ON CASCADING SMALL DECISION TREES

Submitted to Autonomous University of Barcelona
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Computer Science

by Julià Minguillón i Alfonso
Bellaterra, July 2002

© Copyright 2002 by Julià Minguillón i Alfonso

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Bellaterra, July 2002

Jaume Pujol i Capdevila
(Principal Adviser)

to my family and friends,

Preface

This thesis is about using small decision trees for classification and data mining. The intuitive idea behind this thesis is that a sequence of small decision trees may perform better than a large decision tree, reducing both training and exploitation costs.

Our experiments using decision trees started during a research stage in collaboration with Prof. Kenneth Zeger, at the Electric and Computer Engineering Department, of the University of California, San Diego, and with Hewlett-Packard. Our goal was to develop a system capable to recognize several kinds of elements present in a document such as background, text, horizontal and vertical lines, line drawings and images. Then, each element would be treated accordingly to its characteristics. For example, background regions would be removed and not processed at all, while the other regions would be compressed using an appropriate algorithm, the lossy JPEG standard operation mode for images and a lossless method for the rest, for instance.

Our first experiments using decision trees showed that the decision trees we built were too large and they suffered from overfitting. Then, we tried to take advantage of spatial redundancy present in images, using a multi-resolution approach: if a large block cannot be correctly classified, split it in four subblocks and repeat the process recursively for each subblock, using all previous computed knowledge about such block. Blocks that could not be processed at a given block size were labeled as mixed, so the word progressive came up: a first low resolution version of the classified image is obtained with the first classifier, and it is refined by the second one, the third one, etc, until a final version is obtained with the last classifier in the ensemble.

Furthermore, the use of the progressive scheme yield to the use of smaller decision trees, as we no longer need a complex classifier. Instead of building a large and complex classifier for classifying the whole input training set, we only try to solve the easiest part of the classification problem, delaying the rest for a second classifier, and so.

The basic idea in this thesis is, therefore, a trade-off between cost and accuracy under a confidence constraint. A first classification is performed at a low cost; if an element is classified with a high confidence, it is accepted, and if not, it is rejected and a second classification is performed, and so. It is, basically, a variation of the cascading paradigm, where a first classifier is used to compute additional information from each input sample, information that will be used to improve classification accuracy by a second classifier, and so on.

What we present in this thesis, basically, is an extension of the cascading paradigm and an exhaustive empirical evaluation of the parameters involved in the creation of progressive decision trees. Some basic theoretical issues related to progressive decision trees such as system complexity, for example, are also addressed.

Acknowledgments

The author wish to thank his principal adviser, Prof. Jaume Pujol, for his unconditional support and numerous helpful suggestions.

This work would not have been possible without the help of Prof. Gabor Lugosi who gave support to the theoretical part of this thesis.

I would like to thank Prof. Quim Borges for his help in the combinatorial aspects related to decision trees used in this thesis.

I would also wish to thank Prof. Josep Maria Basart, whose discussions about the meaning of several words used in this thesis became of crucial importance to understand the concepts of data, information, knowledge and learning.

Prof. Kenneth Zeger also deserves special mention as the basic subject of this thesis was conceived during a research stage at his laboratory in the University of California, San Diego.

This work has been partially supported by a Catalan Government grant, reference FIAP 97-00609, and by Spanish Government project TIC2000-0739-C04-01.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Introduction: Mot obscène
Gustave Flaubert, *Le Dictionnaire des Idées Reçues*

Classification is, basically, the use of previously acquired knowledge. Nature uses knowledge to maintain and to improve life quality, as decisions based on previous experiences are taken when facing new situations. In the basic scheme of what is named intelligence, that is, the endless circle of perception, reasoning and action (shown in Fig. 1), classification plays an important role. By perception we mean acquiring data or information from our environment, by reasoning we mean using this information to take decisions and, by action, we mean converting these decisions in actions that alter our environment. Classification and, more generally, pattern recognition, involve extracting the right information and using it appropriately. This endless circle is what we call *learning*. Systems which evolve under this paradigm are considered to be intelligent, as stated in (Winston, 1992).

Humans are probably the best classification systems in nature, although humans cannot explain the way they do pattern recognition, classification or simply learning. Knowing the way information is stored and decisions are taken are still elusive goals. Nevertheless, the basic concept of learning is that it may be seen as a process that constructs a framework where to put previous knowledge that might be used to face new problems. In this thesis we deal with systems which try to automate this process.

By *machine learning* (Mitchell, 1997) we understand a (probably) deterministic method or technique which allows us to put acquired knowledge into a framework in order to use it automatically. Within the *artificial intelligence* paradigm, it partially involves both perception and reasoning. By *data mining* (Witten and Frank, 2000) we understand about solving problems by analyzing data relationships in such framework taking a more numerical approach. Our goal is to build appropriate machine learning systems to do data mining. By appropriate we mean two things: firstly, a high classification accuracy is needed in order that the classification system might be useful. And

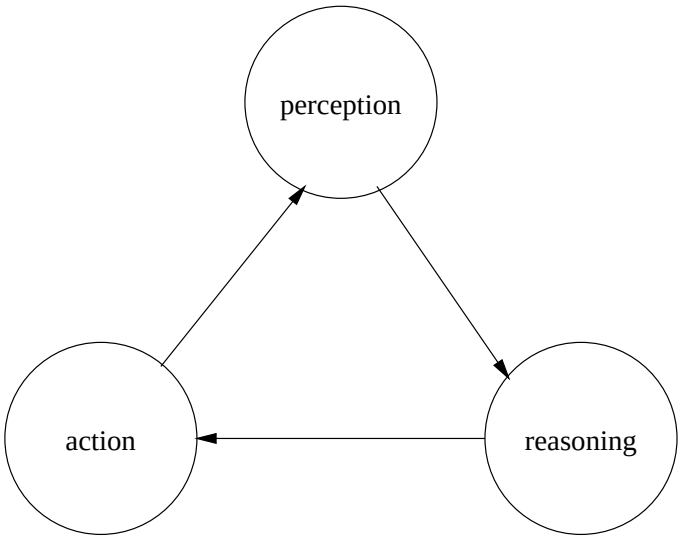

Figure 1: Intelligence as an endless sequence of activities.

secondly, robustness is desired in order that the classification system might be able to maintain such classification accuracy for new information not used during training. When we use previous knowledge during training we talk about *supervised methods* or *supervised learning*, and when we try to extract information without previous knowledge we talk about *unsupervised methods* or *unsupervised learning*. This is also known as *clustering*, as the classification system forms clusters or natural groupings of the input vectors. In this thesis, we deal with supervised classification methods only, but we also try to use clustering techniques for improving classification performance.

Fortunately (for people doing research in pattern recognition), no general classification method exists. As stated in (Antos, Devroye and Györfi, 1999), one can never claim to have a universally superior Bayes error estimation method, no matter how many simulations are performed and no matter how large the sample sizes are. This fact is precisely described in (Duda, Hart and Stork, 2000) with the following statement[1]:

Theorem 1.1 *For any two learning algorithms $g_n^1(f|D_n)$ and $g_n^2(f|D_n)$, $f \in \mathcal{F}$, the following are true, independent of the sampling distribution and the number n of training points in the training set D_n:*

1. *Uniformly averaged over all target functions f, the expected generalization error for both learning algorithms is the same.*

2. *For any fixed training set D_n, uniformly averaged over $\mathcal{F}$, no learning algorithm yields a better*

[1] This theorem is known as the *No Free Lunch* theorem, as suggested by David Haussler.

generalization error.

3. *Uniformly averaged over all priors, the expected generalization error for both learning algorithms is the same.*

4. *For any fixed training set D_n, uniformly averaged over all priors, no learning algorithm yields a better generalization error.*

Nevertheless, there is still room for research in this subject: although it is not possible to find a perfect general learning algorithm, it is possible to tune classifiers for a given problem. As stated in (Duda et al., 2000), it is the match of the learning algorithm to the problem that determines the empirical success. Regarding theoretical issues, in 1984 Valiant (1984) introduced the model for probably approximately correct learning (PAC-learning), and hundreds of papers related to machine learning have been published since then. A good survey may be found in (Kulkarni, Lugosi and Venkatesh, 1998). Several books have also been published since then, among others we should cite (Devroye, Györfi and Lugosi, 1996), (Vapnik, 1998) and (Hastie, Tibshirani and Friedman, 2001) as the most comprehensive texts about machine learning and pattern recognition.

