 objects as node size and 145.9 cardinality (thus not all the 282 objects have fully membership to this node but some of them are passing also by node T16 in figure 4.19). Notice that the sum of the cardinalities of the two successors of a node equals the cardinality of the node.

On the pruned FDT of figure 4.17 we remark that there is a terminal node whose label (output estimation) exceeds the [0,1] limits (see node L8). This is possible because the labels in nodes are determined each time by a linear regression optimization and they are not controlled and restricted to drop in the 0-1 interval. Once the final numerical output is computed by aggregation of all the labels in terminal nodes, we remark that on the pruned version of the FDT, generally this output does not exceed the 0-1 limits, or slightly exceeds the 0 limit (as we may observe in figure 4.25 explained latter). By transforming this numerical output in a two-class classification output (i.e. a class), all the output estimations above 0.5 will be considered one class, all the rest (under 0.5) will be considered the other class. This explains why labels exceeding 0-1 limits do not affect classification results in a two-class problem.

On the refitted FDT of figure 4.18 we remark that the refitting process has changed the terminal nodes labels, sometimes considerably with respect to the labels on the pruned FDT. Here again, they can exceed 0-1 limits.

On the backfitted FDT of figure 4.19 not only the labels in terminal nodes have been changed but also the parameters linked to the tests in every test node (see the numbers on each branch compared with pruned or refitted versions of FDT). The output estimators given by the backfitted FDT may come out of the 0-1 interval (as we may observe in figure 4.26 explained latter).

Figures 4.20, 4.21 and 4.22 are the backfitted FDTs of the forest built for the three-class problem of iris. In each figure caption we indicated the class which is explained by the FDT against the mixture of all the other classes. The fuzzy decision tree built for the recognition of the class iris-setosa (figure 4.21) has very well detected that this class is linearly separable from the other two, in conformity with the definition of the database. The final error rate of this forest is 6.67%.

Figures from 4.23 to 4.26 represents graphics related to omib data. They show the fuzzy class to be estimated (figure 4.23) and the estimation of this class given by the CART regression tree (figure 4.24), pruned (figure 4.25) and backfitted (figure 4.26) fuzzy decision trees. CART and FDT are built on the same growing set with $||GS|| = 500$. FDT backfitting is done on $GS + PS$. We remark the staircase character of the regression tree estimation and the soft continuous one provided by the fuzzy decision trees. Also, by backfitting, this output estimation is more fitted to the data, even if there are some examples for which the 0-1 limits are surpassed.

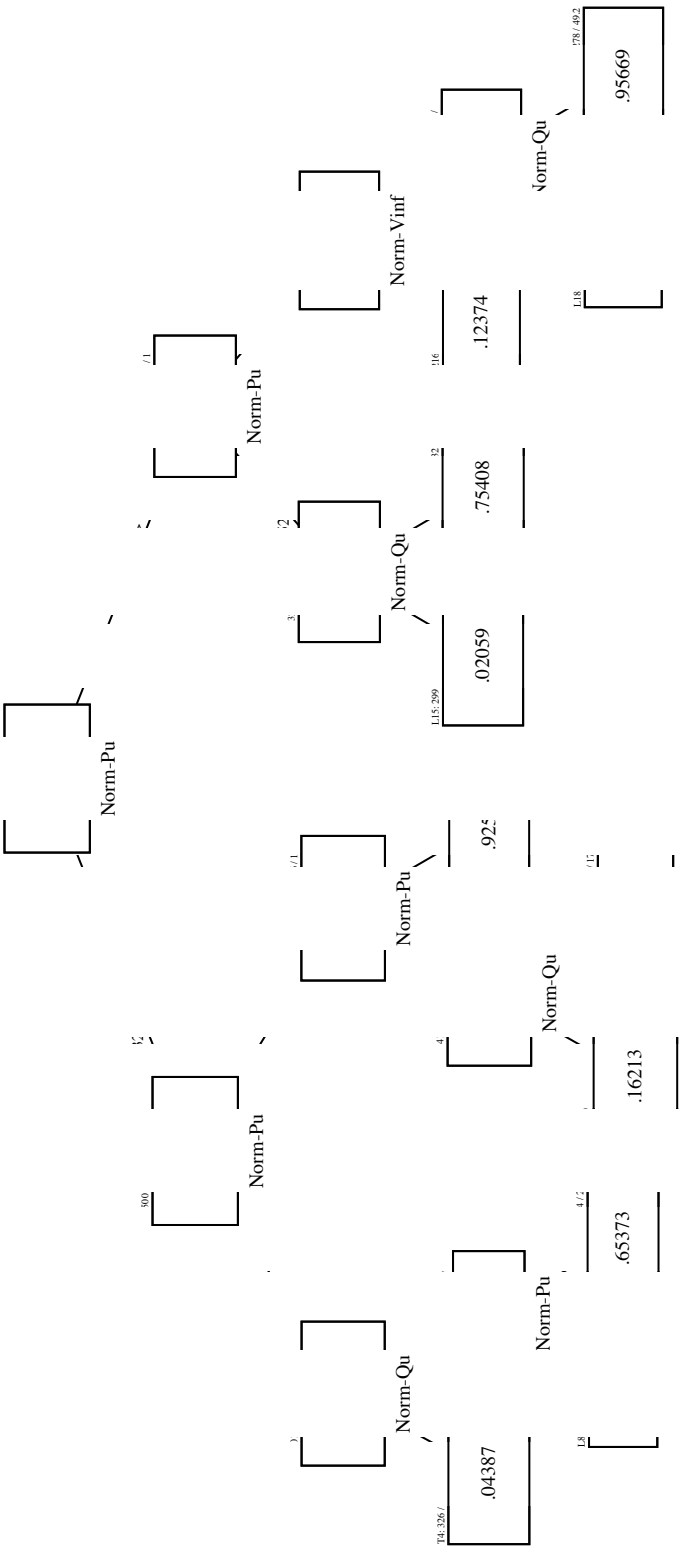


Figure 4.17: Pruned FDT on omib dataset. $card_{min} = 40$, $||GS|| = 500$, $||PS|| = 2000$, $||TS|| = 4000$. $C = 10, Pe = 13.92\%, MSE = 0.050998$.

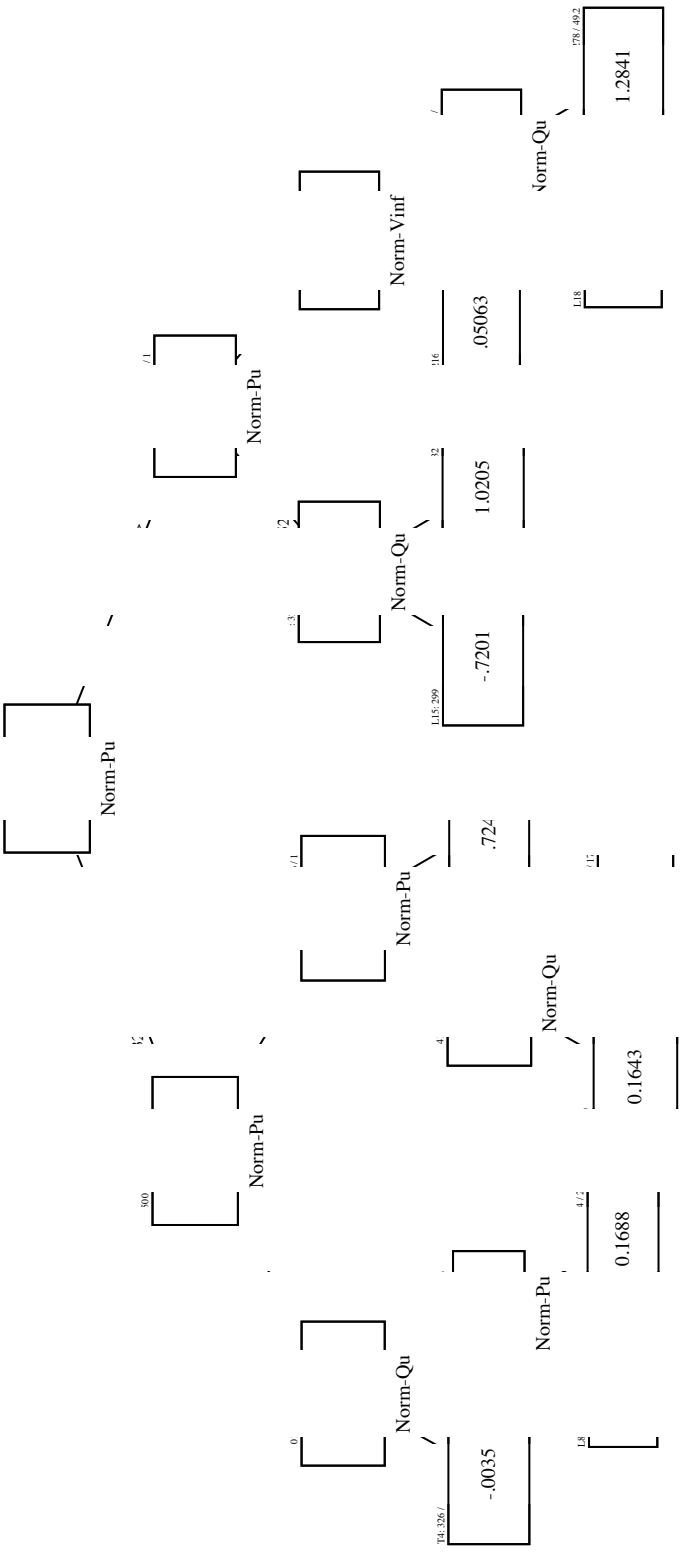


Figure 4.18: Refitted FDT on omib dataset. $||GS|| = 500$, $||PS|| = 2000$, $||RS|| = 2500$, $||TS|| = 4000$. $C = 10, Pe = 13.40\%, MSE = 0.043027$.

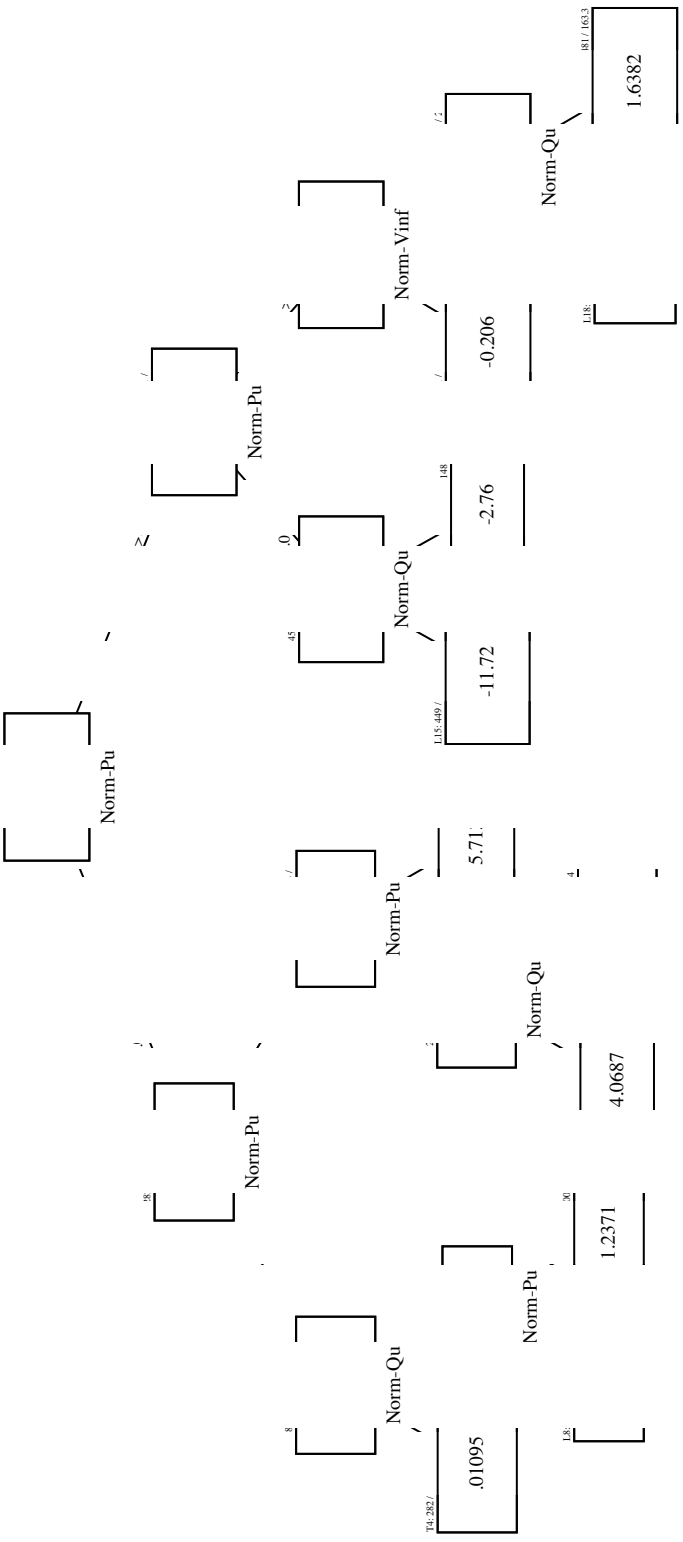


Figure 4.19: Backfitted FDT on omib dataset. $||GS|| = 500$, $||PS|| = 2000$, $||BS|| = 2500$, $||TS|| = 4000$. $C = 10, Pe = 10.72\%, MSE = 0.033169$.

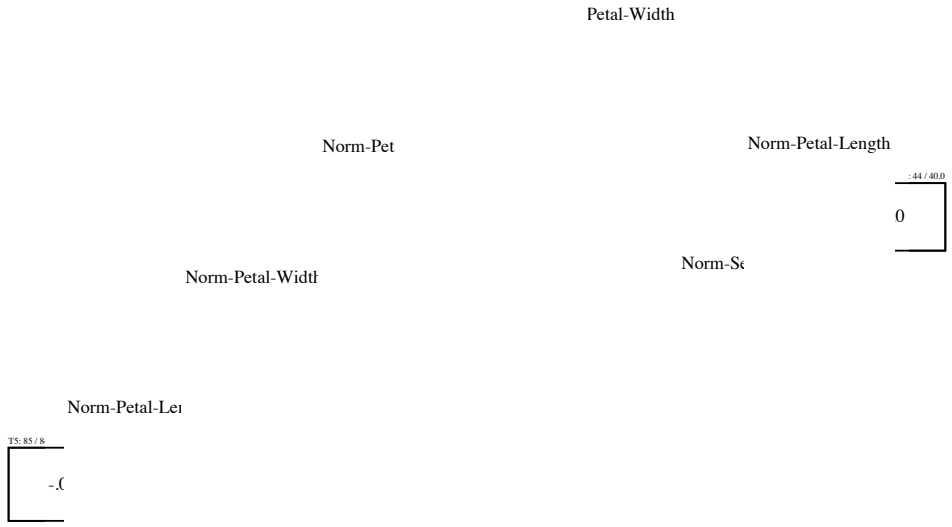


Figure 4.20: First backfitted FDT on i ass=iris-virginica. $||GS|| = ||PS|| = ||RS|| = ||BS|| = 135, ||TS|| = 15$. C 7%, $MSE = 0.066675$.

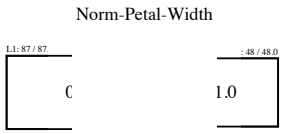


Figure 4.21: Second backfitted FDT on iris dataset. Class=iris-setosa. $||GS|| = ||PS|| = ||RS|| = ||BS|| = 135, ||TS|| = 15$. $C = 1, Pe = 0.0\%, MSE = 0.0$.

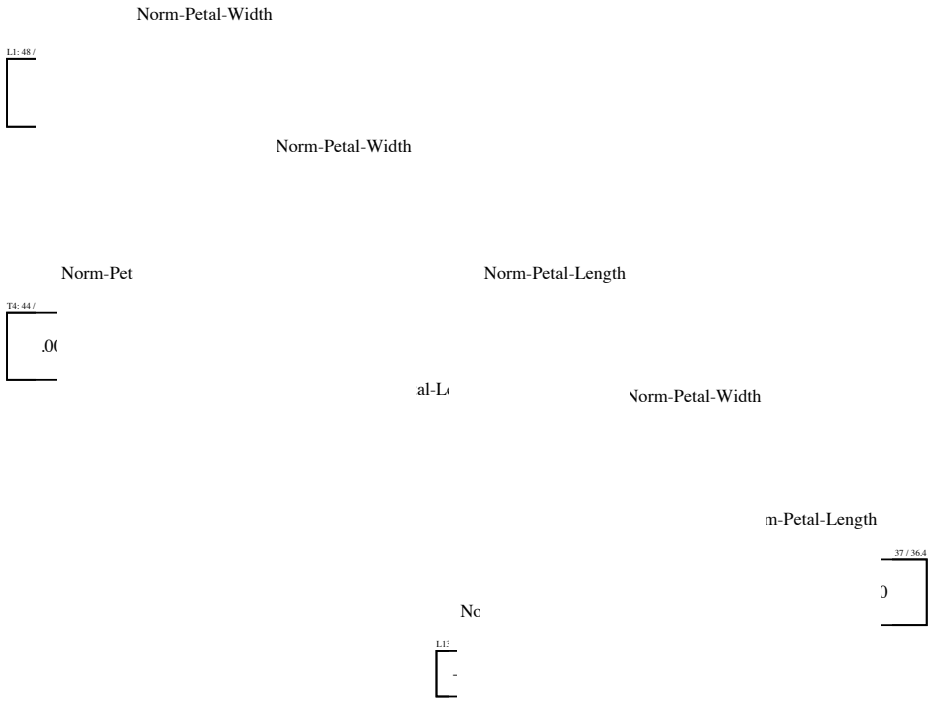


Figure 4.22: Third backfitted FDT on iris dataset. Class=iris-versicolor. $||GS|| = ||PS|| = ||RS|| = ||BS|| = 135, ||TS|| = 15$. $C = 11, Pe = 6.67\%, MSE = 0.066676$.