1.1 Supervised classification systems

A supervised classification system is defined by:

1. The information that will be used during the training stage. Using the appropriate vocabulary, we call this stage *the classification features extraction stage*. This stage is very important, as many classification problems could be easily solved using the right set of classification features, see (Kudo and Sklansky, 2000) for example. Original data can also be transformed and new classification features can be computed from it. This is known as *data massaging* (Murthy, 1997). As we are in a supervised classification context, we need the training set to be labeled, that is, for each input vector x we must know its label y in advance, in order to have pairs of the form (x, y). Therefore, we suppose that the input data follows an unknown joint distribution (X, Y).

2. An algorithm to construct such classification system. We call this *the training stage*. This algorithm will use functions from a class of functions $\mathcal{F}$ that will determine its performance. The more complex $\mathcal{F}$ is, the better it is expected to perform. For example, quadratic discriminant functions usually perform better than linear discriminant functions. Nevertheless, following Occam's razor (Duda et al., 2000), $\mathcal{F}$ should be complex enough to capture the underlying structure of the distribution (X, Y), but no more.

3. A measure to evaluate classification accuracy. We call this *the evaluation stage*. As stated above, we want robustness to be a characteristic of our classification system, not only accuracy.

We would also like to have a method to estimate classification accuracy on new data. This is also known as *generalization error* performance.

Therefore, the quality of a classification system is determined by the quality of the training data and the quality (complexity) of the learning algorithm. A fourth subject could be also included: the way the classification system learns new information. Obviously, the simplest way is to add the new information to the original information and repeat the training and evaluation stages, although this scheme may not be optimal for classification systems with a high training cost. In this thesis, we study the three issues described above: namely, the data we have to classify, which learning algorithms we will use and which parameters they have and, finally, how classification performance can be measured.

1.2 Classification and compression

Classification and lossy compression are closely related to each other, as they pursue the same aim: to represent any input vector by an index in a known set. They may be also used to improve each other: a classifier may help an adaptive compression system in order to take advantage of several compression algorithms, depending on which kind of data the classifier detects. For example, document layout recognition systems may be used to segment documents, detecting text image and background areas. Text areas could be compressed using a lossless compression algorithm, while image areas could be compressed using the JPEG standard (Wallace, 1991; Pennebaker and Mitchell, 1993), for example, and background areas could be not coded at all, thus saving bits. A similar strategy is used in adaptive vector quantization (Gersho and Gray, 1992), where different codebooks are switched depending on an external criterion, which may be given by a classifier, for example. On the other hand, lossy compression may also be used as a rude classification system, as the most populated class for each centroid may be used as the classifier output. Vector quantization is directly related to the nearest neighbor classifier, and several vector quantizers have also been used for classification, see (McLean, 1993) for example.

Indeed, data classification may be seen as a special case of lossy compression, as a space partition containing one or more elements is represented by a single element, a label in the classification case or a centroid in the compression case. Basically, both techniques share the same structure: an optional transformation of the input space, a partition of the transformed space, and the selection of a label or centroid for each region in such partition. The main difference between classification and compression is that, in a classification system, it is impossible to reconstruct original data from labels, mainly because labels are not intended to be used that way. The main idea is that lossy compression and classification may share a common framework. Actually, several methods combining classification and compression have been studied and developed, such as self-organizing maps (Kohonen, 1995), tree structured vector quantization (Oelher and Gray, 1995), or learning

vector quantization (Baras and Dey, 1999), for example. The same theoretical framework may be used for both classification and compression, which is really useful due to the large amount of papers related to vector quantization.

Nevertheless, methods combining classification and compression suffer from a major drawback due to different goals. The main goal of a classification system is to minimize misclassification error, whereas the main goal of a compression system is to minimize distortion (usually measured through mean squared error). Even with combinations of both criteria, the better a hybrid system classifies, the worse it compresses, and vice versa. Unless both compression and classification are needed, better classification results are achieved when specific classification systems are used. We have also studied the relationship between classification and compression in order to measure the degradation of classification performance when the input data is compressed at a certain quality level. New applications of data mining such as hyperspectral imaging require further research in this subject.

1.3 Decision trees

Among classification systems, decision trees are one of the most known methods, mainly because they mimic the way humans take decisions. A set of questions which are asked depending on the answer previously obtained determines the final category of the input. The basic idea behind decision trees is recursive partitioning: if a data set cannot be correctly classified with enough accuracy, try to find a question which splits such data set in several subsets, and then repeat this process recursively for each subset.

The study of decision trees, even from an experimental point of view only, arises a great deal of interesting issues, both theoretical and practical. As introduced above, decision trees are grown in a top-down fashion: an initial training set is recursively split until no more splits are possible, using several criteria for determining each action taken. The most important questions related to decision trees are basically those concerning the following subjects:

1. A method to determine the next node to be split. It is called the *node selection* criterion.
2. A method to determine whether a node must be split or not. It is called the *stopping condition* criterion.
3. A method to determine the way a node is split. It is called the *node splitting* criterion.
4. A method to determine which class is assigned to a node. It is called the *labeling rule*.

Decision trees have been extensively investigated and used since the appearance of the first works of Quinlan (1983) and Breiman, Friedman, Olshen and Stone (1984). Each one of the four subjects introduced above has generated its own research subject, specially the third one, which is the most important one. In fact, the four subjects are fully connected to each other, but usually most effort

is put on improving decision tree performance through the optimization of the splitting criterion. In this thesis, we study several questions related to the subjects stated above.

Classical decision trees suffer from a major drawback, as stated in (Hastie et al., 2001, p. 313): their predictive power is limited. This is partially caused because of the fact that decision trees are unstable classifiers, since the outcome of the decision tree learning algorithm is very dependent on the training set. The intrinsic characteristics of the training set determine the structure of the decision tree, among them training set size (n), data dimensionality (d) and the number of different classes (K) are specially critical. It is well known (Breiman et al., 1984) that decision trees are not accurate classifiers when d or K are large. Traditionally, decision trees have been built using the basic algorithm described in (Breiman et al., 1984) without any special consideration on tuning the training stage parameters for a given data set. This usually causes decision trees to perform poorly when compared with other classifiers which use more complex decision functions, such as neural networks or support vector machines, for example.

In this thesis, we study how to to break the classification problem into a sequence of partial classification problems, in order that the easiest part of the original problem is solved first using a simple decision tree, and the rest is solved applying the same criterion recursively. We tune each stage varying several parameters of the learning algorithm, such as maximum decision tree depth, the splitting criterion and a threshold that determines whether the outcome of the labeling rule can be used or not, yielding to a partial classification of the input training set. Experiments show that it is possible to improve classification accuracy using ensembles of small decision trees instead of using a single large decision tree. As the main contribution of this thesis, we propose a novel way to ensemble decision trees, called *progressive decision trees*, which follows this strategy of cascading several partial classifiers.

1.4 Thesis structure

This thesis is structured as follows:

Chapter 2 describes decision trees and the classical algorithms used for constructing them, including criteria obtained experimentally which may be useful to build better decision trees. Basic definitions and properties of decision trees as well as general thoughts are presented in this chapter.

The following chapters are the core of this thesis, and they are fully connected to each other. In Chapter 3, we introduce progressive decision trees, a novel approach to the construction of ensembles of decision trees, with the aim to overcome some of the problems related to decision trees. We compare our ensemble to other types of ensembles that combine several classifiers into a single one, trying to find similarities and differences. As our ensemble may be seen as a special case of cascading (Gama and Brazdil, 2000), we describe three types of cascading ensembles which may be used to build progressive decision trees.

All the experiments and simulations that give support to our ensemble are compiled in Chapter 4, together with other experiments which can be considered as empirical support for growing better decision trees. A paper summarizing the most important contributions of this thesis can be found in (Minguillón and Pujol, submitted). The different types of progressive decision trees are empirically evaluated with a subset of the UCI[2] collection (Blake and Merz, 1998) of standard data sets, which is the common reference for the machine learning community. Experimental results with real data sets (document layout recognition (Minguillón, Pujol and Zeger, 1999), hyperspectral imaging (Minguillón, Pujol, Serra and Ortuño, 2000; Minguillón, Pujol, Serra, Ortuño and Guitart, 2000), brain tumor classification (Tate, Ladroue and Minguillón, 2002; Minguillón, Tate, Arús and Griffiths, 2002; Minguillón, Tate, Griffiths, Majos, Pujol and Arús, 2002)) are also presented in this chapter.

In Chapter 5, we compare our ensemble with other ensembles of decision trees under a theoretical framework, following previous works of Schapire, Freund, Bartlett and Lee (1998), Golea, Bartlett, Lee and Mason (1998) and Mason, Bartlett and Golea (to appear) among others. Our idea is to see that progressive decision trees have some properties which make it possible to perform better than single decision trees under certain conditions, and also to study the parameters involved in the training stage that determine the kind of cascading ensemble.

The conclusions and contributions of this thesis are summarized in Chapter 6, and some guidelines for further research in this subject which have arisen from the research carried out in this thesis are also outlined.

Finally, two appendices are included in order to improve the understanding and readability of this thesis: Appendix A summarizes the standard data sets used in the experiments performed in this thesis, and Appendix B may be used as a notation table. An index of the most important terms is also included.

This thesis has been written using LaTeX, and figures have been created using Xfig and GNUPLOT.

[2] University of California, Irvine.

Chapter 2

Decision trees

Méthode: Ne sert à rien
Gustave Flaubert, *Le Dictionnaire des Idées Reçues*

In this chapter we define the basic concepts used in this thesis related to decision trees and more precisely, to data mining using decision trees. A previous clarification: decision trees are also known as classification trees, as the set of questions or decisions that has to be solved to reach a leaf is used to give an answer to the question "which class is X?". Nevertheless, decision trees can also be used for regression (Breiman et al., 1984) or vector quantization (Linde, Buzo and Gray, 1980), for example. In the context of this thesis, both terms are indistinguishable one from the other, although we prefer the more general version "decision trees".

Throughout this and the next chapters, we will follow the notation defined by Breiman et al. (1984) for basic tree definitions, and the notation defined by Devroye et al. (1996) for error measures and estimations. A notation table summarizing all mathematical notation used can be found in Appendix B.

2.1 Introduction

Since their development by Breiman et al. (1984), who studied classification and regression trees (CART), decision trees have been extensively used in several areas related to pattern recognition and data mining: optical character recognition (Casey and Nagy, 1984), medical diagnosis (Kononenko, 1993), document image analysis and remote sensing, for example, are maybe the best known applications, but also financial analysis, astronomy and molecular biology applications of decision trees among many other are emerging. Decision trees were previously described in several papers describing the process of converting a decision table into a decision tree (Hartmann, Varshney, Mehrotra and Gerberich, 1982). An excellent although a bit out of date survey of decision trees

may be found in (Safavian and Landgrebe, 1991), and a shorter but newer survey may be found in (Quinlan, 1996b). Finally, a large compilation about new material, applications and algorithms related to decision trees may be found in (Murthy, 1997). Another author that has introduced a lot of new concepts and algorithms related to decision trees is Quinlan (1990), who developed both ID3 (Quinlan, 1983) and C4.5 (Quinlan, 1992) algorithms[1].

But decision trees are not only used as classifiers in so many applications, new algorithms related to training decision trees have been developed by Gelfand, Ravishankar and Delp (1991), Loh and Vanichsetakul (1988), and recently an algorithm which uses a new criterion called *degree of linear separability* has been introduced by Shah and Sastry (1999), showing the importance of decision trees within the research of the machine learning community. New techniques such as boosting (Freund, 1995) and bagging (Breiman, 1996a) have also renewed the interest in classification systems using decision trees. Decision trees have been also studied under a statistical approach using probability models, as described in (Jordan, 1994).

When compared to other pattern recognition methods (Jain, Duin and Mao, 2000), decision trees seem to be relegated to a second line, behind artificial intelligence related methods (neural networks (Bishop, 1998), self-organizing maps (Kohonen, 1995), support vector machines (Cortes and Vapnik, 1995; Vapnik, 1998), etc.) which are the most popular methods among the machine learning community. The reason is that it is easy to think that decision trees are basic tools which only use orthogonal or linear hyperplanes to compute boundaries, while the mentioned methods take advantage of more complex boundary algorithms. Several authors have also proposed methods combining both approaches (Sethi, 1995; Schmitz, Aldrich and Gouws, 1999; Bennett and Blue, 1998). It is important to see that a classification procedure should not only produce accurate classifiers at a reasonable cost, but also provide insight and understanding into the predictive structure of the data. Distance based methods or non linear boundaries do not allow such analysis. Decision trees satisfy these requirements and present many other properties which make them an interesting tool for data inspection:

1. Feature selection: usually, the most important classification features are selected first for data splitting, while other classification features are never selected. This information may be used for feature selection, reducing feature complexity and, therefore, classification cost.

2. Data inspection: when orthogonal hyperplanes are used as splits, each internal node uses only one classification feature for data splitting. This is directly related to feature selection as it may be used to detect the most important features. Furthermore, each set of questions that has to be answered from the start of the tree until a leaf is achieved yields information about feature relationships and, therefore, information about internal data structure.

[1] Software packages and further references for both algorithms (and many other) may be found in `http://kiew.cs-dortmund.de:8001:/mlnet`

3. Categorical variables: unless other distance based methods, linearly categorical variables may be easily included. In real scenarios such as medical diagnosis or credit card scoring, categorical variables are usually the most common type of available data as classification features. Classical approaches for dealing with categorical variables may also be used (see (Andersen, 1997)). Binary variables may be easily include as well.

4. Missing values: another interesting property of decision trees is the possibility of handling missing values, that is, data vectors that have one or more classification features unknown, which is a common situation in real case scenarios.

5. In some problems, given a input vector, what is wanted is an estimate of the probability that the case is in a given class, giving an estimation of the relative probability of the vector for each class. This is easily computed at each leaf of a decision tree.

6. It is also easy to establish a parallelism between classification and regression, the same tree scheme may be used for both problems changing only the final question or labeling function. Actually, both problems are closely related, see Frank, Wang, Inglis, Holmes and Witten (1998) for example, where regression trees are successfully used for classification. Decision trees may be also used for compression (TSVQ and so, see (Gersho, 1982) for example) and, obviously, combining both classification and compression in a single decision tree is also possible (Oelher and Gray, 1995; Perlmutter, Perlmutter, Gray, Olshen and Oehler, 1996).

Maybe the most interesting feature of decision trees is their flexibility, as they are grown using always a known set of rules and criteria. Decision trees allow the user to integrate all knowledge about a problem into a simple and intuitive framework. As stated in McLachlan (1992, pp. 324), classification and regression trees add a flexible non-parametric tool to the data analyst's arsenal. Its potential advantages over traditional approaches are in the analysis of complex nonlinear data sets with many variables.

2.2 Tree definition

Decision trees have been profusely studied in the past years, so there are many useful tools (Breiman et al., 1984; Chou, Lookabaugh and Gray, 1989; Devroye et al., 1996) to describe and to analyze them. Nevertheless, the same flexibility of decision trees makes them hard to analyze under a theoretical approach. In this section we give a basic definition of what a decision tree is, which will be refined in the following sections.

Definition 2.1 *A tree T is a directed, acyclic graph where each node has either no child (in that case it is called a terminal node or a leaf) or exactly c children (and then it is called an internal node or decision node), and each child has exactly one parent except the top node of the tree, which*

is called the root. The set of all leaves of T is denoted by $\tilde{T}$, and the size of T and its number of leaves are denoted by $|T|$ and $|\tilde{T}|$ respectively.

Usually, decision nodes are questions that can be answered using only true or false, so $c = 2$. In this case, we use the term *binary* decision tree. This kind of trees is useful for classification, as a set of simple questions is used to determine the class of an input vector. The case $c > 2$ is often used when categorical variables taking several values are present in the training set, although this situation can also be (suboptimally in a classification cost sense) represented with a binary decision tree, with a reduced training cost. In this thesis we deal with binary decision trees only, so we will drop the term binary except when necessary.