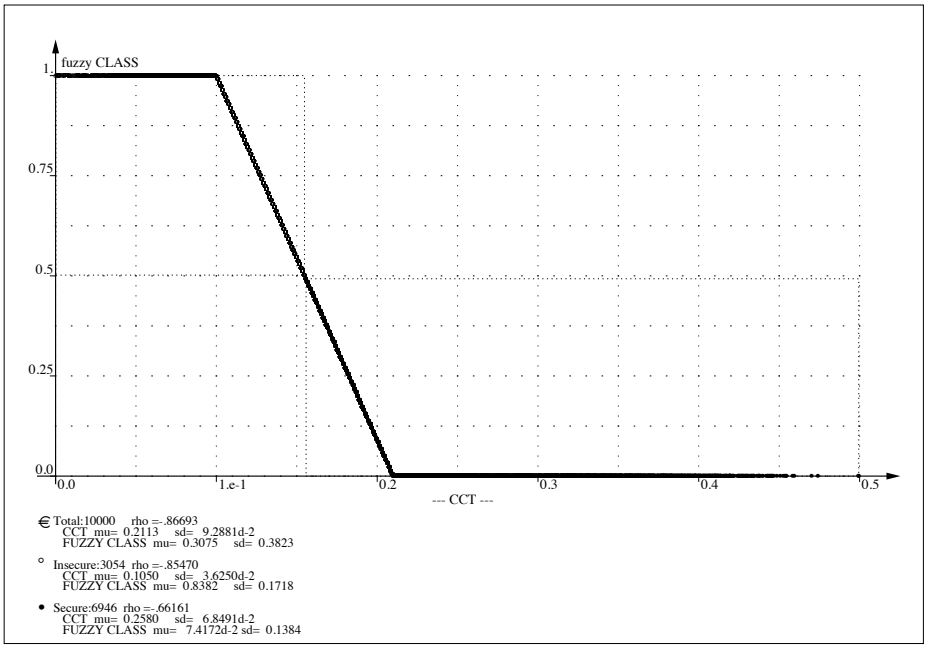


Figure 4.23: Real fuzzy class for omib data

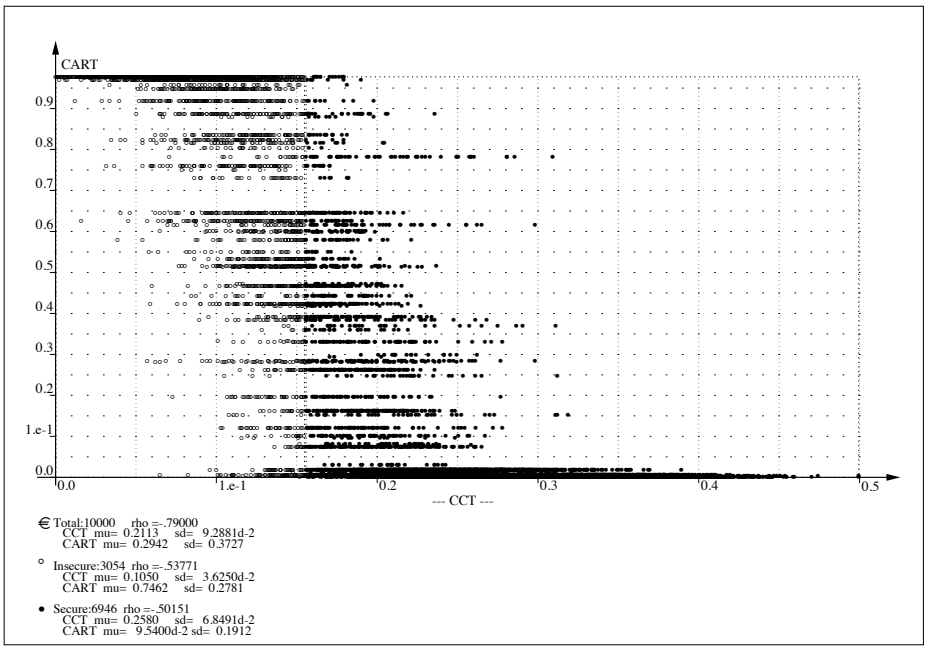


Figure 4.24: CART estimation of the output fuzzy class for omib data. $||GS|| = 500$, $||PS|| = 2000$, $||TS|| = 4000$. $C = 63$, $Pe = 10.45\%$, $MSE = 0.036737$.

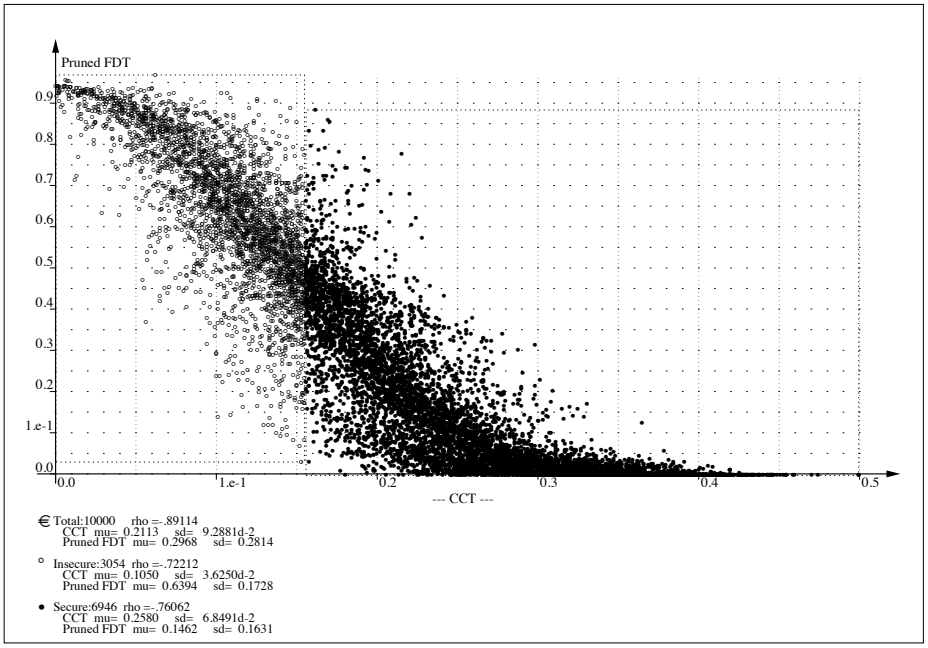


Figure 4.25: Pruned FDT estimation of the output class for omib data. $||GS|| = 500$, $||PS|| = 2000$, $||TS|| = 4000$. $C = 39, Pe = 8.78\%, MSE = 0.030792$.

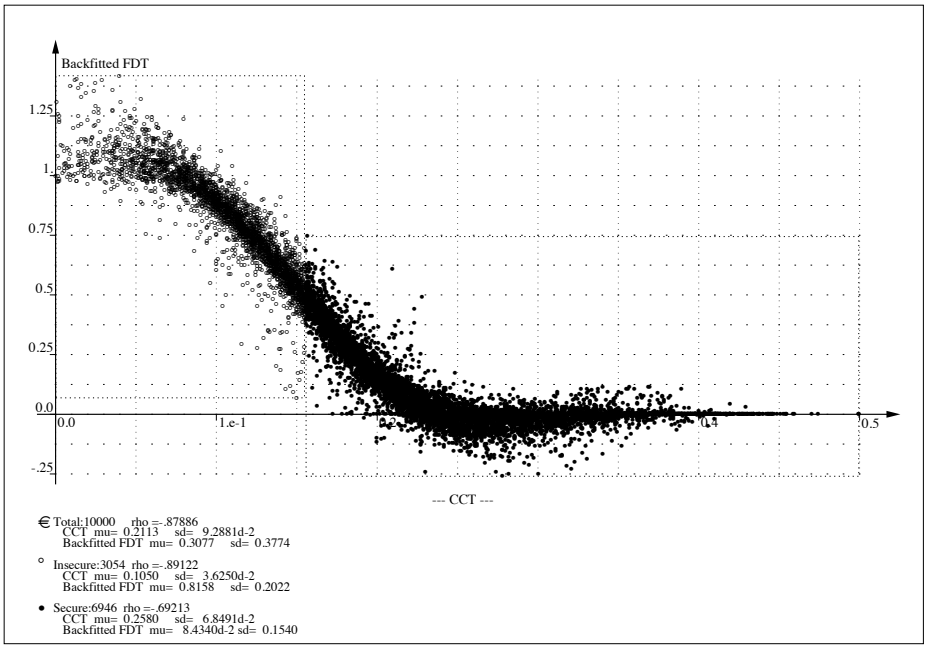


Figure 4.26: Backfitted FDT estimation of the output class for omib data. $||GS|| = 500, ||PS|| = 2000, ||BS|| = 2500, ||TS|| = 4000$. $C = 39, Pe = 2.62\%, MSE = 0.005830$.

Chapter 5

Existing Fuzzy Decision Tree methods

This chapter makes a short review of the major related literature to fuzzy decision trees and points out different aspects of existing fuzzy decision tree methods.

5.1 Introduction

People brought out in the early nineties some work under the generic name of fuzzy decision tree inductive learning and in particular: fuzzy ID3, fuzzy CART, large tree classifier, continuous ID3 algorithm, neuro-fuzzy ID3, globally optimal fuzzy decision trees, fuzzy neural trees, etc.

The first fuzzy decision tree reference [CP77] is attributed to Chang and Pavlidis, in 1977, where the fuzzy decision tree is supposed to be already settled, and a method of how to extract the final prediction for a new input is presented. Since then, more than fifty approaches treat this kind of algorithm, and they are devoted also to the induction process of such a tree from a set of training data, usually with the purpose of classification.

5.2 Classification of related methods

Considering the literature all around the construction of fuzzy decision trees, the following main directions have been addressed:

- 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 [CC87, Bun92, Qui93, Jor94a, Jor94b], or neural implementations [Set95]. By soft splitting, a continuous transition is allowed across the threshold in a node; the splits are not all-or-none, but instead are modulated continuously from a sharp split to no split at all;

- Crisp decision trees that use fuzzy sets (precisely, fuzzy clustering [PS01]) and possibility theory [XH98]) in order to better discretize continuous attributes. The rules extracted from such a tree are still crisp;

- Fuzzy decision rules obtained by fuzzifing the crisp rules extracted from a crisp decision tree [TST92, RJ94, CY96, HL97];

- Fuzzy neural networks or neuro-fuzzy approaches obtained by transforming a crisp decision tree into a neural network which is then fuzzyfied [Hei94, Hei95a, CS97];

- Supervised learning neural networks derived from crisp decision trees (no fuzzy logic inside) [Utg89b, Set90, AG92, CL92, Hei95b, Hei96, Hei98]. [Kub98] initializes a *radial*

basis function network (a single hidden layer neural network using Gaussian activation functions) with a decision tree that define relatively pure regions in the instance space and each of these regions then determines one basis function. [WA93] transforms a decision tree in a four-layer feed-forward neural network, the weights of which are tuned to enhance classification reliability and/or transform the discrete classification information of the tree in a continuous variable;

- Crisp decision trees extracted from supervised learning neural networks [CS96, Boz02a, Boz02b] in order to alleviate the black-box deficiency of a neural network;
- Fuzzy rules obtained by a fuzzy inference system then trained with a neural network [RJ91, RJ93] (no decision tree involved);
- Fuzzy decision rules obtained by other induction learning methods than decision trees [Che88, KI90, KI92, SS93, WM95, Wu99, FFAK99, Ber00];
- Non-linear crisp decision trees that performs a non-linear feature space partitioning [IS96a, IS96b] (by considering products and squares of all the candidate attributes as inputs). These authors [IZR+97] argue that a fuzzy decision tree does also a non-linear fuzzy partitioning;
- Fuzzy decision trees [Mar94, UOHT94, YS95, ISNM96, Jan96a, BW99, SL99, TWY00], etc.

In what follows we focus our discussion only on fuzzy decision tree approaches, thus, precisely, on the last class of methods classified above.

Fuzzy decision tree approaches have common aspects with all the other groups of methods that originate in tree-structured inductive ones. For example, approaches that translate a decision tree in a neural network are useful at the optimization phase of a fuzzy decision tree. The process of fuzzification of the crisp rules extracted from a decision tree has been modified in order to directly fuzzify a decision tree and to obtain a fuzzy decision tree. Of interest for fuzzy decision trees are the methods that fuzzify a neural network extracted from a decision tree so as to obtain a fuzzy neural network.

Fuzzy decision tree approaches may be split further in two main groups:

- Fuzzy decision trees that *prefer starting with a crisp tree* and once the tree architecture has been fixed, searching for the degree of softness in every node of the built tree [JJL97, LP89, LKC+95, SL99]. We say that the initial crisp tree is fuzzyfied;
- Fuzzy decision trees that directly integrate fuzzy techniques *during the growing phase* [AZZ98, Boy95, BW95a, BW99, CS92, HOJ98, ISNM96, Jan96b, Jan98, MBM97, Ram94, SHM99, TWY00, TMMN96, UOHT94, WS87, WCQY00, Web92, YS95, ZS96]. The induction process sets the node fuzziness once a new node of the tree is being developed.

The fuzzy decision tree method presented in this thesis belongs to this latter category.

5.3 Aspects in fuzzy decision tree methods

In what follows we tempt to put in evidence at an intuitive level some aspects that we considered relevant of existent fuzzy decision tree approaches, so that the reader could have an opinion about their state of development. By reading this survey one could better situate the method proposed in this thesis in the context of current fuzzy decision tree approaches.

Note: This analysis of existing fuzzy decision tree methods is based upon a number of approximately fifty references of fuzzy decision trees. Thus all the statements we make are based on this strict literature that one can find at the end of the manuscript.

5.3.1 Applicability

Inputs and outputs. The construction of the majority of methods starts from a crisp decision tree algorithm. [SL99] is a single present method laying on a regression tree algorithm, following CART [BFOS84] and handling also regression problems. Thus, the majority of approaches works only for classification problems: the tree response is expected to be only crisp.

Inputs may be numerical attributes, qualitative, or numerical attributes fuzzyfied a priori to tree growing, thus fuzzy attributes. Janikow FIDMV algorithm is able to treat also attributes with missing values [Jan94].

A few approaches are designed to handle beside crisp classes also fuzzy defined output classes [CS92, SATN95, TWY00, YS95, BW99].

Multiclass problems. Certain methods are conceived for the learning of no more than two classes, in which case a tree may be built for the recognition of a single class against the combination of all the others, all resulting in a forest of fuzzy decision trees [MBM97]. By aggregating the results of all the trees, we may obtain the correct classification for more than two classes. [DK01, SATN95, WS87, ISNM96, SL99, YS95, Ram94, BMMR96, HOJ98, TWY00, VCWC94, TMMN96] are methods that may be directly applied to classification problems with more than 2 classes.

5.3.2 Growing

Fuzzy ID3. Many approaches [BLM98, BMMR96, CS92, HOJ98, ISNM96, IZR+97, Jan96b, Jan98, JJL97, MBM97, MBM99, Ram94, SATN95, SATN96, SYSW01, TWY00, UOHT94, WCQY00, Web92, YS95, ZS95] start from the ID3 decision tree algorithm [Qui86] and inherit its weak points. The attributes may be only of symbolic type, they require an a priori fuzzification at the beginning or user defined fuzzy sets, stopping criteria do not bother about fitting the tree structure to data (no pruning), etc.

Attributes fuzzification. The generation of the membership functions (the fuzzification) in a tree node is done like follows:

- *A priori* to tree building. This is a common practice in fuzzy decision trees. There are a few possible approaches:
  - user predefined fuzzy sets; the fuzziness degree is a fixed one for a given attribute, implicitly for all the nodes concerned by this attribute [SHM99, SYSW01, TWY00, UOHT94, Web92];
  - evenly fuzzy partitioning an attribute when the degree of overlap is specified [BLM98];
  - fuzzy clustering the attribute, based on Kohonen's feature-maps [YS95], based on Bayes decision regions [WS87], or based on K-means clustering [DK01];
  - by fuzzy statistics (counting the data located nearby several points on the fuzzy set), associated then with curve fitting of the reference polynomial functions [HL02].
- By *semi-automatic* generation:

  - searching for the best degree of fuzziness in a tree node by exhaustively spanning an a priori defined list of possible degrees [Boy95, Ram94], or by imposing a minimal degree of fuzziness [Ram94].

- By *automatic* generation:

  - as the tree is being developed; [Jan96b] generates membership functions by dynamic optimization based on genetic algorithms at a node while building the tree (the genetic algorithm encodes the center and width of the fuzzy membership functions subject to tuning); [HOJ98] locally optimizes them by adjusting t-norms and t-conorms operators, both defined function of a free parameter; [CH02] locally optimizes them by fuzzy c-means clustering;
  - after the crisp tree has been developed [SL99].

Excepting the automatic generation of the membership functions as the tree is being developed, all the other approaches generally limit the approximation capabilities of the method, by limiting in a way or another the choice of the degree of softness in nodes. The drawback of the fuzzy partitioning a priori to tree building is usually that the class does not interfere in the process and the fuzzification is done independently of the target class. The automatic generation as the tree is developed permits a fuzzy partition at each step of the tree building, thus the induced partition being related to the *local* learning set of the current step of the tree building.

Marsala developed a less ordinary automatic fuzzy partitioning of a numerical attribute based on mathematical morphology [Mar95, Mar96]. He sees a learning set of an attribute as a "word" and then applies operators from the mathematical morphology like erosion and dilation, opening and closure, and filtering. The word is smoothed through these operations and a fuzzy partition is induced. For each attribute the fuzzy partition is in this way determined and then a discriminating measure compares the partitions for all the attributes. In his Ph.D. thesis [Mar98b] he also extends these operators to the case of fuzzy attribute values and fuzzy classes.

The encountered *shapes* of the membership functions used in the fuzzy partitioning of an attribute are: piecewise linear (triangular, trapezoidal), sigmoidal, Gaussian, clusters and the distance to them [WS87] or B-spline [ISNM96].

Measures for splitting a node. Generally, tree-structured approaches use to select a split based on a minimization of an impurity measure. This measure, called also *discriminating measure* used to select the best attribute for the partitioning of a (fuzzy) set, may differ from the measure used to fuzzy partition (split) the set in one or more fuzzy sets (the difference between finding the attribute and finding its best threshold for split in a node). Measures presently employed in a fuzzy decision tree to evaluate the most discriminating attribute in a node based on local growing set and sometimes also to search for the best choice of attribute fuzzy partitioning (if this fuzzy partitioning is not already done a priori to tree building, as it is the case of ID3 based fuzzy decision trees) are:

- fuzzy entropy based; the most used measures in this category are the (fuzzy) *star entropy* ([MRTZ98]), the information gain ([CH02, GHS02, Jan98, SYSW01, UOHT94]) and the gain ratio ([DK01]); [CS92] presents other four possible fuzzy entropy measures different from the star entropy: the fuzzy entropy defined by De Luca and Termini [DLT72], the index of fuzziness of Kaufmann, a measure defined by Yager and the fuzzy entropy of Kosko [Kos92] with Dombi's generalized fuzzy operators;

- Kolmogorov-Smirnov distance based ([BW96]);

- classification ambiguity ([YS95]);
- fuzzy sum of squared errors ([EVDB98]).

Beside this "classic" way of splitting, there are "non-conventional" methods. [AZZ98] computes at each node the score by a neural network whose inputs are the branches of the tree and whose outputs are the corresponding preference scores. [TMMN96] uses a genetic algorithm for structure identification of the fuzzy decision tree.

[Mar98b] states that the prediction power of a tree is dependent of the chosen discriminating measure and the author defines three methods of validation of the "goodness" of a discriminating measure.

[BW95b, Mar98b, Ram94, SL99] have observed the phenomenon of preferring a crisp partitioning to a fuzzy one in a node of the fuzzy tree for the *convex* discriminating measures and [BW95b] gives a proof to this. [Ram94] and then [Mar98b] define a minimal spread degree so as to avoid the crisp partition. For [SL99] the width parameter is kept fixed in classification problems during the optimization, so as to avoid fuzzy tree collapsing in a crisp decision tree.

Stop splitting rule. The usual criteria encountered are one or more of the following:

- limitation of the fuzzy probability of the node (relative frequency) with respect to one class [ISNM96, HOJ98, UOHT94, Jan98, TWY00];
- a threshold condition on the cardinality of a node [HOJ98, Jan98];
- limitation of the depth of the tree [BW99];
- if the next best splitting does not improve the tree, then no-splitting [BW99];
- limitation on the fuzzy entropy of the node [FJK99, Mar98b, Jan98];
- limitation of the number of instances in the local learning set of the node [Mar98b, SL99, UOHT94];
- limitation on the number of attributes already chosen in the grown tree [Mar98b];
- limitation of the truth level of classifying into one class [YS95];
- a threshold condition on the number of objects which are in error [DK01];
- at least 90% of the objects in node belong to one class [GHS02];
- if the performance on the test set starts to deteriorate, then stop the split [EVDB98].

These criteria are basically referring to two aspects: i). the "impurity" of a node that counts for how alike are the objects in a given fuzzy set, and ii) the "size" of a node that could be too insignificant to continue the partitioning. It happens they employ the same measure used for splitting the node, or a different one.

Terminal node labeling. In the fuzzy decision tree literature, the following possible ways of labeling a terminal node are more encountered:

- in accordance with an established criterion usually providing the most typical membership to each class in the *local* growing subset of the node. This "typicality" may be expressed by:

  - fuzzy probabilities [BLM98, ISNM96, UOHT94];
  - "sharpened" average [BW99] or weighted average [EVDB98] on the local membership degrees to the target class;

- with a single class, the one with the highest truth level in the node [YS95], with the greatest membership degree to the class [UOHT94], or the class of the majority of objects in the node [Mar98a];

- through a global optimization procedure [SL99], all the terminal nodes being labeled at a time; Ref. [MCSL01] replaces tree labels by linear models and globally optimizes them.

Inference step. The fuzzy decision tree approaches differ in the choice of the inference measures: t-norms (min, product, and), t-conorms (max, sum, or), etc. Ref. [Jan98] defines four different inferences. Ref. [Mar98b] tries several operators with its method: Zadeh, probabilistic and Lukasiewicz. Ref. [HOJ98] defines t-norm and t-conorm measures as a function of a parameter which has to be globally optimized.