Fig. 2 shows an example of a decision tree, taken from Breiman et al. (1984), which was developed as a method to identify high risk patients among those that suffered a heart attack.

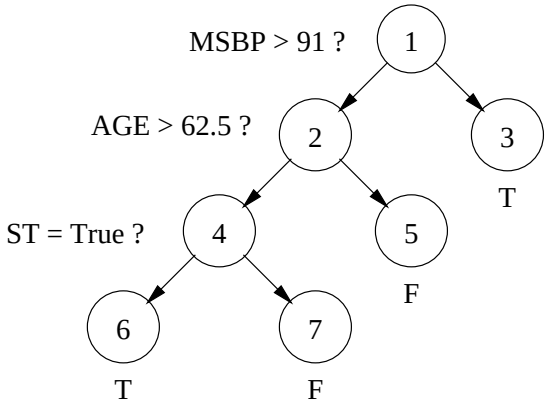

Figure 2: Example of decision tree.

In this example, node 1 is the root of the tree. Nodes 1, 2 and 4 are the internal nodes or questions: node 1 asks "is the minimum systolic blood pressure over the initial 24 hour period greater than 91?", and left branch is taken if the answer is "yes" and right branch otherwise. Node 2 asks for the age of the patient and, finally, node 3 asks "is sinus tachycardia present?". On the other hand, nodes 3, 5, 6 and 7 are the leaves, which are labeled true (T) or false (F) depending on whether the patient is classified as high risk or not. This tree was created using a data set with 19 classification features, but only the three questions showed here where considered important by the decision tree in order to find a class for each patient. This fact allows the experts to identify the most important variables and to find new relationships between them, which may be even more important than the classification itself.

2.3 Growing a decision tree

Decision trees are the result of a supervised learning method, that is, a labeled data set is needed for training. Such data set is called the *training set*, and classification performance is directly related to its properties. Hopefully, the training set is as large as possible, and it represents all possible classes. As stated in (Breiman, 1996a), decision trees are *unstable* classifiers, that is, small perturbations in the raining set may cause large differences in internal tree structure.Therefore, a large training set is preferable, although this is usually not the common situation.

Let D_n be a labeled training data set of length n, that is, a sequence of n pairs (X_i, Y_i), $X_i \in \mathbb{R}^d$, $Y_i \in \{0, 1, \ldots, K-1\}$, $i = 1, \ldots, n$, where d is the dimension of the input space[2] and the number of classification features, and $K \geq 2$ is the number of classes present in the training set. Y_i is the label or class assigned to vector X_i, and (X_i, Y_i) are thus instances of a joint distribution (X, Y), which we are trying to learn.

Hopefully, D_n represents all the possible pairs (X, Y), although this is a bad assumption even for really large training data sets. Nevertheless, because the real distribution of (X, Y) is not known (in that case no decision tree would be needed), we must trust our training set to be representative and try to extract the most important information from it in order to build a good classifier.

Decision trees are built using an algorithm based on space recursive partition, as follows:

Growing algorithm
use D_n as the initial data set
while a stopping condition is not satisfied
    select a data set
    split it in s disjoint subsets
end

The growing algorithm is controlled by a stopping condition, a way to select the node to be split and a method for splitting nodes. This general algorithm, which can be seen as a greedy algorithm, may lead to bad decision trees when training data is not representative or when its control parameters are not well defined, causing what is known as overfitting. In Sect. 2.6 we will discuss an algorithm due to Breiman et al. (1984) which is called *pruning*[3] that can be used to improve decision tree performance reducing its specificity. The pruning algorithm takes a large tree as its input and it computes the optimal subtree (for a given criterion) which minimizes misclassification error on a second training set. The fundamental principle underlying the tree growing algorithm is that of simplicity: decisions that lead to simpler, more compact trees with fewer nodes are preferred, following the Occam's razor rule (Duda et al., 2000), which states that the simplest model that explains data is the one to be preferred. Notice also that this algorithm needs to be applied again

[2] Notice that the input space may be a cartesian product of simpler spaces, including also discrete spaces.
[3] It is also known as the BFOS algorithm.

when new training data arrives. An incremental version is described in (Utgoff, 1989), although it has not been widely used.

2.3.1 Partitions

Due to $n < \infty$, the growing algorithm always stops generating a finite partition of the original data set and, by extension, a finite partition of the input space.

Definition 2.2 *A finite partition $\mathcal{P}$ of $\mathbb{R}^d$ is a finite collection $\{A_1, \ldots, A_P\}$ of subsets of $\mathbb{R}^d$ such that $\cup_{j=1}^{P} A_j = \mathbb{R}^d$ and $A_i \cap A_j = \emptyset$ if $i \neq j$. Each set A_j is called a cell of the partition $\mathcal{P}$.*

Every partition induced by a decision tree is a valid partition of the input space, but not vice versa (finite holes, for example, cannot be created by hyperplanes). The study of partitions induced by decision trees is an interesting subject because partition properties are directly related to tree shape and other related properties.

The partition obtained may be also described using a tree structure: internal nodes are questions, $t : \mathbb{R}^d \to \{1, \ldots, s\}$, and leaves are data sets which are labeled according to a fixed labeling rule. Usually, questions are hyperplanes, that is, $H\mathbf{x} \leq h$, so $s = 2$. Leaves are data sets which are labeled trying to minimize misclassification error. The labeling rule is as follows: given a node t, label it as

$$l(t) = \arg_j \min\{r(t) = \sum_{k=0}^{K-1} C(j,k)p(k|t)\} \quad j = 0, \ldots, K-1 \tag{2.1}$$

where K is the number of classes, $C(j,k)$ is the cost of misclassifying class j as class k, and $p(k|t)$ is the estimated probability of class k in t.

Such a labeling rule is hard, that is, a single label is chosen as $l(t)$ and ties must be solved using an appropiate criterion. A possible extension is the use of posterior class probabilities, or probabilistic concepts, as defined in (Kearns and Schapire, 1994).

In a natural way, a partition $\mathcal{P}_T$ generated by a decision tree T induces a random variable $\Psi_{\mathcal{P}_T}$, which takes the leaf value $t \in \mathcal{P}_T$ with probability $\mathbf{P}_T(t) = p(t)$. We can define several measures on this partition which provide us information about tree shape and specificity, as entropy, for example. Entropy, which is defined[4] as $\mathbf{E}[-\log \mathbf{P}(\Psi_{\mathcal{P}_T})]$ is an interesting measure because allows us to compare partitions and, therefore, decision trees. As stated in (Yockey, 1992), a result due to Kolmogorov is that two probability spaces are isomorphic if and only if they have the same entropy. We will use entropy and other information theory related measures to compare decision trees and to measure their shape and specificity.

[4] Here $\mathbf{E}$ denotes expectation.

2.4 Growing algorithm parameters

Growing a decision tree consists, therefore, in determining a set of parameters that are used during the training stage. These parameters are the stopping condition, the node selection criterion, and the node splitting criterion.

2.4.1 Stopping condition

Stopping condition may be tree based or node based, depending on whether a condition on the whole tree structure or a condition on every leaf of the tree is satisfied. In other words, stopping conditions may be *global* or *local* respectively.

The simplest tree based stopping conditions are a maximum number of iterations or a maximum tree depth, for example. Nevertheless, both conditions are unrelated to misclassification error and, therefore, they yield bad decision trees, as stated by Breiman et al. (1984). A more interesting stopping condition directly related to misclassification error is to measure tree performance for the training set (denoted by $\hat{L}_n(T)$, which will be defined in Sect. 2.5), and then stop when this value falls below a threshold ϵ. This does not prevent overfitting, though.

Regarding node based stopping conditions, they are related to the node selection criteria. Instead of computing a function over the whole tree, the growing algorithm is stopped when a node is selected and it satisfies a given criterion such as node depth (yielding to limited training depth trees, see Sect. 4.3), for example. In this case, the stopping criterion and the selection criterion may be joined in a single function that is computed for each node, and the node yielding the smallest (or the largest) value is chosen for splitting, until this value is too small or too large for a given threshold. Early stopping may avoid overfitting but it may cause a bad performance.

If pruning is going to be used after the training stage, usually the imposed condition is $\hat{L}_n(T) = 0$, that is, a perfect tree T is grown (see Def. 2.5). This is only possible, though, when there are no two pairs (X_i, Y_i) and (X_j, Y_j), $i \neq j$ such that $X_i = X_j$ but $Y_i \neq Y_j$. Those pairs not satisfying such condition should be removed from the training set D_n in order to avoid cheating our growing algorithm. Decision trees are very sensitive to training data quality.

2.4.2 Node selection

For each leaf $t \in \tilde{T}$ an index selection function is computed. The leaf minimizing or maximizing such index is selected to be split. Usually, this criterion is directly related to the node splitting criterion, as the leaf t yielding the maximum decrease in tree impurity is selected (see next section). This is also a tree based or global criterion. For example, the leaf t maximizing the decrease in tree misclassification error is also a global criterion, but according to Breiman et al. (1984), it may yield bad performance results.

Global criteria are computationally expensive, so it would be interesting to have local criteria that could be used to select the next node to split. Possible criteria are selecting the node with the largest misclassification error, selecting the most populated node, an so, but this may generate suboptimal trees.

Both options are greedy, so it is naturally of interest to explore ways to improve such strategies. This is partially achieved with lookahead techniques, which try to find the best sequence of nodes to split. Surprisingly, using lookahead in decision tree growing algorithms does not improve tree performance (Murthy, 1997), so it is better to put more efforts in other areas of the growing algorithm.

Once again, if pruning follows the training stage, all nodes will be selected sooner or later and therefore split until all of them are perfect, so there is no need to compute anything for node selection. One could select nodes to be split depending on its number, for example, without any cost.

2.4.3 Node splitting

Due to the nature of the growing algorithm, node splitting criterion is the most critical issue in decision trees, although some authors (Breiman et al., 1984, p. 38) point out that the properties of the final tree are insensitive to the choice of the splitting criterion within a wide range of splitting criteria. Nevertheless, node splitting is truly a key issue in decision trees because of its importance: a bad split at a certain level may lead to inefficient subtrees in the lower levels. Furthermore, a lot of effort has been put in studying splitting criteria, see (Mingers, 1989b; López de Màntaras, 1991; Buntine and Niblett, 1992; Martin, 1997; Drucker, 1999; Shih, 1999; Drummond and Holte, 2000) for example. Hinging hyperplanes, which extend the concept of split, have been studied in (Breiman, 1993). Once again, though, pruning seems to be the right way to obtain good decision trees avoiding overfitting, regardless the used splitting criterion.

2.4.3.1 Splitting through impurity criteria

Splits are usually selected using an *impurity* criterion, trying to find the split that maximizes a decrease of impurity. Impurity measures how mixed a leaf data set is, that is, the proportion of elements of different classes in a same data set. Another way to select splits is the *twoing* criterion (Breiman et al., 1984, pp. 38), which is not related to a node impurity measure.

Definition 2.3 *Let Δ_K be the set of all K-ary probability distributions*

$$p = \{p_0, \ldots, p_{K-1}\}$$

which satisfy the following conditions:

$$p_j \geq 0 \quad \forall\, j = 0, \ldots, K-1$$

and

$$\sum_{j=0}^{K-1} p_j = 1.$$

A function $\Phi : \Delta_K \to \mathbb{R}$ *is called an impurity function if satisfies the following conditions:*

a) $\Phi(p_0, \ldots, p_{K-1}) \geq 0$.
b) $\Phi(p_0, \ldots, p_{K-1}) = 0 \Leftrightarrow \exists\, p_j = 1, p_k = 0 \quad \forall\, j \neq k$.
c) Φ *achieves its maximum when* $p_0 = p_1 = \ldots = p_{K-1} = \frac{1}{K}$.
d) $\Phi(p_0, \ldots, p_{K-1})$ *is a symmetric function of* $p_0, p_1, \ldots, p_{K-1}$.
e) $\Phi(p_0, \ldots, p_{K-1})$ *is a concave function of* $p_0, p_1, \ldots, p_{K-1}$.

Examples of impurity functions are:

1. Probability of misclassification: $\Phi(p) = 1 - \max\{p_j\}$. When $K = 2$, it is usually defined as $\Phi(p) = \min\{p, 1-p\}$, also called the Bayes error L^*.

2. Gini function (Breiman et al., 1984): $\Phi(p) = 1 - \sum_{i=0}^{K-1} p_j^2$. When $K = 2$, $\Phi(p) = 2p(1-p)$. In other contexts, such as molecular epidemiology, the Gini criterion is also known as *heterozygosity* (Li, 1997). It is also known as the nearest neighbor error (Devroye et al., 1996, p. 22).

3. Entropy: $\Phi(p) = -\sum_{i=0}^{K-1} p_j \log(p_j)$. Entropy is a measure of uncertainty in information theory (Cover and Thomas, 1991) with countless applications in many branches of computer science, mathematical statistics, biology and physics. As stated in Devroye et al. (1996, p. 27), the nearly monotone relationship between the entropy impurity function and the misclassification error will see lots of uses.

4. R-norm (Boekee and der Lubbe, 1980): $\Phi(p) = \frac{R}{R-1}[1 - (\sum_{j=0}^{K-1} p_j^R)^{1/R}]$. The R-norm is a class of impurity functions depending on the parameter $R > 0$. When $R \to 1$, R-norm is the entropy impurity function, and when $R \to \infty$, is the probability of misclassification. When $K = 2$ and $R = \frac{1}{2}$, R-norm is the Matushita error, which is related to the Bhattacharyya measure of affinity (Devroye et al., 1996, p. 23).

Fig. 3 shows the shape of these impurity functions in the interval $[0, 1]$ when $K = 2$. They have been normalized, so the maximum value achieved when $p_0 = p_1 = 1/2$ is also $1/2$. From innermost to outermost, Bayes error, R-norm with $R = 2$, the Gini criterion, Entropy and R-norm with $R = 1/2$ are plotted. This ordering is directly related to impurity criterion performance and behavior, as we will discuss later in Sect. 4.2.

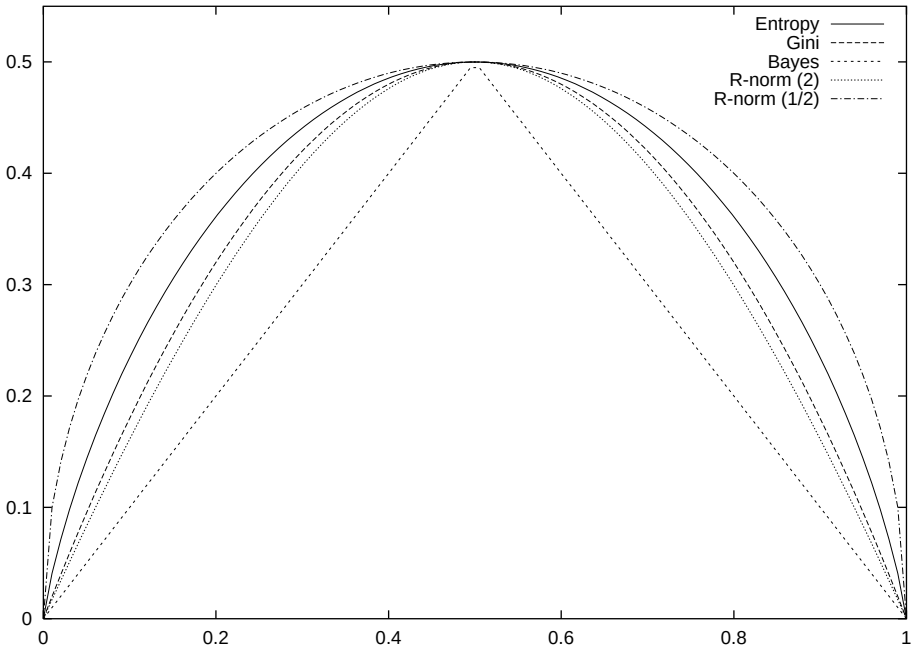

Figure 3: Impurity functions shape.

Node impurity is defined as $i(t) = \Phi(p(0|t), \ldots, p(K-1|t))$, where $p(j|t)$ is the proportion of the elements belonging to class j in node t. Each $p(j|t)$ may be computed as

$$p(j|t) = \frac{\pi_j N_j(t)/N_j}{p(t)}$$

where $N_j(t)/N_j$ is the proportion of elements of class j in t, and π_j is the *a priori* (or prior) probability of class j. Priors may be explicitly given (as part of the knowledge that we have about the classification procedure) or may be computed as the relative proportion of elements of class j in the training set.