Defuzzification. The final answer of a fuzzy decision tree is computed with one of the defuzzification methods: max-criterion [ZS96], center-of-area [SL99, VCWC94], mean-of-maximum [BLM98], etc.

Other (atypical) approaches

- *Look-ahead* based fuzzy decision trees [DK01]. They perform a multilevel look-ahead with the help of a nonparametric method which characterizes the classifiability of instances to be split along branches of a given node. The measure for split in such a tree is a compromise between a fuzzy entropy factor and a look-ahead term. Less complex trees are obtained by such a look-ahead method.

- *Incremental* fuzzy decision trees [GHS02]. They are updated with each new object presented to the tree. Labels in nodes may change, and even the structure of the tree: tests in nodes may swap, some terminal nodes may be further developed and some internal nodes may be pruned). Unlike non-incremental fuzzy decision trees which have no way to adapt their results to new data, but rather have to be restarted from scratch, incremental trees allow rapid alternation between classification and learning of new data.

- Fuzzy decision trees used to *extract fuzzy symbolic comprehensible knowledge from a neural network*. FuzzyTrepan [FJK99] is a tool that trains a neural network, and then a fuzzy decision tree has to learn the objects whose original class has been replaced by the class generated by the trained neural network (the original Trepan of [CS96] extracts a crisp decision tree from a neural network).

5.3.3 Pruning

Pruning, the tradeoff between the accuracy of the tree and its complexity, is rarely provided and in most cases is not meant to cover large databases applications. Ref. [ISNM96] provides a kind of pruning based on Akaike information criterion. Ref. [YS95] proposes a rule simplification technique applied on the rules obtained by transforming the tree in a rule base, based on the truth level of a fuzzy rule. On the other hand, [BLM98] provides pruning based on grouping words (labels of a linguistic variable), words generated when partitioning the attribute. In [WCQY00] one proposes a merging branches algorithm based on a fuzzy value clustering of branches and [SHM99] performs pruning based on the minimum description length. Finally, [Boy95] settles

a complete pruning procedure that works well on a large scale application, but at a cost of high computational requirements.

5.3.4 Optimization

Due to the non continuity and non differentiability of the model error function, many of the present fuzzy decision tree methods can not implement a global training or a global final optimization after identifying the model structure. However, there are global optimization based on genetic algorithms [SATN95], similarity measures [WS87], or backpropagation of neural networks [CS92, ISNM96, SL99, TWY00]. In this last case, the decision tree initially determines the structure of a neural net (relevant inputs, network complexity) and then the free parameters are globally optimized within the neural network.

5.3.5 Real-life applications

From the point of view of fuzzy decision tree applications, in our opinion there is not enough application on large-scale problems. Also, in general, no explanations are provided in order to be able to evaluate the impact of applying these algorithms on large databases. However, in the last few years, several promising applications of fuzzy decision trees in real life domains emerged (some of them are large-scale applications):

- chemistry field, application to the recognition of odors [MRTZ98];

- geographical object-oriented spatial database, where fuzzy decision trees are used in conjunction with an extension of SQL language in order to locate houses and classify them into one of the two classes (urban and non-urban area) [BM98];

- data warehouse in education concerning individual results in high school, where fuzzy decision trees learn based on results of OLAP queries addressed to the database (OLAP determines statistics in every new node of the fuzzy tree) [LGM00, LBMD+00];

- real time robot system in a strategy-based decision making process (a 3-to-3 robot soccer game) [HL02];

- medical field, tumor detection in digital mammography [LKC+95];

- military database, fuzzy decision trees are used in order to predict the behavior of an enemy [SI02];

- power systems field, security assessment task, transient stability degree detection of a large power plant [BW99];

- identification of damaged transformers [UOHT94].

No reference in fuzzy decision trees mentions hints about the efficiency of the learning method in terms of the computational burden or CPU times. An exception is ref. ([WS87]).

5.3.6 Validation

Concerning the validation part, generally, the authors do not compare their fuzzy decision tree results with other interpretable methods on the same data. In the happiest case the authors compare results obtained with variants of the same fuzzy decision tree algorithm. The only literature that compares results of fuzzy decision trees and crisp decision trees (being for regression or classification) is [Boy95, BW99, JF99, MBM97, SL99, TWY00]. Other models used in comparisons are: neural networks [TWY00], statistical analysis [TMMN96], bagging and smearing [MCSL01], k-nearest neighbors [SYSW01]. As exceptions, ref. [DK01] compares its look-ahead fuzzy decision trees with fuzzy decision trees of [CS92] and ref. [CH02] compares its fuzzy classification trees with four types of fuzzy decision trees ([Jan98, SL99, YS95]).

5.3.7 Visualization

One of the most attractive qualities of a fuzzy decision tree is its interpretability: looking to a tree structure, one is able to understand how does the "machine" decide which is the best result in prediction or classification. That is why, we find very important to be able to show the resulting tree or the extracted decision rules (to the reader, to experts in field of concern, or to the tree users). Most of the publications in the field of fuzzy decision tree do not bother with this aspect. Exceptions are references [SHM99, TMMN96, BW99, WCQY00, HOJ98, JJL97, HL02, Mar00], even if they do not seem to have a module on fuzzy decision tree visualization. By visualizing the tree one could rapidly perceive its complexity, the most effective and relevant features contributing to the answer, or which could be the response for a given new instance.

Chapter 6

Conclusions

This chapter concludes the work. Firstly, it briefly reminds the main steps of the proposed algorithm. Secondly, a summary of the empirical results is provided, followed by a short discussion on the intuitive reasons for which fuzzy decision trees have a better accuracy than crisp decision tree methods. Thirdly, directions of further work are ending this chapter and the thesis.

The objective of this thesis has been to propose a new method of fuzzy decision tree induction in the aim of the data mining practices requirements, interpretability, accuracy, efficiency and autonomy, to validate this method and to extensively study it. The method, being issued from crisp regression tree induction and from the fuzzy sets theory, primarily, it inherits two most attractive qualities of crisp regression tree induction: the interpretability and the autonomy, and secondly, due to the help of fuzziness, it can be significantly more accurate than standard decision and regression trees, as shown by the empirical tests carried out in this work.

6.1 Method technicalities

In chapter 3 we described the proposed soft decision tree method. The method comprises:

- A *growing* step which automatically partitions in a fuzzy way each new node of the tree throughout a decomposition of the search in the space of four parameters, based on an error function of squared error type. The search for the cutting point location is done exactly as in a crisp CART regression tree, then the fuzziness degree is determined by FIBONACCI search and for each new fuzziness degree, the labels of the two node successors are updated by explicit linear regression formulas.
- A *pruning* step whose goal is to remove irrelevant parts of the already grown tree. All the tree nodes are sorted by their relevance in terms of the error function, then based on this sorted list, nodes are pruned one by one, obtaining a sequence of nested subtrees bordered at one side by the fully grown tree and at the other side by the trivial tree. Finally, a single tree is picked up from this sequence based on estimations of the errors of each of the generated trees and the one-standard-error-rule. Two variants for the relevance measure have been proposed, one as the local squared error estimate in the node, the other as the global error of the subtree rooted at the node.
- A *refitting* procedure based on linear regression that globally optimizes the labels of all the terminal nodes of the pruned tree, reflected in improved accuracy for the final tree. A variant of this step is called refitting during pruning approach and it consists of refitting every tree from the sequence of pruned trees generated in the pruning step and choosing the best one among these already refitted subtrees.

- A *backfitting* procedure based on LEVENBERG-MARQUARDT non-linear optimization algorithm, which globally optimizes all the free parameters of the pruned soft tree: labels of all terminal nodes, cutting point and fuzziness degree for all internal nodes. Again the aim is improved accuracy.

Other technical aspects characterizing our approach are:

- It handles numerical valued inputs only.
- It is able to handle both classification and regression problems. The output should be numerical. For classification problems, the output should be a priori defined as a numerical function (0/1). When applying the tree, the inverse conversion is done from numerical to symbolic so as to obtain as final result a class.
- Due to its continuous nature, it handles any kind of fuzzy output classes.
- In classification, a single soft decision tree learns a two class problem. Multiclass problems are handled by a forest of soft trees.
- There is a choice in settling the shape of membership function used in partitioning an attribute: it may be piecewise linear or sigmoidal in the present version of the software.

6.2 Summary of results

In this thesis we focused our attention on classification, which is an important data analysis task. At the validation stage, our method has been applied to 18 classification problems. It is compared to interpretable methods: pruned C4.5 and ULG decision trees, pruned CART regression trees and dual perturb and combine ULG decision trees [Geu02], but also to other non-interpretable but more accurate techniques like bagging, boosting, randomized trees and bagging + dual perturb and combine decision trees (the results for these later being published in [Geu02]). Next conclusions are issued from the experimental evaluation of our method.

Results show that classifiers based on our soft decision trees, be they refitted or backfitted, are more complex but significantly more accurate than standard decision and regression trees. The bias-variance study shows that this improvement in error for a soft decision tree versus a standard tree comes mainly from an important *reduction in variance*. Thus the thesis proves empirically that a fuzzy decision tree is indeed another tool for reducing variance, beside pruning and aggregation techniques.

Soft decision trees are more complex but more accurate than dual perturb and combine method. By looking at the proportions of bias and variance in the error we notice a *higher bias* for dual perturb and combine method compared with our method.

The aggregation methods we compared with are more complex but generally more accurate than our soft decision trees. In the cases where a soft decision tree presents better accuracy than one or more of the aggregation methods, this is due to a *lower bias*.

In terms of accuracy, we may situate fuzzy decision trees to the frontier between methods based on one tree model and methods that average several tree models.

Simulations on fuzzy versus crisp output class definition prove that a fuzzy output *decreases both bias and variance* with respect to a crisp output. This is partially due to the fact that a fuzzy class provides richer information than a crisp class and partially due to the possibility to manage this type of information of our fuzzy decision tree method. By using a fuzzy output definition, fuzzy decision trees outrun in terms of accuracy all the studied aggregation methods.

The bias and variance evolution with the model complexity study reveals that the larger a fuzzy decision tree, the better the accuracy, since variance is a small almost constant quantity and bias

decreases with respect to the model complexity. This is in opposition with a standard decision or regression tree, where bias decreases but variance increases with the model complexity, fact which determines an optimal tradeoff between these two quantities so as to obtain the smallest error.

The bias and variance evolution with the growing set size study confirms that the larger the growing set, the better the accuracy, since bias and variance decrease both with the growing set size.

Statistics on the cutting point parameter at the first two levels of depth of a tree show that fuzzy decision trees do not come with an improvement in the parameter variance with respect to standard regression trees, but however, both fuzzy and regression trees present less parameter variance than a standard decision tree. Thus, the reduction in prediction variance that a fuzzy decision tree presents over the crisp decision and regression trees does not rise from a reduction in the parameter variance. And *the fuzzy partitioning does not improve the parameter variance* as we would expect. Also, backfitting has a negative influence over the parameter bias and variance.

From a computational point of view, the method is intrinsically *slower* than crisp tree induction. In average, a refitted soft decision tree is approximately 20 times and a backfitted soft decision tree is 50 times slower than a pruned decision tree. This is the price paid for having a more accurate and still interpretable tool in the same time.

Among the two steps of optimization of the soft decision tree, refitting and backfitting, *we prefer the refitting step*, for its simplicity and efficiency, given that our tests showed that by refitting we already obtain very good results in accuracy.

Soft decision trees have the possibility to use in optimization step both the growing set and the set used for pruning. This contributes to the accuracy improvement.

The soft decision tree method is an autonomous tool. It has been conceived so as not to depend on a human expert in order to choose the best model parameters or complexity for obtaining an optimum result for each problem to be analysed. The pruning step improves a lot the model autonomy.

6.3 Reasons for increased accuracy

An intuitive explanation of why a fuzzy decision tree gives 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 [CC87, DK95, Fri96]. 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 fuzzy decision tree, two examples close to each other in the space of the attributes are treated in a similar fashion. In this way, fuzzy decision trees decrease bias near the region boundaries and thus the tree errors.

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 [Fri96]. In fuzzy 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.

An immediate effect of this growth of local samples size is that a fuzzy 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 [Tur95] 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 [Die00a, Fri96] and this may cause the users loss

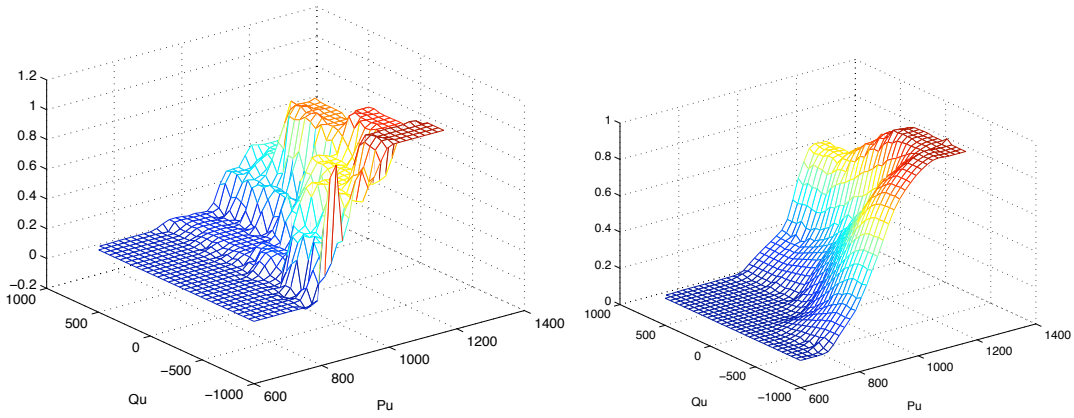


Figure 6.1: Regression tree versus fuzzy decision tree output

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 enough predictive accuracy. Empirical results of chapter 4 showed that fuzzy decision trees present much less global variance than crisp regression and decision trees. Hence, indeed, the stability of the learner is improved.

The third reason for a better accuracy is linked to the continuity of the fuzzy decision tree output. Figure 6.1 shows the discontinuous (staircase) character of a regression tree output (left part) versus the smooth output surface of a fuzzy 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 fuzzy decision tree output. As a consequence, the tree may be seen as a smooth parametric model and powerful optimization techniques may be applied in order to further improve the predictive accuracy of the tree.

6.4 Extensions and improvements

There are two directions of further work which we would explore in a future work: first, the handling of qualitative attributes, which would make the method more independent of the nature (type) of the data to be mined, and second, the improvement in computational time, which would make the method more efficient.

Treatment of qualitative attributes. 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, $a_q : U \rightarrow \{s_1, \ldots s_m\}$, the fuzzy partitioning approach searches for a discriminator function $\nu : \{s_1, \ldots s_m\} \rightarrow [0 \ldots 1]$, $\nu = [\nu_1, \ldots \nu_m]$, that minimizes the same error function as in the numerical attribute case, but instead of two parameters there are m parameters to optimize: ν_1,..., ν_m. A non-linear optimization technique or a clustering approach may be used in order to optimize the m parameters defining ν, or they may be determined by simple estimated conditional probabilities in the considered node, as $\nu_i = P(C|s_i)$ and $1 - \nu_i = P(\bar{C}|s_i)$.

Improvements in computational time. Improvements can be done on the implementation side so as to render the whole method more efficient:

- So far, experiments with refitting during pruning variant reflect accuracy improvement. In this respect, further work should be done in terms of computational complexity, by con-

ceiving incremental procedures so as to reduce the amount of computations, which then becomes cubic in terms of model complexity before pruning.

- For multi-class problems, forest of trees are built. Building in parallel all the trees in the forest by a distributed process, CPU time could drastically be reduced.
- The code behind the fuzzy decision tree method is not optimal and may be improved in certain aspects.

Further experiments. Since some of our experiments are not systematic, there are still some fields where more simulations would confirm or infirm our results: variance and bias studies on multi-class problems, more systematic parameter bias and variance studies, simulations with the variants for pruning and growing on more datasets, more systematic validation of the refitting during pruning approach.

Since all this thesis has been oriented toward classification problems, a clear objective in the future would be to validate the method and to study in depth its behavior also on *regression problems*.

Alternative FDT growing method: residual fitting. As an alternative to the growing procedure experimented in this work, a different one called "residual fitting" [Ola98] may be implemented. The technical formulas of its deployment are presented in appendix C. The idea behind this approach is that at every new node to be split, we minimize an error function that is a kind of residual error: the error that was not yet reduced by the already developed parts of the tree. Thus, at every new node to be split, we take into account the already developed parts of the tree.

It would be interesting to try this residual fitting approach since it has a global character unlike the already experimented growing approach which has a local character.

***

Appendix A

CART - FDT parallel summarized in formulas

This appendix makes a summary on the fuzzy decision tree formulas against the CART regression tree formulas concerning the growing procedure.

Table A.1: CART - FDT parallel

CART	FDT
output $\mu_C(o) \in \mathbb{R}$	
degree of membership to a node S and to each of its successors: S_L and S_R $\mu_{S_L}(o) + \mu_{S_R}(o) = \mu_S(o)$	
$\mu_S(o), \mu_{S_L}(o), \mu_{S_R}(o) \in \{0, 1\}$ if $\mu_{S_L}(o) = 1$, then $\mu_{S_R}(o) = 0$ and $o \in$ left successor S_L if $\mu_{S_L}(o) = 0$, then $\mu_{S_R}(o) = 1$ and $o \in$ right successor S_R	$\mu_S(o), \mu_{S_L}(o), \mu_{S_R}(o) \in [0; 1]$ if $\mu_{S_L}(o) = 1$, then $\mu_{S_R}(o) = 0$ and $o \in$ only to left successor S_L if $\mu_{S_L}(o) = 0$, then $\mu_{S_R}(o) = 1$ and $o \in$ only to right successor S_R if $\mu_{S_L}(o) \in (0; 1)$, then $\mu_{S_R}(o) \in (0; 1)$ and $o \in$ to both successors
$\mu_{S_L}(o) = \mu_S(o)\nu(o)$ $\mu_{S_R}(o) = \mu_S(o)(1 - \nu(o))$	
discriminator	
$\nu(o) = \begin{cases} 1 & \text{if } test_S \text{ is true} \\ 0 & \text{if } test_S \text{ is false} \end{cases}$ where $test_S$ is the test in node S	piecewise linear: $\nu(o) = \begin{cases} 1 & \text{if } o < \alpha - \frac{\beta}{2} \\ 0.5 + \frac{\alpha - o}{\beta} & \text{if } \alpha - \frac{\beta}{2} \leq o \leq \alpha + \frac{\beta}{2} \\ 0 & \text{if } o > \alpha + \frac{\beta}{2} \end{cases}$ or sigmoidal: $\nu(o) = 0.5[1 + tanh(4\frac{\alpha - o}{\beta})]$ where $tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$
cardinality	
$N_S = \sum_{o \in S} \mu_S(o)$ where $\mu_S(o) = 1$ always	$card_S = \sum_{o \in S} \mu_S(o)$ where $\mu_S(o) \in [0; 1]$; $card_S \leq N_S$
error function to be minimized	
mean squared error: $E_S = \frac{1}{N_S} \sum_{o \in S} [\mu_C(o) - \hat{\mu'}_C(o)]^2$ where $\hat{\mu'}_C(o) \in \{L_L, L_R\}$	squared error: $E_S = \sum_{o \in S} \mu_S(o)[\mu_C(o) - \hat{\mu'}_C(o)]^2$ where $\hat{\mu'}_C(o) = \nu(o)L_L + (1 - \nu(o))L_R$
score to be maximized when $\beta = 0$	
variance reduction: $1 - \frac{N_{S_L}}{N_S} \frac{s^2_{S_L}}{s^2_S} - \frac{N_{S_R}}{N_S} \frac{s^2_{S_R}}{s^2_S}$ where variance: $s^2_{S_{L,R}} = \frac{1}{N_{S_{L,R}}} \sum_{o \in S_{L,R}} [\mu_C(o) - L_{L,R}]^2$	squared error reduction: $1 - \frac{E_{S_L}}{E_S} - \frac{E_{S_R}}{E_S}$ where squared error: $E_{S_{L,R}} = \sum_{o \in S_{L,R}} \mu_S(o)[\mu_C(o) - L_{L,R}]^2$
label when $\beta = 0$	
mean: $L_{L,R} = \frac{\sum_{o \in S_{L,R}} \mu_C(o)}{N_{S_{L,R}}}$	weighted mean: $L_{L,R} = \frac{\sum_{o \in S_{L,R}} \mu_S(o)\mu_C(o)}{\sum_{o \in S_{L,R}} \mu_S(o)}$
estimated output	
one label: $\hat{\mu}_C(o) = L_j$	aggregated labels: $\hat{\mu}_C(o) = \frac{\sum_{j \in leaves} \mu_{S_{L_j}}(o) L_j}{\sum_{j \in leaves} \mu_{S_{L_j}(o)}}$

Appendix B

Datasets

This appendix describes all the datasets used in our experiments.