Definition 2.4 *A leaf $t \in \tilde{T}$ is called pure (or perfect) if $i(t) = 0$.*

By impurity function definition, this only happens when all elements in t are the same class. Analogously, tree impurity is a linear combination of all leaf impurities, as follows:

$$I(T) = \sum_{t \in \tilde{T}} p(t) i(t)$$

where $p(t)$ is the probability of t (recall that all tree leaves define a partition of the input space). Analogously,

Definition 2.5 *A tree is perfect if* $I(T) = 0$.

It is easy to see that if T is perfect, its misclassification error for the training set is also zero. This definition of perfect trees may be also found in (Quinlan, 1990), as stated in (Safavian and Landgrebe, 1991).

Splitting a node never increases tree impurity. Let s be a split on a leaf $t \in \tilde{T}$, that is, s splits t creating a new tree T' with a pair of new leaves t_l, t_r, as shown in Fig. 4, and then measure its impurity decrease as

$$\begin{aligned} \Delta_T(s,t) &= I(T) - I(T') \\ &= \sum_{t \in \bar{T}} p(t)i(t) - \sum_{t' \in \bar{T}'} p(t')i(t') \\ &= p(t)i(t) - (p(t_l)i(t_l) + p(t_r)i(t_r)). \end{aligned}$$

It is easy to see from $i(t)$ properties (which are caused by Φ properties) that $\Delta_T(s,t) \geq 0$, with equality if, and only if, $p(j|t_l) = p(j|t_r) = p(j|t)$, $j = 0, \ldots, K-1$. A formal proof can be found in (Breiman et al., 1984, p. 94). Then, the splitting rule is to select the split s in t which maximizes $\Delta_T(s,t)$.

2.4.3.2 Splitting through the twoing criterion

When the twoing criterion is used, at a node t the splitting rule consists in choosing the split s that maximizes

$$\Delta_T(s,t) = \frac{p(t_l)p(t_r)}{4} \left[\sum_{j=0}^{K-1} |p(j|t_l) - p(j|t_r)| \right]^2 . \tag{2.2}$$

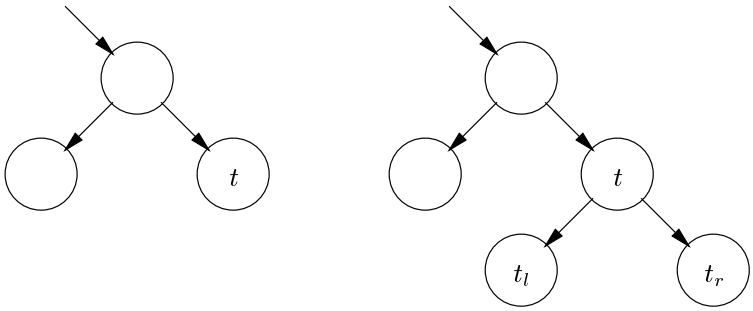

Figure 4: Splitting a node.

Gini criterion looks for the largest class in the selected node and strives to isolate it from all other classes. Gini criterion attempts to separate classes by focusing on one class at a time. It will always favor working on the largest or, if costs or weights are used, the most important class in a

node. While this approach might seem short sighted, Gini performance is frequently so good that one should always experiment with it to see how well it does. Gini criterion is usually the default rule in CART-like software precisely because it is so often the best splitting rule. The philosophy of the twoing criterion is far different than that of Gini. Rather than initially pulling out a single class, the twoing criterion first segments the classes into two groups, attempting to find groups that together add up to 50 percent of the data. The twoing criterion then searches for a split to separate the two subgroups, and so on. The entropy criterion, although is closer to the Gini criterion, is very similar to twoing in practice, and it strives for similar splits. In Sect. 4.2 we test several splitting criteria in order to rank them experimentally.

Another interesting subject is the combination of several splitting criteria at different levels of the decision tree. In (Brodley, 1995), an algorithm that applies a split criterion suited to the location of the decision node in the tree is empirically evaluated. The basic idea is to use the Bayes error criterion at leaves and other criteria at inner nodes.

2.4.3.3 Construction of splits

There are several ways to construct splits using only training data available at a given node t which has been selected to split:

1. Use only orthogonal hyperplanes, that is, only one variable is considered for splitting. Splits are questions "$x_m \leq h$?". This may lead, though, to poor performance when class structure depends on combinations of variables. On the other hand, its simplicity and low computational cost make this method be the most used. It also gives information about which variables are important at a given level, allowing the user to do simple internal data structure inspection.

2. When variable combinations are allowed, decision trees yield results competitive with or even better than linear discriminant analysis (Jobson, 1992; McLachlan, 1992). In this case splits are questions "$\sum_{i=0}^{d-1} H_i x_i \leq h$?", and they are also called multivariate decision nodes or *oblique* decision nodes. Although general hyperplanes achieve the best results, using variable combinations has a high computational cost and, furthermore, all hyperplanes computed using training data statistics are usually too specific leading to poor results. Loh and Vanichsetakul (1988) have developed an approach known as FACT, which has multivariate splits that are distribution dependent at each node, assuming a Gaussian distribution. A distribution free algorithm based on linear programming is due to Brown and Pittard (1996). A method based on a linear discriminant classifier is due to Gama (1999). Other remarkable methods are due to Liu and Setiono (1998), where a feature space transformation is used to compute the splitting hyperplanes, and also due to Brodley and Utgoff (1992), where a multiclass linear discriminant is used in combination with a feature removal algorithm. Finally, Bennett and Blue (1998) have developed a method which uses support vector machines at each decision node.

3. For a categorical variable x_m taking J different values, splits are questions "$x_m \in \mathcal{A}_{\mathcal{J}}$?" where $\mathcal{A}_{\mathcal{J}}$ is a set containing a non trivial subset of $\mathcal{J} = \{1, \ldots, J\}$. Therefore, $2^{J-1} - 1$ non trivial subsets should be tested, which is computationally very expensive when J is large. For several categorical variables considered at once, the number of possible combinations grows even faster due to boolean combination structure. Breiman et al. (1984, p. 101) reduce the number of subsets to J, which provides a considerable improvement in computational efficiency. Chou (1991) describes an optimal partitioning method for classification and regression trees based on divergence measures, trying also to reduce computational complexity.

4. Finally, user knowledge and expertise in the classification problem may be used introducing forced splits at a certain level. For example, in expert systems such as those used in medical diagnosis, a first question on a variable which is an analytical result may be followed by a forced question trying to determine a concrete pathology. Although this method may be useful to combine automatic and manual (non-automatic) learning, we do not consider it because it does not allow a complete analysis to be carried. Nevertheless, decision trees may be used as a helpful tool for internal data structure inspection, helping experts to discover new relationships between classification features.

We do not consider splits that do not use labels Y_i as information. This splits lead to trees with the X-Property, as defined by Devroye et al. (1996). Other types of trees based on other types of splits are also discarded, such as spheric trees, for example, which are used in tree structured vector quantization (Gersho and Gray, 1992; Devroye et al., 1996). Nevertheless, a simple transformation of the original data may be used to find new non-linear splits, specially when several $x_i \in \mathbb{R}$ form a subspace of $\mathbb{R}^d$. A good example could be using polar coordinates instead of Cartesian coordinates (or combined with).

2.4.4 Growing algorithm complexity

It is easy to see that the number of possible trees generated by the growing algorithm grows exponentially with the number of iterations and the number of available training vectors. After each stage, the number of leaves of the tree grows in $c-1$ units, where c is the number of disjoint subsets in which a leaf is split. When $c = 2$, we start with a trivial tree with only one leaf, after the first iteration the tree has two leaves, and so on. The sequence of number of leaves available to split of all possible trees is $1, 2, 5, 14, \ldots$, which may be recognized as the sequence of Catalan numbers[5] (Sloane, 2000). From its definition, it is easy to see that this sequence grows exponentially with the number of leaves c, that is, the number of stages.

Furthermore, it is also known that constructing an optimal decision tree is a NP-complete problem (Hyafil and Rivest, 1976), optimal in the sense of minimizing the expected number of tests required

[5]Catalan numbers are defined by $C(c) = (2c)!/(c!(c+1)!) = \binom{2c}{c}/(c+1)$, $c \geq 1$.

to classify an unknown sample. The problem of finding the split that minimizes the number of misclassified points, given two sets of mutually exclusive points, is also NP-complete (Heath, 1992). Therefore, trying to find by exhaustive search the best leaf to split and the best split is an infeasible problem, so heuristic approaches need to be considered.

2.4.5 Decision tree complexity

Once a decision tree has been built, two measures are used as complexity estimators of internal decision tree structure, the average rate $R(T)$ (or simply R) and the maximum depth $\overline{d}(T)$ (or simply $\overline{d}$).

2.5 Misclassification error estimation

Tree performance is usually measured using an error-counting estimator $\hat{L}_n$, defined as follows:

$$\hat{L}_n(T) = \frac{1}{n} \sum_{i=1}^{n} 1_{\{T(X_i) \neq Y_i\}} \tag{2.3}$$

where 1 is an indicator function (it is one when the condition evaluated is true and zero otherwise), and $T(X_i)$ is the label assigned by T to the input vector X_i.

Therefore, $\hat{L}_n(T)$ counts the number of mistakes that T makes classifying D_n. Nevertheless, we are more interested in estimating the true error probability, that is, $L_n(T) = \mathbf{P}\{T(X) \neq Y \mid D_n\}$ of a decision tree T for all possible pairs (X, Y), not only for those in D_n. The main reason is that $\hat{L}_n$ may be very much smaller than L_n, depending on the decision tree grown for D_n and, therefore, may be unreliable. Using $\hat{L}_n$ to estimate decision tree performance is also known as the *resubstitution method* (Breiman et al., 1984).

Unfortunately, growing a decision tree from a data set D_n and then using $\hat{L}_n$ to measure tree classification performance suffers from a major drawback. Usually, the obtained tree performs well for the training data set D_n, but it does poorly for other data sets, yielding a higher misclassification error. This undesirable phenomenon is called *overfitting*. The main problem is that misclassification error is optimistically underestimated, usually because the same data is used for both training and validation. For example, if the training set D_n is split until all leaves in T are perfect or pure, $\hat{L}_n(T) = 0$, which is clearly unreliable.

What we want is to study the error generalization capability of a given decision tree. In Sect. 5.1.1 we will take a more theoretical approach for this subject. In a more practical sense, there are several methods (Jain et al., 2000) that can be used to obtain a better estimation of L_n without having extra training data (which may be useful for small training sets, see (Raudys and Jain, 1991; Jain and Zongker, 1997; Raudys, 1997) for example):

1. *Hold-Out method:* use $n^* < n$ vectors for training and $n - n^*$ vectors for L_n estimation. This method is only recommended for large n, and usually is pessimistically biased. Furthermore, different partitions may lead to different estimations. As a rule of thumb (Breiman et al., 1984), use $n^* = \frac{2}{3}\, n$.

2. *Cross-Validation method:* build $\binom{n}{k}$ decision trees using k vectors in each training set and $n-k$ vectors to estimate L_n, and then average all obtained estimations. When $k = 1$ this method it is also known as the *leave-one-out method*. Estimation is unbiased, but it has a large variance and a higher computational cost when n is large.

3. *N-fold Cross-Validation method:* split the n vectors in N subsets containing approximately the same number of elements, and for each subset build a new training set using $N-1$ subsets and use the selected subset to estimate L_n, and then average all obtained estimations. Depending on N, it may be prohibitive. Usually this method is repeated k times so a total of $N \times k$ experiments are carried out (each repetition using a different partition of L_n). According to (Hastie et al., 2001), a value between 5 and 10 for N is usually a good choice.

4. *Bootstrap method:* generate many bootstrap sample sets of size n by sampling with replacement, and then average all obtained estimations. Variance of a bootstrap estimator is lower that that of the commonly used leave-one-out method, but only when sample size is extremely small or when the classification error is large (Raudys, 1988).

Nevertheless, all of these methods (except the hold-out method) use training data for both decision tree construction and misclassification error estimation, which may be not fair in a strict error estimation sense. A study of its accuracy may be found in (Kohavi, 1995). It would be better to use a second data set starting with the decision tree generated by D_n and then to improve such tree in order to remove those paths that are too specific of the training set. This method is known as *pruning*. Actually, pruning may use the hold-out method in order to have different training and corpus sets and may also be combined with the N-fold cross-validation method when N is small ($N = 2$ or $N = 3$).

2.5.1 Confusion matrix

Although Eq. (2.3) allows us to compute the classification accuracy, it does not give information about what kind of mistakes makes a given decision tree. Let D_n be the training set with K classes, each class containing n_k elements. Let $n_{j,k}^T$ be the number of elements of class j classified as class k by T. Then, the *confusion matrix* of T is:
where n_k^T is the number of elements of D_n classified as class k. Notice that Eq. (2.3) may be computed from the confusion matrix, it is the sum of all elements not in the diagonal divided by n.

$j\backslash k$	0	1	...	$K-1$	$\sum$
0	$n^T_{0,0}$	$n^T_{0,1}$	...	$n^T_{0,K-1}$	n_0
1	$n^T_{1,0}$	$n^T_{1,1}$	...	$n^T_{1,K-1}$	n_1
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
K-1	$n^T_{K-1,0}$	$n^T_{K-1,1}$	...	$n^T_{K-1,K-1}$	n_{K-1}
$\sum$	n^T_0	n^T_1	...	n^T_{K-1}	n

Table 1: Confusion matrix for a decision tree.

The confusion matrix gives information about what classes are being misclassified, which may be useful to group several classes in a single superclass that could be refined later.

2.6 Pruning

Pruning was originally developed as part of CART by Breiman et al. (1984), and it is also known as the BFOS algorithm. In a later work, Chou et al. (1989) extended the original pruning algorithm in order to cover a variety of applications in source coding and modeling in which trees are involved, such as tree structured vector quantization, variable order Markov modeling and optimum bit allocation, for example. Other pruning methods may be found in (Mingers, 1989a). All of these methods are node-based, that is, decisions are taken depending on the values computed at each node. A new pruning algorithm which is edge-based may be found in (Pereira and Singer, 1999).

The main idea in pruning is finding a subtree of T that minimizes the misclassification error for a second data set called the *corpus set* or the *test set*, C_m, that is,

$$\hat{L}_{n,m}(T) = \frac{1}{m}\sum_{i=1}^{m} 1_{\{T(X'_i)\neq Y'_i\}} \tag{2.4}$$

where $C_m = \{(X'_i, Y'_i)\}$, $i = 1, \ldots, m$. $\hat{L}_{n,m}$ is called the corpus set resubstitution estimate.

Measuring misclassification error using a second data set that has not been used during the training stage yields an estimator that is clearly unbiased, in the sense that

$$\mathbf{E}\{\hat{L}_{n,m} \,|\, D_n\} = L_n.$$

Using a result from (Devroye et al., 1996, p. 122) based on Hoeffding's inequality, we can bound L_n using a second data set of m elements as follows: for every $\epsilon > 0$,

$$\mathbf{P}\{|\hat{L}_{n,m} - L_n| > \epsilon \,|\, D_n\} \leq 2e^{-2m\epsilon^2}. \tag{2.5}$$

For example, if we want our misclassification error estimation to be at most 0.01 far from the real L_n with probability 0.95, we need at least

$$m \geq \frac{\log((1-0.95)/2)}{-2\cdot 0.01^2} = 18444.397$$

vectors in the corpus set. Notice that the term ϵ^2 makes m grow really fast, and that this bound does not use how good our classifier is. A better bound may be obtained using Bernstein's inequality (Devroye et al., 1996, p. 124). Let

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(1_{\{T(X_i')\neq Y_i'\}} - \hat{L}_{n,m})^2$$

be the observed variance of our error estimation. Then, for every $\epsilon > 0$,

$$\mathbf{P}\{|\hat{L}_{n,m} - L_n| > \epsilon \,|\, D_n\} \leq 2\exp\left(\frac{-m\epsilon^2}{2\sigma^2 + 2c\epsilon/3}\right), \tag{2.6}$$

where $c = 1 - \hat{L}_{n,m}$. Suppose $\hat{L}_{n,m} = 0.148333$ and $\sigma^2 = 0.12633$, for example. Then, repeating previous example, we need at least

$$m \geq (2\sigma^2 + 2 \cdot 0.851667 \cdot 0.01/3)\frac{\log((1-0.95)/2)}{-0.01^2} = 9529.38$$

vectors in the corpus set, which is sensibly lower than Hoeffding's inequality. Bernstein's inequality is useful when ϵ is larger than about $\max(\sigma/\sqrt{m}, c/\sqrt{m})$, and it is typically better than Hoeffding's inequality when $\sigma \ll c$.