All the databases are taken from the UCI Machine Learning Repository [HB99], excepting omib and gaussian data, which are databases simulated at University of Liège Belgium.

For all two-class databases we formulated each numerical output as a 0/1 function based on the classification output, excepting omib data where a fuzzy class has been defined equally. For all multi-class problems, in CART regression trees and our fuzzy decision trees, each class is learned against the union of the other two; the numerical output is defined for every subproblem as a 0/1 function, and then the results are aggregated as shown in section 3.4.8 and a final symbolic class is pulled out.

B.1 Part I

Table 4.1 (section 4.2.1 of chapter 4) describes the datasets used for method validation in Part I of experiments. The datasets are large, with only numerical attributes and no missing values. The seven datasets investigated and the associated *classification* tasks are:

- **Gaussian**. It is a synthetic database with 20000 objects, 2 classes and 2 attributes. Each class corresponds to a bi-dimensional Gaussian distribution. The first Gaussian distribution is centered at (0.0;0.0) and has a diagonal covariance matrix, while the second one is centered at (2.0;2.0) and has a non-diagonal covariance matrix. There is an important overlapping between the two classes which determines a Bayes error rate nearby 11%. The Bayes classifier is of quadratic shape.

- **Twonorm** [Bre98] is a database with 2 classes and 20 attributes. Each class is drawn from a multivariate normal distribution with unit covariance matrix. One class has mean (a,a,...a) while the other class has mean (-a,-a,...-a) where $a = \frac{2}{\sqrt{20}}$. The optimal separating surface (Bayes classifier) is an oblique plane, hard to approximate by the multidimensional rectangles used in a crisp tree.

- **Omib** [Weh98] is an electric power system database, already introduced in chapter 3, in the example of section 3.1. It does a transient security assessment task. The OMIB (One Machine Infinite Bus) system is composed of a generator, a transformer, a load connected at the EHV (extra-high voltage) side of this transformer and a reactance representing the equivalent impedance of the EHV system. A three-phase short-circuit occurs close to the generator, normally cleared after 155ms. The problem output reflects the system security state after the short-circuit fault occurrence. This output is defined based on a security margin used in

transient stability studies, called the critical clearing time (CCT) of the disturbance. The input space is defined by 6 attributes representing pre-fault operating conditions of the OMIB system: active (Pu) and reactive (Qu) power of the generator, amount of load nearby the generator (Pl), voltage magnitudes at the load bus (Vl) and at the infinite bus (Vinf) and the reactance (Xinf). The continuous output is defined as the fuzzy (or crisp) class of figure 3.2. The problem is deterministic, that is why the Bayes error rate (the residual error) is null.

- **Waveform** [BFOS84] 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. The Bayes error rate has been estimated at 14%.

- **Satellite**. Each object of this database corresponds to a 3x3 grid of pixels corresponding to different areas of a satellite image of a particular region of the earth. Each of these grids corresponds to one of the 6 possible types of soil classes. 36 attributes reflect energy levels in different frequency bands obtained for each pixel in the 3x3 grid.

- **Pendigits** corresponds to a handwriting digit recognition problem. Handwritten digits (250 samples from 44 writers) are collected with the help of a sensitive tablet attached to a PC, on which the writers has been asked to write in random order. After a spatial resampling (points regularly spaced in arc length), 16 attributes are obtained, linked to the (x, y) coordinate information of these digits, integers in the range [0;100]. The class is coded from 0 to 9 (each class correspond to a digit to be recognized).

- **Dig44**. This is also a digit recognition problem. The 10 classes are examples of digits from 0 to 9 gathered from postcodes on letters in Germany. Examples were digitized into images with 16x16 pixels and then 16 attributes were extracted by averaging over 4x4 neighborhoods in the original images. The database contains 15000 objects.

B.2 Part II

Table 4.25 (section 4.3.1) describes the datasets used for method validation in Part II of experiments. The eleven datasets investigated and the associated *classification* tasks are:

- **Balance**. Balance Scale Weight & Distance database has been generated in order to model psychological experimental results. Each of the 625 examples is classified as having the balance scale tip to the right, tip to the left, or be balanced (3 classes). The 4 attributes are: the left-weight, the left-distance, the right-weight and the right-distance. The way to find the class is: if the greater of left-distance*left-weight and the greater of right-distance*right-weight are equal, the case is balanced.

- **Glass**. Glass Identification database is a problem of classifying the glass in 6 types (glass from a window, vehicle, container, head lamps, etc), motivated by criminologycal investigation (at the scene of the crime, the glass left can be used as evidence). The 9 attributes represent constituents of the glass (Na, Mg, Al, Si, K, Ca, Ba, Fe) plus a refracting index. There are 214 glass examples to be classified.

- **Heart** data detects the absence or presence of a heart disease (2 classes). 270 observations are described by 13 attributes like: age, sex, resting blood pressure, chest pain type, exercise induced angina, maximum heart rate achieved, etc.

- **Ionosphere**. This radar data has been collected by a system in Goose Bay, Labrador, which consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not, their signals passing through the ionosphere. Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the system. The 351 instances of the database are described by 2 attributes per pulse number (thus by 34 attributes), corresponding to the complex values returned by the function resulted from the complex electromagnetic signal.

- **Iris**. Iris Plants database contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class (iris-setosa) is linearly separable from the other two, but the latter (iris-virginica and iris-versicolor) are not linearly separable from each other. The 4 attributes used to classify the 150 plants are: the sepal width, the sepal length, the petal width and the petal length.

- **Monks** The 3 Monks datasets have the same domains: 432 instances are classified into 2 classes based on 6 attributes, from a1 to a6. Monks-1 problem considers $(a1 = a2)$ or $(a5 = 1)$. Monks-2 considers exactly two of the conditions: $a1 = 1$, $a2 = 1$, $a3 = 1$, $a4 = 1$, $a5 = 1$, $a6 = 1$. Monks-3 have $((a5 = 3)$ and $(a4 = 1))$ or $((a5 \neq 4)$ and $(a2 \neq 3))$.

- **Pima**. Pima Indians Diabetes database has 2 classes representing a diagnostic: whether the patient shows signs of diabetes according to World Health Organization criteria. The population lives near Phoenix, Arizona, USA. All 768 patients of the database are females at least 21 years old of Pima Indian heritage. A number of 8 attributes as the number of times pregnant, body mass index, age, 2-hour serum insulin, blood pressure, etc, are investigated.

- **Sonar** is a problem of classification of sonar signals in 2 classes: sonar signals bounced off a metal cylinder and sonar signals bounced off a roughly cylindrical rock. 208 objects are determined by a number of 60 attributes, numbers in the range 0 to 1, representing the energy within a particular frequency band, integrated over a certain period of time.

- **Wine** data is constituted from results of a chemical analysis of wines cultivated in the same region in Italy but delivered from 3 different cultivars. The analysis should determine from which cultivar comes the wine (3 classes). The available attributes represent 13 constituents found in each of the three types of wine. The classes are separable. This is a good data set for testing a new classifier, but not very challenging.

Appendix C

Alternative FDT growing method: residual fitting

This appendix presents the formulas behind an alternative FDT growing method, called residual fitting. It is a different way of growing a fuzzy decision tree that may be exploited in a future work. It is based on idea that at every new split of a node, we take into account the already developed parts of the tree.

C.1 Definitions

Suppose a partially developed tree with a complexity $K(t)$ (number of test nodes at moment t) and node S considered the current node to be further developed. We would like to minimize the error function [1]

$$E_S^r = \sum_{o \in S} \mu_S(o)[\mu_C(o) - \hat{\mu''}_C(o)]^2 \tag{C.1}$$

with

$$\hat{\mu''}_C(o) = \mu_S(o)\hat{\mu'}_C(o) + \sum_{j=1}^{K(t)} \mu_{S_{L_j}}(o)L_j \tag{C.2}$$

and

$$\hat{\mu'}_C(o) = \nu(a(o), \alpha, \beta)L_L + (1 - \nu(a(o), \alpha, \beta))L_R. \tag{C.3}$$

The notation $\hat{\mu''}_C(o)$ represents the membership degree to the target class estimated by the whole tree. The first term of eq. (C.2) represents the estimation of the membership degree given by the current node S development, whereas the second term of the equation represents the membership degree estimated by the rest of the tree, thus given by all the other terminal nodes (minus the current node to be split). The number of these terminal nodes minus the current node S to be split is $K(t)$.

Therefore, we may write the error function as

$$E_S^r = \sum_{o \in S} \mu_S(o)[\mu_C(o) - \mu_S(o)\hat{\mu'}_C(o) - \sum_{j=1}^{K(t)} \mu_{S_{L_j}}(o)L_j]^2. \tag{C.4}$$

[1] The index "r" comes from "residual".

We note the constant quantity

$$\mu_C(o) - \sum_{j=1}^{K(t)} \mu_{S_{L_j}}(o) L_j = \mu_{C_{residual}}(o) \tag{C.5}$$

that is an already known value at the moment t of the development of the tree, since both its terms are available at moment t. It represents the residual membership degree that has still to be explained by the splitting of node S. Thus, the procedure may be enunciated as follows.

Objective. Given a partially developed tree with a complexity $K(t)$ and node S to be developed, find attribute $a(\cdot)$, threshold α and width β together with successors labels L_L and L_R, so as to minimize the squared error function

$$E_S^r = \sum_{o \in S} \mu_S(o)[\mu_{C_{residual}}(o) - \mu_S(o)\hat{\mu'}_C(o)]^2, \tag{C.6}$$

with $\hat{\mu'}_C(o)$ given by the eq. (C.3).

Strategy. The local search is decomposed as in section 3.4.2:

- Searching for the attribute, split location and successors labels when $\beta = 0$ (crisp split).
- Fuzzification by FIBONACCI search and labeling.

We deduce further on all the necessary formulas in order to implement this kind of approach.

C.2 When $\beta = 0$

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

$$\max[1 - \frac{E_{S_L}^r}{E_S^r} - \frac{E_{S_R}^r}{E_S^r}]$$

where $E_S^r, E_{S_L}^r$ and $E_{S_R}^r$ are the squared error functions at nodes S, S_L (left successor of node S) and S_R (right successor of node S) respectively.

When $\beta = 0$, the two sets of objects S_L and S_R are mutually exclusive, the piecewise linear discriminator $\nu(\cdot)$ of figure 3.3 becomes crisp, and the error function E_S^r of eq. (C.6) can be written based on eq. (C.3) as

$$\begin{array}{rcl} E_S^r & = & \sum_{o \in S_L} \mu_S(o)[\mu_{C_{residual}}(o) - \mu_S(o)L_L]^2 + \sum_{o \in S_R} \mu_S(o)[\mu_{C_{residual}}(o) - \mu_S(o)L_R]^2 \\ & = & E_{S_L}^r + E_{S_R}^r. \end{array}$$

To minimize the error E_S^r translates thus in minimizing the two errors $E_{S_L}^r$ and $E_{S_R}^r$.
Therefore,

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

$$\begin{array}{rcl} -2\sum_{o \in S_L} \mu_S^2(o)[\mu_{C_{residual}}(o) - \mu_S(o)L_L] & = & 0 \\ -2\sum_{o \in S_R} \mu_S^2(o)[\mu_{C_{residual}}(o) - \mu_S(o)L_R] & = & 0, \end{array}$$

and finally, the two successors labels are estimated as

$$L_L = \frac{\sum_{o \in S_L} \mu_S^2(o)\mu_{C_{residual}}(o)}{\sum_{o \in S_L} \mu_S^3(o)}, \qquad L_R = \frac{\sum_{o \in S_R} \mu_S^2(o)\mu_{C_{residual}}(o)}{\sum_{o \in S_R} \mu_S^3(o)}, \tag{C.7}$$

Introducing these labels estimates in $E^r_{S_L}$ and $E^r_{S_R}$, we get the incremental formulas for computing the errors:

$$E_A = \sum_{o \in A} \mu_S(o)\mu^2_{C_{residual}}(o) - \frac{(\sum_{o \in A} \mu_S^2(o)\mu_{C_{residual}}(o))^2}{\sum_{o \in A} \mu_S^3(o)} \tag{C.8}$$

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.

When node S is the root node, $\mu_{C_{residual}}(o) = \mu_C(o)$ and if $\mu_{GS}(o) = 1$ for all objects $o \in GS$, the label of the root is

$$L_{root} = \frac{\sum_{o \in GS} \mu_C(o)}{N}$$

where $N = ||GS||$ is the size of the growing set.

C.3 Labeling

In order to optimize the successor labels at fixed α and β already found and settled, the error function of eq. (C.6) is minimized, thus we make

$$\frac{\partial E^r_S}{\partial L_L} = 0 \quad and \quad \frac{\partial E^r_S}{\partial L_R} = 0.$$

In conformity with eq. (C.3)

$$\begin{array}{rcl} -2\sum_{o \in S} \mu_S^2(o)\nu(a(o))\{\mu_{C_{residual}}(o) - \mu_S(o)[\nu(a(o))L_L + (1-\nu(a(o)))L_R]\} & = & 0 \\ -2\sum_{o \in S} \mu_S^2(o)(1-\nu(a(o)))\{\mu_{C_{residual}}(o) - \mu_S(o)[\nu(a(o))L_L + (1-\nu(a(o)))L_R]\} & = & 0. \end{array}$$

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

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

where all the terms generically noted $a(o), b(o), c(o), d(o)$ are sums computed in terms of $\mu_S(o), \mu_{C_{residual}}(o)$ and $\nu(a(o))$:

$$\begin{array}{ll} a(o) = \sum_{o \in S} \mu_S^3(o)\nu(a(o))^2 & b(o) = \sum_{o \in S} \mu_S^3(o)\nu(a(o))(1-\nu(a(o))) \\ c(o) = \sum_{o \in S} \mu_S^2(o)(1-\nu(a(o)))^2 & d(o) = -\sum_{o \in S} \mu_S^2(o)\mu_{C_{residual}}(o)\nu(a(o)) \\ e(o) = -\sum_{o \in S} \mu_S^2(o)\mu_{C_{residual}}(o)(1-\nu(a(o))). & \end{array}$$

C.4 Remarks

The formulas behind this approach are similar with the ones in the proposed growing procedure of chapter 3. The only difference comes from the fact that here we have an additional calculus: every time a new node has to be split, we have to compute the residual membership degree, quantity given by eq. (C.5), calculus linear in the tree complexity at the moment and in the growing set size.

Appendix D

Detailed experimental results

This appendix groups detailed results[1] *of the experiments of chapter 4.*

D.1 Results of section 4.2.2

Table D.1: Omib dataset

Method	Complexity	Error rate (%)	MSE	variance	bias
Omib, GS=500, fuzzy, piecewise linear discriminator, q = 25					
Pruned ULG	20.5±6.3	12.34±0.88	-	-	-
Pruned CART	67.0±10.3	11.91±1.00	0.040377±0.003272	0.028362	0.012015
Pruned C4.5	22.9±2.5	11.13±0.65	-	-	-
Full FDT	125.9±7.7	8.75±1.16	0.035587±0.004771	0.005840	0.029747
Pruned FDT	59.9±12.4	9.58±1.48	0.033649±0.004699	0.005618	0.028031
Ref. FDT(GS)	59.9±12.4	5.24±1.08	0.014376±0.004575	0.008339	0.006037
Backf. FDT(GS)	59.9±12.4	4.47±0.67	0.011391±0.002004	0.006442	0.004949
Ref. FDT(GS+PS)	59.9±12.4	4.62±1.18	0.011425±0.003887	0.005289	0.006136
Backf. FDT(GS+PS)	59.9±12.4	3.01±0.59	0.007383±0.002903	0.002863	0.004520
Omib, GS=500, fuzzy, Sigmoidal discriminator, q = 25					
Full FDT	128.3±5.3	8.93±0.79	0.034814±0.003417	-	-
Pruned FDT	55.7±15.2	10.05±1.26	0.033124±0.003016	-	-
Ref. FDT(GS+PS)	55.7±15.2	4.88±0.98	0.011726±0.003365	0.005018	0.006708
Backf. FDT(GS+PS)	55.7±15.2	3.73±0.89	0.008383±0.002529	0.003028	0.005353
Omib, GS=500, crisp, piecewise linear discriminator, q = 25					
Pruned ULG	20.5±6.3	12.34±0.88	-	-	-
Pruned CART	25.4±3.8	11.90±0.83	0.111309±0.007245	0.065340	0.045969
Pruned C4.5	22.9±2.5	11.13±0.65	-	-	-
Full FDT	141.0±13.6	9.84±1.13	0.082929±0.005236	0.014714	0.068215
Pruned FDT	48.5±11.8	10.25±1.37	0.080842±0.006303	0.012589	0.068253
Ref. FDT(GS)	48.5±11.8	7.83±1.57	0.066955±0.008783	0.025728	0.041227
Backf. FDT(GS)	48.5±11.8	8.86±1.43	0.071135±0.010418	0.029364	0.041771
Ref. FDT(GS+PS)	48.5±11.8	6.32±1.44	0.055686±0.007179	0.011976	0.043710
Backf. FDT(GS+PS)	48.5±11.8	5.27±0.70	0.050244±0.005054	0.008848	0.041396

[1] When a forest of trees is needed (i.e. for multiclass problems, CART and FDT methods), complexities displayed are averages over the forest.

Table D.2: Gaussian and twonorm datasets

Method	Complexity	Error rate (%)	MSE	variance	bias
Gaussian, GS=100, piecewise linear discriminator, q = 25					
Pruned ULG	3.3±1.9	14.04±1.96	-	-	-
Pruned CART	5.5±2.8	14.38±1.87	0.122229±0.020858	0.036018	0.086211
Pruned C4.5	2.4±0.8	14.76±2.63	-	-	-
Full FDT	90.6±21.3	13.08±1.07	0.099828±0.009928	0.013676	0.086151
Pruned FDT	22.4±15.6	12.22±0.55	0.092416±0.004461	0.008422	0.083994
Ref. FDT(GS)	22.4±15.6	13.01±1.23	0.111291±0.023816	0.026767	0.084524
Backf. FDT(GS)	22.4±15.6	13.25±1.24	0.107706±0.018706	0.023319	0.084387
Ref. FDT(GS+PS)	22.4±15.6	11.62±0.27	0.085439±0.002191	0.002128	0.083311
Backf. FDT(GS+PS)	22.4±15.6	11.53±0.15	0.084421±0.001152	0.001202	0.083219
Gaussian, GS=100, Sigmoidal discriminator, q = 25					
Full FDT	89.7±21.3	13.70±1.85	0.102877±0.010828	-	-
Pruned FDT	18.1±14.1	12.25±0.79	0.092203±0.005336	-	-
Ref. FDT(GS+PS)	18.1±14.1	11.72±0.23	0.086173±0.002893	0.002813	0.083360
Backf. FDT(GS+PS)	18.1±14.1	11.64±0.19	0.084836±0.001462	0.001272	0.083564
Twonorm, GS=300, piecewise linear discriminator, q = 25					
Pruned ULG	15.1±3.5	22.70±1.16	-	-	-
Pruned CART	18.4±2.6	22.48±0.93	0.213642±0.008960	0.139276	0.074366
Pruned C4.5	18.4±1.8	21.55±1.12	-	-	-
Full FDT	191.2±25.6	14.47±1.31	0.117589±0.005272	0.029138	0.088451
Pruned FDT	71.5±12.2	14.52±1.74	0.117116±0.006783	0.029075	0.088042
Ref. FDT(GS)	71.5±12.2	14.57±1.58	0.119712±0.011788	0.064911	0.054801
Backf. FDT(GS)	71.5±12.2	16.03±1.87	0.126719±0.020575	0.059357	0.067362
Ref. FDT(GS+PS)	71.5±12.2	11.29±1.79	0.091095±0.008910	0.039209	0.051886
Backf. FDT(GS+PS)	71.5±12.2	7.94±1.59	0.075440±0.007156	0.027264	0.048176
Twonorm, GS=300, Sigmoidal discriminator, q = 25					
Full FDT	184.4±26.6	15.36±1.37	0.119625±0.005264	-	-
Pruned FDT	73.9±10.1	15.65±1.43	0.120036±0.006589	-	-
Ref. FDT(GS+PS)	73.9±10.1	11.31±1.72	0.090106±0.008350	0.038454	0.051652
Backf. FDT(GS+PS)	73.9±10.1	11.98±2.03	0.082712±0.010929	0.033158	0.059554

Table D.3: Waveform and satellite datasets

Method	Complexity	Error rate (%)
Waveform, GS=300, piecewise linear discriminator, q = 25		
Pruned ULG	12.6±4.8	29.97±1.37
Pruned CART	12.6±2.0	29.39±2.02
Pruned C4.5	26.1±2.1	28.91±1.23
Full FDT	132.7±14.4	23.34±1.52
Pruned FDT	30.0±9.8	22.06±1.71
Ref. FDT(GS+PS)	30.0±9.8	19.70±1.46
Backf. FDT(GS+PS)	30.0±9.8	19.08±1.09
Waveform, GS=300, Sigmoidal discriminator, q = 25		
Full FDT	133.4±11.1	22.99±1.47
Pruned FDT	37.7±8.8	21.28±1.43
Ref. FDT(GS+PS)	37.7±8.8	19.23±1.06
Backf. FDT(GS+PS)	37.7±8.8	19.45±1.21
Satellite, GS=500, piecewise linear discriminator, q = 25		
Pruned ULG	11.1±4.8	20.60±1.43
Pruned CART	8.8±1.5	21.52±1.21
Pruned C4.5	36.0±3.9	20.36±1.02
Full FDT	74.4±5.8	16.43±1.09
Pruned FDT	11.4±1.8	16.99±0.90
Ref. FDT(GS+PS)	11.4±1.8	16.08±0.73
Backf. FDT(GS+PS)	11.4±1.8	15.73±0.80
Satellite, GS=500, Sigmoidal discriminator, q = 25		
Full FDT	70.4±6.7	16.60±1.06
Pruned FDT	12.2±2.0	16.33±0.96
Ref. FDT(GS+PS)	12.2±2.0	15.80±0.73
Backf. FDT(GS+PS)	12.2±2.0	15.54±0.75

Table D.4: Pendigits and dig44 datasets

Method	Complexity	Error rate (%)
Pendigits, GS=500, piecewise linear discriminator, q = 25		
Pruned ULG	32.1±7.4	19.48±1.91
Pruned CART	8.5±0.8	23.22±1.90
Pruned C4.5	32.0±2.8	19.58±1.80
Full FDT	29.4±2.1	17.01±1.60
Pruned FDT	12.3±1.1	16.98±1.65
Ref. FDT(GS+PS)	12.3±1.1	15.18±1.41
Backf. FDT(GS+PS)	12.3±1.1	13.22±1.26
Pendigits, GS=500, Sigmoidal discriminator, q = 25		
Full FDT	27.1±2.6	16.95±1.43
Pruned FDT	12.1±1.3	17.30±1.52
Ref. FDT(GS+PS)	12.1±1.3	15.66±1.32
Backf. FDT(GS+PS)	12.1±1.3	14.67±1.38
Dig44, GS=500, piecewise linear discriminator, q = 25		
Pruned ULG	38.7±11.2	30.27±1.49
Pruned CART	10.4±0.8	33.84±1.23
Pruned C4.5	55.4±4.3	28.86±1.77
Full FDT	66.3±4.1	25.85±1.32
Pruned FDT	17.5±2.3	24.31±1.16
Ref. FDT(GS+PS)	17.5±2.3	21.10±1.02
Backf. FDT(GS+PS)	17.5±2.3	19.08±1.21
Dig44, GS=500, Sigmoidal discriminator, q = 25		
Full FDT	65.1±5.0	27.03±1.47
Pruned FDT	16.7±2.3	25.22±1.35
Ref. FDT(GS+PS)	16.7±2.3	21.71±1.30
Backf. FDT(GS+PS)	16.7±2.3	21.13±1.13

D.2 Results of section 4.2.4

Table D.5: Comparing mean squared error (MSE), global variance and bias between crisp and fuzzy output for omib data, $q = 25, GS = 500$, piecewise linear discriminator

Method	fuzzy			crisp		
	MSE	variance	bias	MSE	variance	bias
Pruned CART	0.040377	0.028362	0.012015	0.111309	0.065340	0.045969
Full FDT	0.035587	0.005840	0.029747	0.082929	0.014714	0.068215
Pruned FDT	0.033649	0.005618	0.028031	0.080842	0.012589	0.068253
Ref. FDT (GS)	0.014376	0.008339	0.006037	0.066955	0.025728	0.041227
Backf. FDT (GS)	0.011391	0.006442	0.004949	0.071135	0.029364	0.041771
Ref. FDT (GS+PS)	0.011425	0.005289	0.006136	0.055686	0.011976	0.043710
Backf. FDT (GS+PS)	0.007383	0.002863	0.004520	0.050244	0.008848	0.041396

Table D.6: Summarized comparison: mean squared error (MSE), global variance and bias of different algorithms on two-class databases

Method	Gaussian GS=100			Twonorm GS=300			Omib - crisp GS=500		
	MSE	variance	bias	MSE	variance	bias	MSE	variance	bias
q = 25									
Pruned CART	0.122229	0.036018	0.086211	0.213642	0.139276	0.074366	0.111309	0.065340	0.045969
q = 25, Piecewise linear discriminator									
Full FDT	0.099828	0.013676	0.086151	0.117589	0.029138	0.088451	0.082929	0.014714	0.068215
Pruned FDT	0.092416	0.008422	0.083994	0.117116	0.029075	0.088042	0.080842	0.012589	0.068253
Ref. FDT (GS)	0.111291	0.026767	0.084524	0.119712	0.064911	0.054801	0.066955	0.025728	0.041227
Backf. FDT (GS)	0.107706	0.023319	0.084387	0.126719	0.059357	0.067362	0.071135	0.029364	0.041771
Ref. FDT (GS+PS)	0.085439	0.002128	0.083311	0.091095	0.039209	0.051886	0.055686	0.011976	0.043710
Backf. FDT (GS+PS)	0.084421	0.001202	0.083219	0.075440	0.027264	0.048176	0.050244	0.008848	0.041396
q = 50, (taken from [Geu02])									
Full ULG	0.1757	0.0909	0.0848	0.2159	0.1507	0.0652	0.1213	0.0789	0.0424
Pruned ULG ($\sigma = 0$)	0.1199	0.0337	0.0862	0.2099	0.1445	0.0654	0.1161	0.0698	0.0463
Dual P&C	0.1165	0.0171	0.0994	0.1283	0.0199	0.1084	0.0810	0.0104	0.0706
Bagging	0.1117	0.0265	0.0852	0.0827	0.0155	0.0670	0.0611	0.0144	0.0466
Boosting	0.1196	0.0269	0.0927	0.0828	0.0094	0.0734	0.0627	0.0079	0.0547
Extra-tree avg.	0.1091	0.0250	0.0841	0.0693	0.0091	0.0600	0.0525	0.0090	0.0434
Bag. + dual P&C	0.1101	0.0105	0.0995	0.1213	0.0044	0.1170	0.0846	0.0035	0.0811

Table D.7: Comparing mean squared error (MSE), global variance and bias between piecewise linear and sigmoidal discriminators, for 3 databases, $q = 25$

	Gaussian GS=100			Twonorm GS=300			Omib - fuzzy GS=500		
Method	MSE	variance	bias	MSE	variance	bias	MSE	variance	bias
q = 25, Piecewise linear discriminator									
Ref. FDT (GS+PS)	0.085439	0.002128	0.083311	0.091095	0.039209	0.051886	0.011425	0.005289	0.006136
Backf. FDT (GS+PS)	0.084421	0.001202	0.083219	0.075440	0.027264	0.048176	0.007383	0.002863	0.004520
q = 25, Sigmoidal Discriminator									
Ref. FDT (GS+PS)	0.086173	0.002813	0.083360	0.090106	0.038454	0.051652	0.011726	0.005018	0.006708
Backf. FDT (GS+PS)	0.084836	0.001272	0.083564	0.092712	0.033158	0.059554	0.008383	0.003028	0.005355

Table D.8: Variance and bias function of tree complexity on omib dataset

Cardinality th.	Complexity	Error rate (%)	MSE	variance	bias
GS=500, full FDT, piecewise linear discriminator, fuzzy class, q = 25					
300	1.3±0.5	21.23±0.89	0.081921±0.004688	0.003315	0.078606
200	3.6±1.0	20.73±2.88	0.082556±0.011139	0.009213	0.073343
100	11.4±2.2	17.5±1.74	0.071760±0.008020	0.008738	0.063022
50	17.5±2.1	14.81±1.92	0.057591±0.008047	0.007937	0.049654
25	31.4±3.3	12.35±1.74	0.049430±0.004998	0.006240	0.043227
20	38.4±3.9	11.77±1.68	0.047888±0.004770	0.006586	0.041303
15	51.7±4.7	11.30±1.65	0.044824±0.004401	0.006536	0.038288
10	73.5±6.2	10.14±1.68	0.040803±0.005158	0.006229	0.034574
5	125.9±7.7	8.75±1.16	0.035587±0.004771	0.005840	0.029747
2.5	222.0±12.1	7.75±0.88	0.029914±0.003805	0.005253	0.024661
1	454.8±27.9	6.70±0.60	0.024116±0.0027400	0.004803	0.019313
GS=500, full FDT, piecewise linear discriminator, crisp class, q = 25					
300	1.5±0.6	21.50±1.04	0.145855±0.007587	0.005926	0.139929
200	4.2±1.6	20.83±3.85	0.144878±0.019906	0.015271	0.129608
100	14.0±3.6	17.73±2.40	0.130229±0.010529	0.015858	0.114371
50	21.4±4.6	15.32±1.91	0.113402±0.006803	0.014346	0.099056
25	43.3±6.3	13.30±1.74	0.102236±0.006700	0.013780	0.088456
20	53.0±6.9	12.70±1.33	0.098498±0.006567	0.014680	0.083818
15	64.4±7.1	11.48±1.62	0.093902±0.007033	0.014536	0.079366
10	85.8±8.3	10.77±1.30	0.089148±0.005885	0.014839	0.074309
5	141.0±13.6	9.84±1.13	0.082929±0.005236	0.014714	0.068215
2.5	233.3±23.9	9.02±1.04	0.076695±0.004758	0.015004	0.061691
1	432.7±53.5	8.28±0.86	0.069246±0.004939	0.015538	0.053707

Table D.9: Variance and bias function of tree complexity on twonorm dataset

Cardinality th.	Complexity	Error rate (%)	MSE	variance	bias
GS=300, full FDT, piecewise linear discriminator, q = 25					
200	1.1±0.3	32.52±1.18	0.212978±0.016282	0.046418	0.166560
100	3.8±0.7	27.17±2.77	0.184755±0.014418	0.036137	0.148617
50	17.8±9.0	24.81±2.74	0.173557±0.010361	0.032784	0.140773
30	37.7±8.3	21.62±2.43	0.160360±0.011218	0.029417	0.130943
25	43.6±7.3	20.46±2.46	0.156002±0.010739	0.028644	0.127358
20	49.3±6.3	19.18±2.19	0.149930±0.009436	0.029424	0.120506
15	59.4±7.2	18.45±1.90	0.144875±0.008585	0.029714	0.115161
10	82.6±9.9	17.53±1.81	0.137649±0.007517	0.030071	0.107578
5	138.7±16.7	15.70±1.40	0.125718±0.006735	0.029393	0.096325
3	191.2±25.6	14.47±1.31	0.117589±0.005272	0.029138	0.088451
1	342.0±62.7	12.86±1.47	0.104351±0.006189	0.029579	0.074771
GS=300, full CART, q = 25					
-	1	33.80±1.59	0.228124±0.006319	0.047388	0.180737
-	2	30.62±1.58	0.217035±0.006996	0.066492	0.150543
-	3	28.97±1.53	0.209570±0.005996	0.077116	0.132454
-	4	28.05±1.02	0.205651±0.007626	0.084943	0.120708
-	5	26.98±0.94	0.204316±0.007200	0.091838	0.112478
-	10	24.09±1.10	0.205032±0.008249	0.116284	0.088748
-	15	23.14±0.94	0.207553±0.009478	0.128785	0.078767
-	20	22.49±0.99	0.212965±0.009985	0.139946	0.073019
-	24	22.24±0.82	0.217301±0.009097	0.146466	0.070835
GS=300, Extra-tree avg., q = 25					
-	146	-	0.1928	0.0026	0.1902
-	190	-	0.1789	0.0033	0.1755
-	233	-	0.1688	0.0038	0.1650
-	255	-	0.1644	0.0039	0.1605
-	276	-	0.1608	0.0041	0.1567
-	303	-	0.1568	0.0043	0.1525
-	370	-	0.1485	0.0046	0.1438
-	466	-	0.1397	0.0049	0.1348
-	574	-	0.1322	0.0051	0.1272
-	723	-	0.1246	0.0050	0.1196
-	1035	-	0.1133	0.0052	0.1081
-	1861	-	0.0944	0.0051	0.0893
-	3713	-	0.0766	0.0050	0.0717
-	5837	-	0.0698	0.0051	0.0647
-	8654	-	0.0666	0.0054	0.0612

Table D.10: Variance and bias function of growing set size on omib dataset

GS size	Complexity	Error rate (%)	MSE	variance	bias
Backfitted FDT, piecewise linear discriminator, fuzzy class, q = 20					
50	41.7±17.9	14.37±2.63	0.057809±0.017519	0.035418	0.022391
100	48.3±17.6	11.24±1.99	0.039994±0.010000	0.025943	0.014051
200	59.0±13.9	8.19±1.86	0.024883±0.006656	0.015248	0.009635
300	64.8±11.6	6.38±1.52	0.017762±0.004718	0.011307	0.006455
400	67.4±14.4	5.32±0.91	0.013374±0.002299	0.007496	0.005879
500	70.0±8.6	4.62±1.14	0.011481±0.003015	0.006509	0.004972
750	60.2±17.0	4.01±0.77	0.010345±0.002824	0.005207	0.005139
1000	63.5±13.6	3.55±1.20	0.008940±0.004676	0.004420	0.004520
1500	56.9±17.2	2.98±0.55	0.007247±0.001932	0.002841	0.004405

D.3 Results of section 4.2.5

Table D.11: Cutting point parameter variance at two levels, C=3, for omib fuzzy output data. Comparison between methods. $q = 20$.

GS size	Method	$\sigma_{\text{root}}(Pu)$	$\sigma_{\text{left/yes}}(Qu)$	$\sigma_{\text{right/no}}(Qu)$	$card_{\text{left/yes}}$	$card_{\text{right/no}}$
200	ULG	79	361	328	122	78
	CART	46	196	261	122	78
	FDT(R)	46	217	237	121	79
	FDT(B)	369	240	228	133	67
300	ULG	62	226	246	182	118
	CART	36	164	203	185	115
	FDT(R)	36	190	173	184	116
	FDT(B)	313	153	227	197	103
400	ULG	57	237	217	234	166
	CART	42	148	165	236	164
	FDT(R)	42	168	124	236	164
	FDT(B)	185	171	173	269	131
500	ULG	40	209	276	296	204
	CART	35	168	162	302	198
	FDT(R)	35	145	150	303	197
	FDT(B)	237	130	166	364	136
750	ULG	59	280	219	470	280
	CART	41	192	168	435	315
	FDT(R)	41	128	169	435	315
	FDT(B)	225	106	157	515	235
1000	ULG	34	199	234	585	415
	CART	23	122	129	600	400
	FDT(R)	23	114	129	600	400
	FDT(B)	98	96	147	646	354
1500	ULG	25	212	167	871	629
	CART	18	81	134	874	626
	FDT(R)	18	93	141	873	627
	FDT(B)	149	81	115	951	549

Table D.12: Cutting point parameter variance at two levels, C=3, for omib crisp output data. Comparison between methods. $q = 20$.

GS size	Method	$\sigma_{\text{root}}(Pu)$	$\sigma_{\text{left/yes}}(Qu)$	$\sigma_{\text{right/no}}(Qu)$	$card_{\text{left/yes}}$	$card_{\text{right/no}}$
200	ULG	79	361	328	122	78
	CART	52	186	320	131	69
	FDT(R)	52	186	243	130	70
	FDT(B)	219	176	268	140	60
300	ULG	62	226	246	182	118
	CART	39	181	218	198	101
	FDT(R)	39	193	212	198	102
	FDT(B)	159	154	209	213	87
400	ULG	57	237	217	234	166
	CART	42	183	173	241	159
	FDT(R)	42	183	119	240	160
	FDT(B)	207	144	171	265	135
500	ULG	40	209	276	296	204
	CART	35	175	168	304	196
	FDT(R)	35	175	148	306	194
	FDT(B)	280	128	126	343	157
750	ULG	59	280	219	470	280
	CART	42	178	188	481	269
	FDT(R)	42	167	150	482	268
	FDT(B)	276	107	229	533	217
1000	ULG	34	199	234	585	415
	CART	29	171	129	608	392
	FDT(R)	29	156	146	607	393
	FDT(B)	168	115	90	667	333
1500	ULG	25	212	167	871	629
	CART	18	119	120	908	593
	FDT(R)	18	110	150	905	595
	FDT(B)	212	106	160	995	505

Table D.13: Cutting point parameter variance at two levels, C=3, for gaussian data. Comparison between methods. $q = 20$.

GS size	Method	$\sigma_{\text{root}}(A1)$	$\sigma_{\text{left/yes}}(A1)$	$\sigma_{\text{right/no}}(A2)$	$card_{\text{left/yes}}$	$card_{\text{right/no}}$
200	ULG	0.429	0.532	0.907	112	88
	CART	0.258	0.373	0.281	106	94
	FDT(R)	0.258	0.294	0.226	104	96
	FDT(B)	0.277	0.515	0.252	91	109
300	ULG	0.415	0.437	0.328	156	144
	CART	0.298	0.346	0.251	156	144
	FDT(R)	0.298	0.289	0.221	155	145
	FDT(B)	0.262	1.157	0.134	127	173
400	ULG	0.343	0.278	0.285	220	180
	CART	0.179	0.245	0.235	216	184
	FDT(R)	0.179	0.268	0.197	213	187
	FDT(B)	0.632	0.560	0.179	193	207
500	ULG	0.344	0.259	0.368	273	227
	CART	0.279	0.181	0.316	266	234
	FDT(R)	0.279	0.260	0.236	264	236
	FDT(B)	1.915	0.919	0.367	231	269
750	ULG	0.295	0.413	0.268	422	328
	CART	0.198	0.202	0.230	405	345
	FDT(R)	0.198	0.272	0.208	401	349
	FDT(B)	0.104	1.144	0.125	327	423
1000	ULG	0.211	0.350	0.242	572	428
	CART	0.199	0.228	0.121	536	464
	FDT(R)	0.199	0.283	0.152	530	470
	FDT(B)	0.376	1.349	0.093	456	544
1500	ULG	0.249	0.315	0.212	853	647
	CART	0.168	0.266	0.180	788	712
	FDT(R)	0.168	0.106	0.120	777	723
	FDT(B)	0.053	0.496	0.063	659	841

D.4 Results of section 4.2.6

Table D.14: Gaussian dataset, piecewise linear discriminator, 5 trees (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Pruned ULG	3.2±1.2	13.75±1.8	-
	Pruned CART	10.0±4.2	15.59±1.65	0.143224±0.018703
	Pruned C4.5	2.8±1.5	16.52±4.90	-
	Full FDT	100.4±12.5	13.26±1.0	0.099158±0.006498
	Pruned FDT	26.8±23.4	12.55±0.78	0.092943±0.008554
	Refitted FDT	26.8±23.4	11.93±1.04	0.087285±0.006264
	Backfitted FDT	26.8±23.4	12.02±0.96	0.087480±0.006212
500	Pruned ULG	3.0±1.3	13.01±0.47	-
	Pruned CART	9.2±5.0	13.24±0.88	0.101492±0.008265
	Pruned C4.5	3.2±1.9	13.56±1.26	-
	Full FDT	105.2±8.2	11.90±0.28	0.088542±0.001594
	Pruned FDT	24.8±12.4	11.87±0.34	0.086268±0.001942
	Refitted FDT	24.8±12.4	11.52±0.16	0.084117±0.000930
	Backfitted FDT	24.8±12.4	11.53±0.11	0.083682±0.000540
1000	Pruned ULG	7.0±1.7	12.50±0.47	-
	Pruned CART	11.6±10.3	12.51±0.77	0.096027±0.006252
	Pruned C4.5	4.0±1.9	12.64±0.61	-
	Full FDT	125.4±7.1	12.45±0.46	0.090521±0.002850
	Pruned FDT	30.4±15.4	12.12±0.42	0.086891±0.002348
	Refitted FDT	30.4±15.4	11.54±0.21	0.083810±0.000750
	Backfitted FDT	30.4±15.4	11.59±0.12	0.084224±0.000609
2000	Pruned ULG	8.6±2.9	12.11±0.66	-
	Pruned CART	37.6±38.6	12.76±1.25	0.100917±0.014391
	Pruned C4.5	5.6±1.6	12.06±0.64	-
	Full FDT	114.4±3.4	11.75±0.19	0.088016±0.003156
	Pruned FDT	38.0±12.3	11.49±0.20	0.086229±0.001996
	Refitted FDT	38.0±12.3	11.38±0.09	0.083478±0.000291
	Backfitted FDT	38.0±12.3	11.52±0.08	0.083661±0.000323

Table D.15: Gaussian dataset, Sigmoidal discriminator (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Full FDT	100.4±17.0	14.86±1.84	0.106428±0.011206
	Pruned FDT	25.2±22.8	12.85±0.93	0.093182±0.004763
	Refitted FDT	25.2±22.8	11.66±0.15	0.084858±0.000896
	Backfitted FDT	25.2±22.8	11.61±0.17	0.084464±0.000511
500	Full FDT	116.2±11.7	12.26±0.45	0.090297±0.003663
	Pruned FDT	26.6±8.8	11.56±0.15	0.084290±0.000485
	Refitted FDT	26.6±8.8	11.54±0.10	0.083945±0.000648
	Backfitted FDT	26.6±8.8	11.56±0.05	0.083767±0.000475
1000	Full FDT	137.0±5.3	12.00±0.36	0.089305±0.001332
	Pruned FDT	10.6±6.0	11.81±0.30	0.086576±0.002753
	Refitted FDT	10.6±6.0	11.49±0.11	0.083784±0.000362
	Backfitted FDT	10.6±6.0	11.62±0.16	0.084137±0.000428
2000	Full FDT	124.6±5.9	11.84±0.28	0.086708±0.000669
	Pruned FDT	24.8±15.4	11.78±0.35	0.085184±0.002120
	Refitted FDT	24.8±15.4	11.50±0.06	0.083620±0.000455
	Backfitted FDT	24.8±15.4	11.53±0.06	0.083565±0.000261

Table D.16: Gaussian dataset, piecewise linear discriminator, variant for pruning, variant for Fibonacci search (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Full FDT	100.6±15.7	13.46±1.33	0.100062±0.006718
	Pruned FDT	23.2±17.1	12.13±0.37	0.090426±0.002882
	Refitted FDT	23.2±17.1	11.61±0.29	0.084995±0.001483
	Backfitted FDT	23.2±17.1	11.52±0.08	0.084354±0.000473
500	Full FDT	104.4±5.7	12.34±0.33	0.089057±0.002731
	Pruned FDT	34.2±16.8	12.09±0.62	0.087921±0.003639
	Refitted FDT	34.2±16.8	11.45±0.11	0.084037±0.000571
	Backfitted FDT	34.2±16.8	11.62±0.14	0.084134±0.000816
1000	Full FDT	124.6±8.1	12.42±0.43	0.090772±0.002059
	Pruned FDT	32.4±11.4	12.25±0.62	0.088639±0.004064
	Refitted FDT	2.4±11.4	11.52±0.14	0.083817±0.000598
	Backfitted FDT	2.4±11.4	11.66±0.13	0.084099±0.000370
2000	Full FDT	116.6±5.7	12.18±0.18	0.089668±0.002680
	Pruned FDT	26.2±20.4	11.76±0.25	0.086387±0.003012
	Refitted FDT	26.2±20.4	11.51±0.13	0.083873±0.000537
	Backfitted FDT	26.2±20.4	11.54±0.12	0.083600±0.000547

Table D.17: Twonorm dataset, piecewise linear discriminator (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Pruned ULG	7.0±1.5	26.09±1.21	-
	Pruned CART	6.6±1.5	26.42±1.34	0.251852±0.006024
	Pruned C4.5	6.6±1.0	26.08±1.60	-
	Full FDT	86.2±25.0	20.53±2.18	0.155840±0.019140
	Pruned FDT	41.2±8.7	21.54±2.64	0.161472±0.022632
	Refitted FDT	41.2±8.7	17.28±2.99	0.127850±0.019042
	Backfitted FDT	41.2±8.7	9.41±2.3	0.085790±0.011055
300	Pruned ULG	14.2±4.0	22.83±0.60	-
	Pruned CART	18.8±1.9	22.28±1.49	0.210557±0.009813
	Pruned C4.5	18.2±1.8	21.06±0.71	-
	Full FDT	190.4±16.6	14.73±1.77	0.118990±0.005083
	Pruned FDT	68.2±9.5	13.73±1.67	0.116111±0.005412
	Refitted FDT	68.2±9.5	10.24±1.57	0.085234±0.007538
	Backfitted FDT	68.2±9.5	6.70±1.28	0.069981±0.004581
1000	Pruned ULG	35.0±14.1	20.85±0.64	-
	Pruned CART	53.8±7.1	19.94±0.67	0.189428±0.005099
	Pruned C4.5	53.4±2.9	19.16±0.64	-
	Full FDT	250.0±5.6	9.20±0.65	0.120295±0.003591
	Pruned FDT	76.0±10.5	9.69±0.74	0.116475±0.003200
	Refitted FDT	76.0±10.5	7.43±0.64	0.069262±0.002793
	Backfitted FDT	76.0±10.5	5.16±0.58	0.060791±0.003332
2000	Pruned ULG	71.0±12.3	18.42±1.77	-
	Pruned CART	83.2±9.9	17.69±0.59	0.163790±0.005254
	Pruned C4.5	98.4±6.9	17.30±0.29	-
	Full FDT	239.6±11.7	7.38±0.27	0.121386±0.002703
	Pruned FDT	90.2±11.5	7.98±0.80	0.115621±0.002802
	Refitted FDT	90.2±11.5	8.92±0.56	0.064139±0.004729
	Backfitted FDT	90.2±11.5	3.59±0.78	0.054419±0.001710

Table D.18: Twonorm dataset, Sigmoidal discriminator (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Full FDT	71.0±20.2	22.35±2.53	0.169175±0.030437
	Pruned FDT	27.8±11.8	22.66±2.56	0.167569±0.027244
	Refitted FDT	27.8±11.8	19.12±3.66	0.135786±0.022830
	Backfitted FDT	27.8±11.8	11.51±4.43	0.096017±0.021544
300	Full FDT	183.2±16.3	15.41±1.43	0.120041±0.005616
	Pruned FDT	73.2±9.7	14.75±1.03	0.118158±0.004711
	Refitted FDT	73.2±9.7	10.08±1.07	0.085281±0.007493
	Backfitted FDT	73.2±9.7	9.07±2.57	0.081849±0.014575
1000	Full FDT	242.4±9.3	9.18±1.44	0.130418±0.005404
	Pruned FDT	87.2±20.9	9.81±1.32	0.124485±0.004017
	Refitted FDT	87.2±20.9	7.01±0.53	0.067511±0.003163
	Backfitted FDT	87.2±20.9	8.41±0.90	0.072847±0.002995
2000	Full FDT	202.8±8.7	7.27±0.43	0.139992±0.003846
	Pruned FDT	84.4±22.0	8.15±1.25	0.127413±0.002462
	Refitted FDT	84.4±22.0	5.79±0.49	0.062246±0.003305
	Backfitted FDT	84.4±22.0	5.01±0.68	0.059127±0.001433

Table D.19: Twonorm dataset, piecewise linear discriminator, variant for Fibonacci search ($q = 5$)

GS size	Method	Complexity	Error rate (%)	MSE
100	Full FDT	76.6±24.2	20.69±2.77	0.156773±0.023468
	Pruned FDT	31.2±10.3	20.98±2.76	0.159844±0.023210
	Refitted FDT	31.2±10.3	19.02±2.86	0.134827±0.019208
	Backfitted FDT	31.2±10.3	10.60±2.63	0.092847±0.013794
300	Full FDT	183.8±10.1	15.25±1.89	0.117626±0.005382
	Pruned FDT	73.6±14.0	14.77±1.43	0.116215±0.004206
	Refitted FDT	73.6±14.0	11.24±1.27	0.090399±0.006824
	Backfitted FDT	73.6±14.0	7.94±2.32	0.075101±0.009783
1000	Full FDT	252.6±6.0	9.57±0.95	0.119765±0.004666
	Pruned FDT	71.6±13.5	10.40±0.74	0.116169±0.004305
	Refitted FDT	71.6±13.5	7.87±0.74	0.072388±0.004044
	Backfitted FDT	71.6±13.5	5.99±0.95	0.065235±0.003893
2000	Full FDT	240.0±9.3	7.05±0.68	0.121595±0.000703
	Pruned FDT	82.0±9.5	7.40±1.13	0.114333±0.002303
	Refitted FDT	82.0±9.5	6.14±0.78	0.063949±0.005544
	Backfitted FDT	82.0±9.5	3.72±0.62	0.054027±0.002289

Table D.20: Twonorm dataset, piecewise linear discriminator, refitting during pruning (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Full FDT	86.2±25.0	20.53±2.18	0.155840±0.019140
	Pruned FDT	24.2±7.7	22.20±1.35	0.180243±0.014983
	Refitted FDT	24.2±7.7	19.35±2.42	0.138136±0.015282
	Backfitted FDT	24.2±7.7	10.52±1.93	0.093615±0.008964
300	Full FDT	190.4±16.6	14.73±1.77	0.118990±0.005083
	Pruned FDT	38.2±4.7	13.63±2.62	0.106116±0.013861
	Refitted FDT	38.2±4.7	11.79±2.41	0.094950±0.010297
	Backfitted FDT	38.2±4.7	6.73±1.40	0.072660±0.006494
1000	Full FDT	250.0±5.6	9.20±0.65	0.120295±0.003591
	Pruned FDT	69.4±3.8	8.04±0.47	0.073622±0.000562
	Refitted FDT	69.4±3.8	7.38±0.46	0.070697±0.000911
	Backfitted FDT	69.4±3.8	5.35±0.78	0.061928±0.002797
2000	Full FDT	239.6±11.7	7.38±0.27	0.121386±0.002703
	Pruned FDT	93.2±10.0	5.91±0.77	0.064054±0.004768
	Refitted FDT	93.2±10.0	5.74±0.72	0.063158±0.004569
	Backfitted FDT	93.2±10.0	4.29±0.32	0.055281±0.000799

Table D.21: Omib dataset, piecewise linear discriminator (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Pruned ULG	3.0±2.1	19.74±2.10	-
	Pruned CART	15.0±2.0	17.18±1.10	0.073231±0.009916
	Pruned C4.5	6.0±0.6	18.18±1.25	-
	Full FDT	99.4±5.4	12.39±0.91	0.044148±0.002615
	Pruned FDT	51.8±9.1	12.82±0.92	0.043770±0.004212
	Refitted FDT	51.8±9.1	7.67±1.83	0.021486±0.006517
	Backfitted FDT	51.8±9.1	4.62±2.97	0.014460±0.008941
500	Pruned ULG	21.4±8.5	12.13±0.56	-
	Pruned CART	70.4±12.9	11.13±0.87	0.038129±0.003107
	Pruned C4.5	20.8±1.3	11.60±0.31	-
	Full FDT	127.8±3.5	8.85±2.46	0.034983±0.005563
	Pruned FDT	64.8±14.1	9.62±2.22	0.032965±0.006077
	Refitted FDT	64.8±14.1	4.21±1.40	0.010984±0.004943
	Backfitted FDT	64.8±14.1	2.76±0.49	0.006151±0.002318
1000	Pruned ULG	27.0±7.4	10.35±0.41	-
	Pruned CART	122.4±13.0	9.34±0.51	0.028945±0.001428
	Pruned C4.5	36.8±2.3	9.82±0.41	-
	Full FDT	130.2±4.4	8.01±1.30	0.034715±0.002527
	Pruned FDT	64.4±3.9	8.73±1.40	0.032032±0.003248
	Refitted FDT	64.4±3.9	3.84±0.68	0.009860±0.002083
	Backfitted FDT	64.4±3.9	2.73±0.38	0.005746±0.001066
2000	Pruned ULG	53.8±10.1	9.22±0.49	-
	Pruned CART	213.6±28.5	8.04±0.46	0.022690±0.001094
	Pruned C4.5	61.8±4.4	8.30±0.52	-
	Full FDT	132.0±6.9	7.77±0.60	0.035325±0.000857
	Pruned FDT	62.2±17.8	8.78±0.29	0.032424±0.001117
	Refitted FDT	62.2±17.8	3.68±1.84	0.009506±0.005888
	Backfitted FDT	62.2±17.8	2.40±0.72	0.005519±0.002783

Table D.22: Omib dataset, Sigmoidal discriminator (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Full FDT	91.2±5.6	15.03±2.48	0.051611±0.009126
	Pruned FDT	47.2±5.6	15.24±2.47	0.053054±0.010686
	Refitted FDT	47.2±5.6	10.08±1.07	0.028325±0.004682
	Backfitted FDT	47.2±5.6	6.74±2.74	0.018126±0.007678
500	Full FDT	126.8±3.1	8.36±1.16	0.033147±0.002279
	Pruned FDT	63.8±20.4	9.41±1.60	0.031427±0.003633
	Refitted FDT	63.8±20.4	4.64±1.20	0.010701±0.003447
	Backfitted FDT	63.8±20.4	3.33±0.68	0.006940±0.001815
1000	Full FDT	133.4±3.8	7.80±0.39	0.033603±0.001464
	Pruned FDT	47.6±11.4	10.15±1.08	0.033171±0.002943
	Refitted FDT	47.6±11.4	5.76±1.62	0.014369±0.005444
	Backfitted FDT	47.6±11.4	3.86±0.81	0.009757±0.002646
2000	Full FDT	137.8±3.7	8.08±0.41	0.035444±0.001307
	Pruned FDT	58.8±15.4	9.09±0.41	0.031467±0.001380
	Refitted FDT	58.8±15.4	4.22±0.90	0.009611±0.002804
	Backfitted FDT	58.8±15.4	4.03±1.55	0.008783±0.004250

Table D.23: Omib dataset, piecewise linear discriminator, variant for pruning (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Full FDT	99.4±5.4	12.39±0.91	0.044148±0.002615
	Pruned FDT	69.8±13.2	12.36±0.65	0.043955±0.002748
	Refitted FDT	69.8±13.2	6.50±0.80	0.016974±0.002588
	Backfitted FDT	69.8±13.2	3.12±0.42	0.008154±0.001821
500	Full FDT	127.8±3.5	8.85±2.46	0.034983±0.005563
	Pruned FDT	60.0±8.4	8.87±2.67	0.032232±0.006719
	Refitted FDT	60.0±8.4	4.45±1.61	0.011066±0.005657
	Backfitted FDT	60.0±8.4	2.65±0.49	0.006008±0.002645
1000	Full FDT	130.2±4.4	8.01±1.30	0.034715±0.002527
	Pruned FDT	55.6±14.9	8.63±1.30	0.031760±0.002762
	Refitted FDT	55.6±14.9	4.14±0.67	0.010952±0.001672
	Backfitted FDT	55.6±14.9	2.97±0.40	0.006912±0.000836
2000	Full FDT	132.0±6.7	7.77±0.60	0.035325±0.000857
	Pruned FDT	49.2±19.3	9.38±0.83	0.033765±0.002230
	Refitted FDT	49.2±19.3	5.13±2.34	0.014013±0.008211
	Backfitted FDT	49.2±19.3	3.11±1.29	0.007997±0.004856

Table D.24: Omib dataset, piecewise linear discriminator, variant for Fibonacci search (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Full FDT	101.4±4.6	12.58±0.51	0.044857±0.003987
	Pruned FDT	58.4±10.9	12.82±0.46	0.045859±0.003510
	Refitted FDT	58.4±10.9	7.72±1.97	0.021103±0.006653
	Backfitted FDT	58.4±10.9	4.98±2.97	0.012624±0.008590
500	Full FDT	124.0±7.2	9.37±2.71	0.034995±0.004978
	Pruned FDT	70.4±11.9	9.85±2.29	0.033680±0.005862
	Refitted FDT	70.4±11.9	4.21±1.10	0.009916±0.002923
	Backfitted FDT	70.4±11.9	2.71±0.40	0.005870±0.001336
1000	Full FDT	128.4±2.7	7.56±0.92	0.033728±0.001116
	Pruned FDT	55.6±13.1	8.83±0.96	0.032380±0.002489
	Refitted FDT	55.6±13.1	4.51±0.99	0.011260±0.003040
	Backfitted FDT	55.6±13.1	2.90±0.48	0.006622±0.001372
2000	Full FDT	137.5±5.2	7.39±0.88	0.034388±0.000975
	Pruned FDT	63.4±14.2	8.83±0.89	0.032518±0.001372
	Refitted FDT	63.4±14.2	3.92±1.80	0.010054±0.005171
	Backfitted FDT	63.4±14.2	2.55±0.62	0.005433±0.002094

Table D.25: Omib dataset, piecewise linear discriminator, refitting during pruning (q = 5)

GS size	Method	Complexity	Error rate (%)	MSE
100	Full FDT	99.4±5.4	12.39±0.91	0.044148±0.002615
	Pruned FDT	21.6±7.1	11.58±1.64	0.038413±0.006397
	Refitted FDT	21.6±7.1	9.84±1.55	0.029160±0.005869
	Backfitted FDT	21.6±7.1	5.33±2.83	0.016182±0.008342
500	Full FDT	127.8±3.5	8.85±2.46	0.034983±0.005563
	Pruned FDT	89.2±14.4	3.84±0.71	0.009554±0.001678
	Refitted FDT	89.2±14.4	3.20±0.55	0.007383±0.001562
	Backfitted FDT	89.2±14.4	2.01±0.17	0.004165±0.001166
1000	Full FDT	130.2±4.4	8.01±1.30	0.034715±0.002527
	Pruned FDT	106.0±11.8	3.45±0.58	0.009195±0.001447
	Refitted FDT	106.0±11.8	3.30±0.48	0.007948±0.001176
	Backfitted FDT	106.0±11.8	2.25±0.38	0.005556±0.000576
2000	Full FDT	132.0±6.7	7.77±0.60	0.035325±0.000857
	Pruned FDT	112.0±10.6	2.94±1.05	0.002591±0.002591
	Refitted FDT	112.0±10.6	2.81±0.90	0.006202±0.002552
	Backfitted FDT	112.0±10.6	1.90±0.46	0.003713±0.001146

Table D.26: Waveform dataset, piecewise linear discriminator (q = 5)

GS size	Method	Complexity	Error rate (%)
100	Pruned ULG	5.2±1.3	31.68±2.56
	Pruned CART	4.9±1.3	31.60±2.02
	Pruned C4.5	9.6±0.5	32.16±1.67
	Full FDT	61.3±5.2	28.06±1.08
	Pruned FDT	12.5±5.8	24.98±1.08
	Refitted FDT	12.5±5.8	21.94±1.10
	Backfitted FDT	12.5±5.8	21.24±0.78
300	Pruned ULG	13.8±3.2	29.50±1.28
	Pruned CART	12.1±1.8	27.42±1.42
	Pruned C4.5	25.4±2.2	30.00±2.18
	Full FDT	132.6±16.2	23.60±0.65
	Pruned FDT	34.1±10.2	22.36±1.61
	Refitted FDT	34.1±10.2	19.70±1.92
	Backfitted FDT	34.1±10.2	19.02±1.22
1000	Pruned ULG	16.0±4.5	27.62±1.51
	Pruned CART	34.1±8.0	25.00±0.55
	Pruned C4.5	75.2±3.9	26.38±1.59
	Full FDT	185.2±12.1	18.76±1.05
	Pruned FDT	43.0±13.5	18.46±0.82
	Refitted FDT	43.0±13.5	17.68±0.61
	Backfitted FDT	43.0±13.5	18.18±1.17
2000	Pruned ULG	40.4±24.4	26.12±1.22
	Pruned CART	55.2±9.4	24.06±1.59
	Pruned C4.5	136.0±8.7	25.46±0.77
	Full FDT	187.3±8.5	17.12±0.93
	Pruned FDT	44.6±7.2	17.96±0.19
	Refitted FDT	44.6±7.2	17.44±0.42
	Backfitted FDT	44.6±7.2	17.40±0.53

Table D.27: Satellite dataset, piecewise linear discriminator (q = 5)

GS size	Method	Complexity	Error rate (%)
100	Pruned ULG	5.8±0.7	26.50±1.39
	Pruned CART	3.5±0.4	31.33±1.69
	Pruned C4.5	10.0±1.1	25.88±1.57
	Full FDT	41.5±6.3	25.19±3.54
	Pruned FDT	7.1±1.4	22.95±1.99
	Refitted FDT	7.1±1.4	20.58±1.41
	Backfitted FDT	7.1±1.4	19.12±1.01
500	Pruned ULG	13.8±6.3	20.75±1.26
	Pruned CART	10.1±1.5	21.07±1.15
	Pruned C4.5	36.8±6.1	20.74±0.34
	Full FDT	75.3±3.3	16.20±1.52
	Pruned FDT	11.2±2.0	17.21±0.65
	Refitted FDT	11.2±2.0	16.24±0.48
	Backfitted FDT	11.2±2.0	15.89±0.80
1000	Pruned ULG	15.6±6.3	18.96±0.63
	Pruned CART	15.2±0.9	18.13±0.81
	Pruned C4.5	60.4±5.3	17.70±0.60
	Full FDT	94.1±4.1	14.81±0.70
	Pruned FDT	14.2±2.5	15.69±0.72
	Refitted FDT	14.2±2.5	14.91±1.06
	Backfitted FDT	14.2±2.5	14.97±1.15
2000	Pruned ULG	36.4±9.4	16.80±0.49
	Pruned CART	21.4±1.9	16.65±0.55
	Pruned C4.5	112.2±8.4	16.12±0.56
	Full FDT	114.5±4.4	14.15±0.63
	Pruned FDT	13.7±2.2	16.14±1.15
	Refitted FDT	13.7±2.2	14.93±0.72
	Backfitted FDT	13.7±2.2	14.43±0.56

Table D.28: Pendigits dataset, piecewise linear discriminator (q = 5)

GS size	Method	Complexity	Error rate (%)
100	Pruned ULG	14.2±2.9	34.01±2.98
	Pruned CART	3.4±0.3	40.98±5.53
	Pruned C4.5	12.8±0.8	32.60±2.80
	Full FDT	12.0±2.5	34.47±5.53
	Pruned FDT	5.7±1.0	31.57±4.78
	Refitted FDT	5.7±1.0	27.19±3.37
	Backfitted FDT	5.7±1.0	20.69±2.40
500	Pruned ULG	26.2±5.1	19.53±1.72
	Pruned CART	9.0±0.9	23.75±1.92
	Pruned C4.5	32.2±1.8	19.22±1.29
	Full FDT	29.5±0.8	16.91±1.00
	Pruned FDT	12.4±0.5	17.17±0.91
	Refitted FDT	12.4±0.5	15.31±1.39
	Backfitted FDT	12.4±0.5	13.31±0.95
1000	Pruned ULG	51.2±9.0	15.33±1.90
	Pruned CART	12.7±1.2	18.20±1.44
	Pruned C4.5	48.6±6.1	14.96±1.25
	Full FDT	40.6±2.2	12.01±1.31
	Pruned FDT	15.5±1.8	12.97±1.79
	Refitted FDT	15.5±1.8	12.32±1.71
	Backfitted FDT	15.5±1.8	10.45±1.68
2000	Pruned ULG	69.2±9.4	12.77±1.22
	Pruned CART	18.9±1.7	13.95±0.61
	Pruned C4.5	72.4±4.9	11.46±0.82
	Full FDT	50.8±0.9	10.31±1.53
	Pruned FDT	17.1±1.1	11.41±0.85
	Refitted FDT	17.1±1.1	10.37±0.96
	Backfitted FDT	17.1±1.1	9.62±1.20

Table D.29: Dig44 dataset, piecewise linear discriminator (q = 5)

GS size	Method	Complexity	Error rate (%)
100	Pruned ULG	13.8±2.0	44.38±3.92
	Pruned CART	4.1±0.4	52.67±3.61
	Pruned C4.5	16.4±0.5	43.00±1.67
	Full FDT	21.1±1.6	46.37±2.23
	Pruned FDT	7.2±0.9	44.23±2.90
	Refitted FDT	7.2±0.9	36.61±2.22
	Backfitted FDT	7.2±0.9	30.37±1.96
500	Pruned ULG	28.2±4.0	29.73±1.94
	Pruned CART	11.0±0.5	34.33±1.50
	Pruned C4.5	55.6±2.6	28.98±0.58
	Full FDT	64.8±5.1	25.62±2.39
	Pruned FDT	17.0±1.8	23.98±2.05
	Refitted FDT	17.0±1.8	20.98±1.02
	Backfitted FDT	17.0±1.8	18.77±1.02
1000	Pruned ULG	54.2±9.2	25.35±0.55
	Pruned CART	15.8±1.7	28.02±1.89
	Pruned C4.5	93.6±4.0	24.22±0.61
	Full FDT	93.3±2.4	20.02±0.43
	Pruned FDT	20.3±1.8	19.29±0.49
	Refitted FDT	20.3±1.8	17.57±0.84
	Backfitted FDT	20.3±1.8	16.41±0.77
2000	Pruned ULG	74.8±14.2	22.32±0.53
	Pruned CART	25.6±1.5	23.74±0.66
	Pruned C4.5	156.0±7.1	19.92±0.86
	Full FDT	121.9±5.3	16.07±0.84
	Pruned FDT	23.0±1.6	15.81±0.80
	Refitted FDT	23.0±1.6	14.70±0.36
	Backfitted FDT	23.0±1.6	14.03±0.27

Bibliography

[ACM+90] L. Atlas, R. Cole, Y. Muthusamy, A. Lippman, J. Connor, D. Park, M. El-Sharkawi, and R.J. Marks II. A performance comparison of trained multilayer perceptrons and trained classification trees. *Proceedings of the IEEE*, 78(10):1614–1619, October 1990.

[AG92] R. Araya and P. Gigon. Segmentation trees: A new help building expert systems and neural networks. In *Proceedings of Stats*, pages 119–124, 1992.

[Alp99] E. Alpaydin. Combined 5x2 cv F Test for comparing supervised classification algorithms. *Neural computation*, 11:1885–1892, 1999. Note communicated by Thomas Dietterich.

[AZZ98] B. Apolloni, G. Zamponi, and A.M. Zanaboni. Learning fuzzy decision trees. *Neural Networks*, 11:885–895, 1998.

[Ber00] M. Bernadet. Basis of a fuzzy knowledge discovery system. In Zighed D.A, Komorowski J., and Zytkov J., editors, *Principles of Data Mining and Knowledge Discovery, Proceedings of the 4th European Conf., PKDD 2000*, pages 24–33, Lyon, France, September 2000.

[Bez93] J.C. Bezdek. Editorial: Fuzzy Models - What Are They, and Why? *IEEE Transactions on Fuzzy Systems*, 1(1):1–6, February 1993.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C.J. Stone. *Classification and regression trees*. Chapman and Hall, New-York, 1984.

[Bis97] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1997.

[BK99] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning*, 36:105–139, 1999.

[BLM98] J.F. Baldwin, J. Lawry, and T.P. Martin. Mass assignment based induction of decision trees on words. In *Proceedings of Information Processing and Management of Uncertainty in Knowledge-Based Systems*, volume 1, pages 524–531, Paris, July 6-10 1998.

[BM93] B. Bouchon-Meunier. *La logique floue*. Collection “Que sais-je?”. Presses Universitaires de France, Paris, 1993.

[BM98] N.M. Bigolin and C. Marsala. Fuzzy spatial OQL for fuzzy knowledge discovery in databases. In *Principles of Data Mining and Knowledge Discovery, Proceedings of the 2nd European Symposium PKDD’98*, volume 1510 of *LNAI*, pages 246–254, Nantes, France, 1998. Springer-Verlag.

[BMMR96] B. Bouchon-Meunier, C. Marsala, and M. Ramdani. Learning from imperfect data. In *Fuzzy Sets Methods in Information Engineering: a Guided tour of Applications*, pages 139–148. D. Dubois, H. Prade and R.R. Yager, 1996.

[BN92] W. Buntine and T. Niblett. Technical note. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.

[Boy95] X. Boyen. Design of fuzzy logic-based decision trees applied to power system transient stability assessment. Master's thesis, University of Liège, 1995.

[Boz02a] O. Boz. Converting a trained neural network to a decision tree Dec-Text - decision tree extractor. In *ICMLA2002, International Conference on Machine Learning Applications*, 24-27 June 2002.

[Boz02b] O. Boz. Extracting decision trees from trained neural networks. In *KDD2002, 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 23-26 2002.

[Bre96] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Bre98] L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

[Bun92] W. Buntine. Learning classification trees. *Statistics and Computing*, 2:63–73, 1992.

[BW95a] X. Boyen and L. Wehenkel. Fuzzy decision tree induction for power system security assessment. In *Proceedings of SIPOWER'95, IFAC Symposium on Control of Power Plants and Power Systems*, pages 151–156, Cancun, Mexico, December 1995.

[BW95b] X. Boyen and L. Wehenkel. On the unfairness of convex discriminator quality measures for fuzzy partitioning in machine learning. Technical report, University of Liège, 1995.

[BW96] X. Boyen and L. Wehenkel. Automatic induction of continuous decision trees. In *Proceedings of IPMU'96, Information Processing and Management of Uncertainty in Knowledge-Based Systems*, volume 1, pages 419–424, Granada, Spain, July 1-5 1996.

[BW99] X. Boyen and L. Wehenkel. Automatic induction of fuzzy decision trees and its applications to power system security assessment. *Fuzzy Sets and Systems*, 1(102):3–19, February 1999.

[CC87] C. Carter and J. Catlett. Assessing credit card applications using machine learning. *IEEE Expert*, pages 71–79, Fall 1987.

[CH02] I.-J. Chiang and J. Y.-J. Hsu. Fuzzy classification trees for data analysis. *Fuzzy Sets and Systems*, 130:87–99, 2002.

[Che88] S-M. Chen. A new approach to handling fuzzy decision making problems. *IEEE Transactions on Systems, Man and Cybernetics*, 18(6):1012–1016, November/December 1988.

[CL92] K.J. Cios and N. Liu. A machine learning method for generation of a neural network architecture: A Continuous ID3 algorithm. *IEEE Transactions on Neural Networks*, 3(2):280–291, March 1992.

[Coh95] P. R. Cohen. *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, Massachusetts, 1995.

[CP77] R.L.P. Chang and Th. Pavlidis. Fuzzy decision tree algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-7(1):28–35, January 1977.

[CS92] K.J. Cios and L.M. Sztandera. Continuous ID3 algorithm with fuzzy entropy measures. In *Proceedings of The First IEEE Conference on Fuzzy Systems*, pages 469–476, San Diego, 1992.

[CS96] M.W. Craven and J.W. Shavlik. Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing Systems*, 8:24–30, 1996. MIT Press, Cambrige, MA.

[CS97] K.J. Cios and L.M. Sztandera. Ontogenic neuro-fuzzy algorithm: F-CID3. *Neurocomputing*, 14(4):383–402, 1997.

[CY96] Z. Chi and H. Yan. ID3-derived fuzzy rules and optimized defuzzification for handwritten numeral recognition. *IEEE Transactions on Fuzzy Systems*, 4(1):24–31, February 1996.

[DAG01] W. Duch, R. Adamczak, and K. Grabczewski. A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on neural networks*, 12(2):277–306, March 2001.

[DHB90] T. Dietterich, H. Hild, and G. Bakiri. A comparative study of ID3 and backpropagation for english text-to-speech mapping. In *Proceedings of the Conference on Machine Learning*, pages 24–31, Austin, Texas, 1990.

[Die96] T.G. Dietterich. Editorial. *Machine Learning*, 24(2):1–3, 1996.

[Die98] T.G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.

[Die00a] T.G. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *First International Workshop on Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15, Cagliari, Italy, 2000. Springer Verlag.

[Die00b] T.G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

[DK95] T.G. Dietterich and 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.

[DK01] M. Dong and R. Kothari. Look-ahead based fuzzy decision tree induction. *IEEE Transactions on Fuzzy Systems*, 9(3):461–468, June 2001.

[DLT72] A. De Luca and S. Termini. A definition of a nonprobabilistic entropy in the setting of fuzzy sets theory. *Information and Control*, 20:301–312, 1972.

[DP80] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, 1980.

[Elk00] C. Elkan. [http://www.cs.ucsd.edu/~elkan/250b/may8.html], CSE 250B Lecture Notes, 2000.

[EMS97] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, May 1997.

[EVDB98] D. Ettes and J. Van Den Berg. Representation and learning capabilities of additive fuzzy systems. In P. Kopacek and L.J. Rudas, editors, *Proceedings of the 1998 IEEE International Conference on Intelligent Engineering Systems, INES'98*, pages 121–126, Vienna, Austria, September 1998.

[FFAK99] C.S. Fertig, A.A Freitas, L.V.R. Arruda, and C. Kaestner. A fuzzy beam-search rule induction algorithm. In *Principles of Data Mining and Knowledge Discovery: Proceedings of the 3rd European Conference PKDD'99*, volume 1704 of *LNAI*, pages 341–347. Springer-Verlag, 1999.

[FJK99] M. Faifer, C. Janikow, and K. Krawiec. Extracting fuzzy symbolic representation from artificial neural networks. In *Proceedings of the 18th International Conference of the North American Fuzzy Information Processing Society, NAFIPS*, New York City, June 10-12 1999.

[FM88] D.H. Fisher and B. McKusick. An empirical comparison of ID3 and back-propagation. *Artificial Intelligence*, 3:251–288, 1988.

[FPSSU96] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in knowledge discovery and data mining*. AAAI Press / MIT Press, Menlo Park / Cambridge, 1996.

[Fri77] J.H. Friedman. A recursive partitioning decision rule for nonparametric classifier. *IEEE Transactions on Computers*, C-26:404–408, 1977.

[Fri96] J.H. Friedman. Local learning based on recursive covering. Technical report, Department of Statistics, Standford University, August 1996.

[Fri97] J.H. Friedman. On bias, variance, 0/1-loss and curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1:55–77, 1997.

[FS96] Y. Freund and R. E. Schapire. Experiment with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–156, 1996.

[FS98] Y. Freund and R. Schapire. Discussion of the paper "Arcing Classiers" by Leo Breiman. *Annals of Statistics*, 26(3):824–832, 1998.

[FS99] Y. Freund and R. E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, September 1999.

[FWI+98] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I.H. Witten. Using model trees for classification. *Machine learning*, 32:63–76, 1998.

[Gam99] J. Gama. Discriminant trees. *Proceedings of the 16th International Conference on Machine Learning*, pages 134–142, 1999.

[GBD92] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[Geu01] P. Geurts. Dual perturb and combine algorithm. In *Proceedings of AI & Statistics 2001*, pages 196–201, Key-West, Florida, January 2001.

[Geu02] P. Geurts. *Contributions to Decision Tree Induction: Bias/Variance Tradeoff and Time Series Classification*. PhD thesis, University of Liège, Belgium, Mai 2002.

[Geu03] P. Geurts. Traitement de données volumineuses pas ensemble d'arbres aléatoires. In O. Boussaid and S. Lallich, editors, *Proceedings of "Journées de statistique": Revue des Nouvelles Technologies de l'information*, number 1, pages 111–122, Lyon, June 2003.

[GGR99] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining very large databases. *IEEE Computer*, August 1999.

[GHS02] M. Guetova, S. Hölldobler, and H.-P. Störr. Incremental fuzzy decision trees. In M. Jarke, J. Koehler, and G. Lakemeyer, editors, *Proceedings of the 25th German Conference on Artificial Intelligence (KI2002)*, pages 67–81, Aachen, Germany, 2002. Springer Verlag.

[GOW01] P. Geurts, C. Olaru, and L. Wehenkel. Improving the bias/variance tradeoff of decision trees: towards soft tree induction. *Engineering Intelligent Systems*, 9(4):195–204, December 2001.

[Gre03] H.J. Greenberg. Mathematical programming glossary. http://www.cudenver.edu/~hgreenbe/glossary/, 2003.

[GW00] P. Geurts and L. Wehenkel. Investigation and reduction of discretization variance in decision tree induction. In *ECML 2000, Proceedings of 11th European Conference on Machine Learning*, pages 162–170, Barcelona, Spain, May/June 2000.

[HAC+99] J. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. Haas. Interactive data analysis: The control project. *IEEE Computer*, August 1999.

[HB99] S. Hettich and S. D. Bay. The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science, 1999.

[Hei94] A. P. Heinz. Learning and generalization in adaptive fuzzy logic networks. In *EUFIT'94, Proceedings of the Second European Congress on Intelligent Techniques and Soft Computing*, pages 1347–1351, Aachen, Germany, 20-23 September 1994.

[Hei95a] A. P. Heinz. Adaptive fuzzy neural trees. In G.E. Lasker and X. Liu, editors, *Advances in Intelligent Data Analysis, Proceedings of the IDA-95 Symposium*, pages 70–74, Baden-Baden, Germany, 17-19 August 1995.

[Hei95b] A. P. Heinz. Pipelined neural tree learning by error forward-propagation. In *ICNN'95, Proceedings of IEEE International Conference on Neural Networks*, volume I, pages 394–397, Perth, Western Australia, 27 November - 1 December 1995.

[Hei96] A. P. Heinz. Tree-structured neural network fo real-time adaptive control. In *ICONIP'96, Proceedings of the International Conference in Neural Information Processing*, volume 2, pages 956–931, Hong-Kong, 24-27 September 1996.

[Hei98] A. P. Heinz. Efficient top-down jacobian evaluation of tree-structured neural networks. In *ICANN'98, Proceedings of the 8th International Conference on Artificial Neural Networks*, volume 1, pages 87–92, Skovde, Sweden, 2-4 September 1998.

[HL97] L. O. Hall and P. Lande. Generating fuzzy rules from decision trees. In *Proceedings of IFSA International Fuzzy Systems Association World Congress*, volume 2, pages 418–423, Prague, 1997.

[HL02] H.-P. Huang and C.-C. Liang. Strategy-based decision making of a soccer robot system using a real-time self-organizing fuzzy decision tree. *Fuzzy Sets and Systems*, 127:49–64, 2002.

[HMS01] D. Hand, H. Mannila, and P. Smyth. *Principles of data mining*. MIT Press, Cambridge, England, 2001.

[HOJ98] I. Hayashi, J. Ozawa, and L.C. Jain. Generation of fuzzy decision trees by Fuzzy ID3 with adusting mechanism of and/or operators. *IEEE*, 1998.

[HTF01] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of Statistical Learning*. Springer Series in Statistics. Springer, 2001.

[IS96a] A. Ittner and M. Schlosser. Discovery of relevant features by generating non-linear decision trees. In E. Simoudis, J. Han, and U. Fayyad, editors, *Proceedings of the 12th International Conference on Knowledge discovery and data mining, KDD96*, pages 108–113, Menlo Park, CA, USA, 1996. AAAI Press.

[IS96b] A. Ittner and M. Schlosser. Non-linear decision trees - NDT. In *Proceedings of the 13th International Conference on Machine Learning, ICML96*, pages 252–257, San-Francisco, CA, USA, 1996. Morgan Kaufmann Publishers.

[ISNM96] H. Ichihashi, T. Shirai, K. Nagasaka, and 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*, 81:157–167, 1996.

[IZR+97] A. Ittner, J. Zeidler, R. Rossius, W. Dilger, and M. Schlosser. Feature space partitioning by non-linear and fuzzy decision trees. In *Proceedings of International Fuzzy Systems Association World Congress*, volume 2, pages 394–398, Prague, 1997.

[Jan94] C.Z. Janikow. Fuzzy decision tree with missing values. FIDMV_1.3. Technical report, Department of Mathematics and Computer Science, University of Missouri, 1994.

[Jan96a] C.Z. Janikow. Exemplar learning in fuzzy decision trees. In *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems*, volume 2, pages 1500–1505, New Orleans, USA, 1996.

[Jan96b] C.Z. Janikow. A genetic algorithm method for optimizing fuzzy decision trees. *Information Sciences*, 89:275–296, 1996.

[Jan98] C.Z. Janikow. Fuzzy decision trees: Issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 28(1):1–14, February 1998.

[JF99] C.Z. Janikow and M. Faifer. Fuzzy partitioning with FID3.1. In *Proceedings of the 18th International Conference of the North American Fuzzy Information Processing Society*, pages 467–471, 1999.

[JJL97] B. Jeng, Y-M. Jeng, and T-P. Liang. FILM: a fuzzy inductive learning method for automated knowledge acquisition. *Decision Support Systems*, 21:61–73, 1997.

[Jor94a] M. I. Jordan. A statistical approach to decision tree modeling. In M. Warmuth, editor, *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, New York, 1994. ACM Press.

[Jor94b] M.I. Jordan. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[Kau73] A. Kaufmann. *Introduction à la théorie des sous-ensembles flous. 1. Eléments théoriques de base*. Masson et Cie, Paris, 1973.

[KD95] E.B. Kong and T.G. Dietterich. Error-correcting output coding corrects bias and variance. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 313–321, Tahoe City, CA. San Francisco, 1995. Morgan Kaufmann.

[KI90] J. Kacprzyk and C. Iwanski. Inductive learning from incomplete and imprecise examples. In B. Bouchon-Meunier, R.R. Yager, and L.A. Zadeh, editors, *Uncertainty in Knowledge Bases, Proceedings of IPMU'90*, pages 424–430, Paris, France, July 2-6 1990.

[KI92] J. Kacprzyk and C. Iwanski. Learning from erroneous examples using fuzzy logic and "textbook knowledge". In *Proceedings of Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 559–561, Mallorca, July 6-10 1992.

[Kos92] B. Kosko. *Neural networks and fuzzy systems: A Dynamical Systems Approach to Machine Intelligence*. Prentice Hall, 1992.

[Kub98] M. Kubat. Decision tress can initialize radial-basis function networks. *IEEE Transactions on neural Networks*, 9(5):813–821, September 1998.

[LBMD+00] A. Laurent, B. Bouchon-Meunier, A. Doucet, S. Gancarski, and C. Marsala. Fuzzy data mining from multidimensional databases. In *ISCI2000, International Symposium on Computational Intelligence*, volume 549 of *Studies CI Series*, Kosice, Slovakia, 2000. Springer-Verlag.

[LC84] P. Langley and J.G. Carbonell. Approaches to machine learning. *Journal of the american society for information science*, 35(5):306–316, 1984.

[Lee90] C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller - Part I and II. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):404–435, March/April 1990.

[LGM00] A. Laurent, S. Gancarski, and C. Marsala. Coopération entre un système d'extraction de connaissances floues et un système de gestion de base de données multidimensionnelles. In *LFA2000, Rencontre Francophones sur la Logique Floue et ses Applications*, pages 325–332, La Rochelle, 18-20 Octombre 2000. Cepadries Editions.

[LKC+95] H.D. Li, M. Kallergi, L.P. Clarke, V.K. Jain, and R.A. Clark. Markov random field for tumor detection in digital mammography. *IEEE Transactions on Medical Imaging*, 14(3), September 1995.

[LLS00] T-S. Lim, W-Y. Loh, and Y-S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40(3):203–228, September 2000.

[LP89] K.C. Lee and S.J. Park. A knowledge-based fuzzy decision tree classifier for time series modeling. *Fuzzy Sets and Systems*, 33:1–18, 1989.

[Mar94] C. Marsala. Arbres de decision et sous-ensambles flous. Rapport de stage, Laforia 94/21, 20 pages, Institut Blaise Pascal, Octobre 1994.

[Mar95] C. Marsala. Fuzzy partition inference over a set of numerical values. Technical report, LAFORIA-IBP, University Pierre et Marie Curie, France, October 1995.

[Mar96] C. Marsala. Fuzzy partitioning using mathematical morphology in a learning scheme. In *Proceedings of the 5th International Conference on Fuzzy Systems, FUZZ-IEEE'96*, pages 1512–1517, New Orleans, September 1996.

[Mar98a] C. Marsala. Application of fuzzy rule induction to data mining. In *Proceedings of the 3rd International Conference FQAS'98*, volume 1495 of *LNAI*, pages 260–271, Roskilde, Denmark, May 1998. Springer.

[Mar98b] C. Marsala. *Apprentissage inductif en présence de données imprécises: construction et utilisation d'arbres de décision flous*. PhD thesis, Université Paris 6, 1998.

[Mar00] C. Marsala. Fuzzy decision trees to help flexible querying. *Kybernetika*, 36(6):689–705, 2000.

[MBK98] R.S. Michalski, I. Bratko, and M. Kubat, editors. *Machine learning and data mining: methods and applications*. Wiley, Chichester, 1998.

[MBM97] C. Marsala and B. Bouchon-Meunier. Forests of fuzzy decision trees. In *Proceedings of International Fuzzy Systems Association World Congress*, volume 2, pages 369–374, Prague, 1997.

[MBM99] C. Marsala and B. Bouchon-Meunier. An adaptable system to construct fuzzy decision trees. In *Proceedings of the International Conference of NAFIPS'99 (North American Fuzzy Information Processing Society)*, pages 223–227, New York, USA, June 1999.

[McG01] M. McGranaghan. Trends in power quality monitoring. *IEEE Power Engineering Review*, pages 3–8, October 2001.

[MCM83] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach*. Springer-Verlag, Berlin, 1983.

[MCSL01] V. Medina-Chico, A. Suarez, and J.F. Lutsko. Backpropagation in decision trees for regression. In De Raedt and P. Flach, editors, *Proceedings of ECML 2001*, LNAI 2167, pages 348–359, Freiburg, Germany, September 3-7 2001. Springer-Verlag Berlin Heidelberg.

[MDIS97] J.T. McClave, F.H. Dietrich-II, and T. Sincich. *Statistics. Seventh edition*. Prentice Hall, 1997.

[MGW03] R. Marée, P. Geurts, and L. Wehenkel. Une méthode générique pour la classification d'images á partir des pixels. In O. Boussaid and S. Lallich, editors, *Proceedings of "Journées de statistique": Revue des Nouvelles Technologies de l'information*, number 1, pages 227–238, Lyon, June 2003.

[MI99] M.M Morcos and W.R.A. Ibrahim. Electric power quality and artificial intelligence: Overview and applicability. *IEEE Power Engineering Review*, pages 5–10, June 1999.

[Min89a] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.

[Min89b] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3:319–342, 1989.

[Mit97] T.M. Mitchell. *Machine Learning*. McGraw-Hill Companies, 1997.

[MRTZ98] C. Marsala, M. Ramdani, M. Toullabi, and D. Zakaria. Recognition of odors: a fuzzy decision tree approach. In *Proceedings of Information Processing and Management of Uncertainty in Knowledge-Based Systems*, volume 1, pages 532–539, Paris, July 6-10 1998.

[MST94] D. Michie, D.J. Spiegelhalter, and C.C Taylor, editors. *Machine learning, neural and statistical classification*. Ellis Horwood, 1994.

[NKK97] D. Nauck, F. Klawonn, and R. Kruse. *Foundations of Neuro-Fuzzy Systems*. John Wiley, 1997.

[NM00] D. Niebur and O. Malik. Tutorial on fuzzy logic applications in power systems. In K. Tomsovic and M.Y. Chow, editors, *IEEE-PES Winter meeting*, Singapore, January 2000.

[OGW99] C. Olaru, P. Geurts, and L. Wehenkel. Data mining tools and applications in power system engineering. In *Proceedings of the conference PSCC*, volume 1, pages 324–330, Trondheim, Norway, June 28 - July 2nd 1999.

[Ola98] C. Olaru. Fuzzy decision tree induction using square error type of criterion. Internal Report, University of Liege, Dept. of Electrical Circuits, Belgium, October 1998.

[OW99] C. Olaru and L. Wehenkel. Data mining. *IEEE Computer Applications in Power*, 12(3):19–25, July 1999.

[OW00] C. Olaru and L. Wehenkel. On neurofuzzy and fuzzy decision trees approaches. In B. Bouchon-Meunier, R.R. Yager, and L.A. Zadeh, editors, *Information, Uncertainty, Fusion*, pages 131–145. Kluwer Academic, 2000.

[OW03] C. Olaru and L. Wehenkel. A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, (138):221–254, 2003.

[PK99] B.D. Pitt and D.S. Kirschen. Application of data mining to load profiling. In *PICA'99, Proceedings of the 21st International Conference on Power Industry Computer Applications*, pages 131–136, Santa Clara, California, 16-21 May 1999.

[PS01] W. Pedrycz and Z. Sosnowski. The design of decision trees in the framework of granular data and their application to software quality models. *Fuzzy Sets and Systems*, 123:271–290, 2001.

[PTVF94] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C. The Art of Scientific Computing. Second Edition*. Cambridge University Press, 1994.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui90] J. R. Quinlan. Decision trees and decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):339–346, March/April 1990.

[Qui93] J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers, San-Mateo, 1993.

[Qui96] J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[Ram94] M. Ramdani. *Système d'induction formelle à base de connaissances imprécises*. PhD thesis, Université Paris VI, Paris, France, February 1994.

[RJ91] J.-S. Roger Jang. Rule extraction using generalized neural networks. In *Proceedings of IFSA World Congress*, volume AI, pages 82–85, Brussels, Belgium, 1991.

[RJ93] J.-S. Roger Jang. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Transactions on System, Man, and Cybernetics*, 23(3):665–685, 1993.

[RJ94] J.-S. Roger Jang. Structure determination in fuzzy modeling: a fuzzy CART approach. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, volume 1, pages 480–485, Orlando, Florida, June 26-29 1994.

[Sap90] G. Saporta. *Probabilités, analyse des données et statistique*. Editions Technip - Paris, 1990.

[SATN95] T. Shibata, T. Abe, K. Tanie, and M. Nose. Motion planning of a redundant manipulator based on criteria of skilled operators using Fuzzy-ID3 and GMDH. In *Proceedings of 6th IFSA World Congress*, volume 1, pages 613–616, Sao Paulo, Brasil, July 21-28 1995.

[SATN96] T. Shibata, T. Abe, K. Tanie, and M. Nose. Skill based motion planning in hierarchical intelligent control of a redundant manipulator. *Robotics and Autonomous Systems*, 18:65–73, 1996.

[Sch93] C. Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.

[Sch00] M. Schmitt. On the complexity of computing and learning with multiplicative neural networks. In *NeuroCOLT Workshop "New perspectives in the theory of neural nets"*, Graz, Austria, May 3-5 2000.

[Set90] I. K. Sethi. Entropy Nets: From decision tress to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, 1990.

[Set95] I. K. Sethi. Neural implementation of tree classifiers. *IEEE Transactions on Systems, Man and Cybernetics*, 25(8):1243–1249, August 1995.

[SHM99] G.H. Shah Hamzei and D.J. Mulvaney. On-line learning of fuzzy decision trees for global path planning. *Engineering Applications of Artificial Intellingence*, 12:93–109, 1999.

[SI02] J. F. Smith III. Data mining for fuzzy decision tree structure with a genetic program. *Lecture Notes in Computer Science, LNCS*, 2412:13–18, 2002.

[SL91] S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):660–674, May-June 1991.

[SL99] A. Suarez and F. Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions Pattern Analysis Machine Intelligence*, 21(12):1297–1311, December 1999.

[SMT91] J. Shavlik, R.J. Mooney, and G.G. Towell. Symbolic and neural learning algorithms: an experimental comparison. *Machine Learning*, 6:111–143, 1991.

[SS93] M. Sebag and Schoenauer. *Inductive Learning of Membership Functions and Fuzzy Rules*. Machine Intelligence and Pattern Recognition series. Elsevier Science Publishers B.V., 1993.

[SSR01] S. Sumathi, S.N. Sivanandam, and R. Ravindran. Design of a soft computing hybrid model classifier for data mining applications. *Engineering Intelligent systems*, 9(1):33–56, March 2001.

[SYSW01] S.C.K. Shiu, D.S. Yeung, C.H. Sun, and X.Z. Wang. Transferring case knowledge to adaptation knowledge: an approach for case-base maintenance. *Computational Intellingence*, 17(2), 2001.

[TMMN96] T. Tsuchiya, T. Maeda, Y. Matsubara, and M. Nagamachi. A fuzzy rule induction method using genetic algorithm. *International Journal of Industrial Ergonomics*, 18:135–145, 1996.

[Tor98a] L. Torgo. A comparative study of reliable error estimators for pruning regression trees. In H. Coelho, editor, *Proceedings of the Iberoamerican Conference on Artificial Intelligence*, 1998.

[Tor98b] L. Torgo. Error estimators for pruning regression trees. In H. Coelho, editor, *Proceedings of the Iberoamerican Conference on Artificial Intelligence*, 1998.

[Tor99] L. Torgo. *Inductive Learning of Tree-based Regression Models*. PhD thesis, Department of Computer Science, Faculty of Sciences, University of Porto, September 1999.

[TST92] T. Tani, M. Sakoda, and K. Tanaka. Fuzzy modeling by ID3 algorithm and its application to prediction of heater outlet temperature. In *Proceedings of The First IEEE Conference on Fuzzy Systems*, pages 923–930, San Diego, 1992.

[Tur95] P. Turney. Technical Note: Bias and quantification of stability. *Machine Learning*, 20:23–33, 1995.

[TWY00] E.C.C. Tsang, X.Z. Wang, and D.S. Yeung. Improving learning accuracy of fuzzy decision trees by hybrid neural networks. *IEEE Transactions on Fuzzy Systems*, 8(5):601–614, October 2000.

[UOHT94] M. Umano, H. Okamoto, I. Hatono, and H. Tamura. Fuzzy decision trees by Fuzzy ID3 algorithm and its application to diagnosis systems. In *Proceedings of The Third IEEE Conference on Fuzzy Systems*, volume 3, pages 2113–2118, Orlando, Florida, June 26-29 1994.

[Utg89a] P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 1989.

[Utg89b] P. E. Utgoff. Perceptron trees: A case study in hybrid concept representations. *Connection Science*, 1(4):377–391, 1989.

[VCWC94] C. Voudouris, P. Chernett, C.J. Wang, and L. Callaghan. Fuzzy hierarchical control for autonomous vehicles. Report CSM-216, Department of Computer Science, University of Essex, UK, December 1994.

[Ves58] A. Vessereau. *La statistique*. Collection "Que sais-je?". Presses Universitaires de France, Paris, 1958.

[WA93] L. Wehenkel and V.B. Akella. A hybrid decision tree - neural network approach for power system dynamic secuity assessment. In *Proceedings of the 4th Internal Symposium on Expert Systems Application to Power Systems*, pages 285–291, Melbourne, Australia, January 1993.

[WCQY00] X. Z. Wang, B. Chen, G. Qian, and F. Ye. On the optimization of fuzzy decision trees. *Fuzzy Sets and Systems*, 112(1):117–125, 2000.

[Web92] R. Weber. Fuzzy ID3: A class of methods for automatic knowledge acquisition. In *Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Networks*, pages 265–268, Iizuka, Japan, July 17-22 1992.

[Weh92a] L. Wehenkel. An information quality based decision tree pruning method. In *Proceedings of the IPMU'92 Conference*, Palma de Mallorca, Spain, July 6-10 1992.

[Weh92b] L. Wehenkel. A probabilistic framework for the induction of decision trees. Internal Report, University of Liege, 1992.

[Weh94] L. Wehenkel. Machine learning approaches to power system security assessment. Aggregation thesis, University of Liege, Belgium, May 1994.

[Weh97] L. Wehenkel. Discretization of continuous attributes for supervised lerning. Variance evaluation and variance reduction. Invited paper. In *Proceedings of 7th IFSA World Congress*, volume 2, pages 381–388, Prague, June 25-29 1997.

[Weh98] L. Wehenkel. *Automatic learning techniques in power systems*. Kluwer Academic, Boston, 1998.

[Weh00] L. Wehenkel. Automatic learning approaches for electric power systems. *Knowledge-Based Systems*, 3:977–1036, 2000.

[WJ97] L. Wehenkel and Y. Jacqumart. Tutorial course on automatic learning methods. Application to dynamic security assessment. In *PICA'97 - IEEE Power Industry*, February 1997.

[WK89] S. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proceedings of IJCAI*, Detroit, 1989. Morgan Kaufmann.

[WM95] X. Wu and P. Mahlen. Fuzzy interpretation of induction results. In Fayyad U.M. and Uthurusamy R., editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 325–330, Montreal, Canada, 20-21 August 1995. AAAI Press.

[WP93] L. Wehenkel and M. Pavella. Decision tree approach to power system security assessment. *Electrical Power and Energy Systems*, 15(1):13–36, February 1993.

[WS87] Q.R. Wang and C.Y. Suen. Large tree classifier with heuristic search and global training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(1):91–102, January 1987.

[Wu99] X. Wu. Fuzzy interpretation of discretized intervals. *IEEE Transactions on Fuzzy Systems*, 7(6):753–759, December 1999.

[XH98] W. Xizhao and J. Hong. On the handling of fuzziness for continuous-valued attributes in decision tree generation. *Fuzzy Sets and Systems*, 99:283–290, 1998.

[YS95] Y. Yuan and M.J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, 69:125–139, 1995.

[Zad65] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[Zad84] L. A. Zadeh. Making computers think like people. *IEEE Spectrum*, pages 26–32, August 1984.

[ZR00] D. A. Zighed and R. Rakotomalala. *Graphes d'induction*. Hermes Science Publications, Paris, 2000.

[ZS95] J. Zeidler and M. Schlosser. Fuzzy handling of continuous-valued attributes in decision trees. In Y. Kodratoff, G. Nakhaeizadeh, and C. Taylor, editors, *Proceedings of MLNet Familiarization Workshop: Statistics, Machine Learning and Knowledge Discovery in Databases*, pages 41–46, Heraklion, Crete, Greece, 1995.

[ZS96] J. Zeidler and M. Schlosser. Continuous-valued attributes in fuzzy decision trees. In *Proceedings of Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 395–400, Granada, 1996.

[Zur92] J. M. Zurada. *Introduction to Artificial Neural Systems*. West Publishing, 1992.