Eqs. (2.5) and (2.6) may be used to obtain a confidence of the error estimation. We will use them in Sect. 4.1.2 where a large hyperspectral image with thousands of training vectors is available.

2.6.1 Pruning algorithm

Basically, the BFOS algorithm finds a *nested* sequence of pruned subtrees which are optimal in a R/D sense, where usually R is measured through the number of leaves and D is the misclassification error. The key issue in the BFOS algorithm is the word nested, because the number of possible subtrees of a given tree grows exponentially with the size of such tree. Fig. 5 shows a typical set of rate/distortion pairs. Each plus sign + is a possible tree that has been built during the training stage (of several experiments), while each training process generates a path starting from a zero rate condition until a zero distortion condition is achieved. A possible training process is also depicted.

The BFOS algorithm finds all points (that is, all possible subtrees) that are in the lower contour of the convex hull, that is, each point in the convex hull is the tree yielding the smallest distortion for a given rate. We could also define the upper contour of the convex hull, but it would be useful for our purposes.

Definition 2.6 *A subtree S is called a pruned subtree of T, or $S \preceq T$, if the root of S is the root of T. Thus, $\tilde{S} = \tilde{T} \Leftrightarrow S = T$.*

Fig. 6 shows all the possible pruned subtrees of a tree T, which form a nested sequence $S_3 \preceq S_2 \preceq S_1 \preceq T$. It is easy to see that the number of pruned subtrees also grows exponentially with

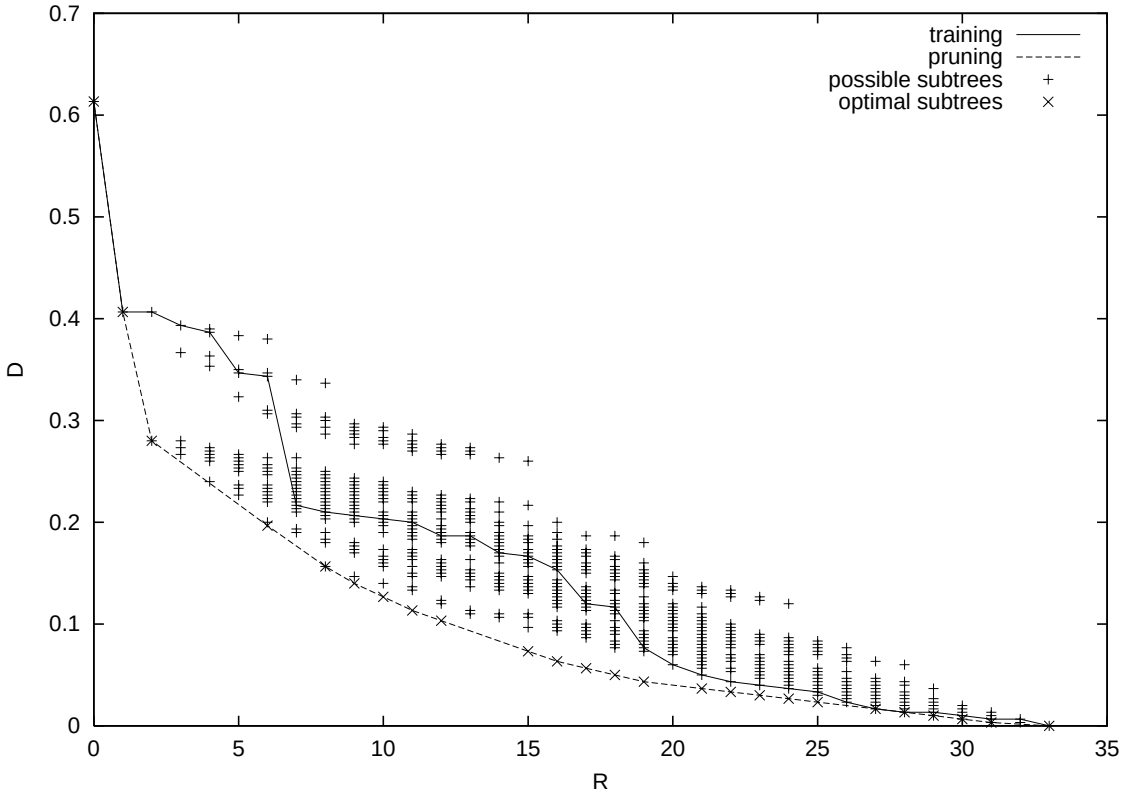

Figure 5: R/D pairs generated by all possible subtrees and the convex hull generated by the pruning algorithm.

the depth of the tree in a recursive fashion. A complete tree of maximum depth $\overline{d}$ (with $2^{\overline{d}}$ leaves) has $S(\overline{d})$ possible subtrees, where

$$S(\overline{d}) = \begin{cases} 1 & \text{if } \overline{d} = 0 \\ 1 + S(\overline{d}-1)^2 & \text{if } \overline{d} > 0. \end{cases}$$

For example, a decision tree with minimum depth 4 and maximum depth 5, which is really a small tree, has between 677 and 458330 possible pruned subtrees. Actually, the exact number of possible pruned subtrees $S(T)$ may be recursively computed as follows:

$$S(T) = \begin{cases} 1 & \text{if } T \text{ is a single leaf} \\ 1 + S(T_l)S(T_r) & \text{if } T \text{ is a tree with two branches } T_l \text{ and } T_r. \end{cases}$$

Therefore, to compute all the possible subtrees of T and then choose those that minimize misclassification error for the corpus set is computationally very expensive. BFOS algorithm is a clever way to handle this problem in order that only a nested sequence of pruned subtrees is evaluated.

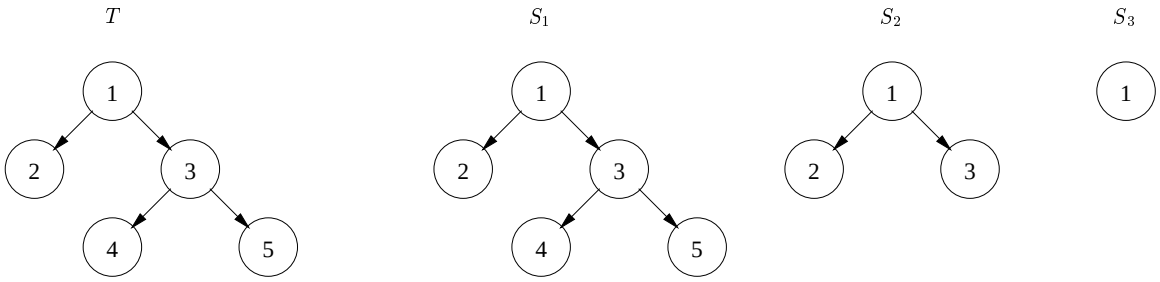

Figure 6: Pruned subtrees of a tree.

The idea behind pruning (Breiman et al., 1984, p. 66) is to define a new measure on a tree called its *complexity*, which is a representation of the size of the tree.

Pruning works as follows. Each node in the tree is the starting point for a subtree which will end with several leaves. Before pruning, the leaves will contain examples belonging to only one class (if a perfect tree has been built), but as pruning progresses, the remaining leaves will be not pure. Such leaves are labeled according to the labeling rule, and the most frequent class is selected. If the subtree is pruned back, then the expected error rate is that of the root of such subtree, which becomes a single leaf. If the subtree is not pruned back, then the error rate is the average of the error rates at the leaves weighted by the number of examples at each leaf. Each pruned subtree increases the error rate in the training set, and dividing this increase by the number of leaves in such subtree gives a measure of the reduction in error per leaf. This is known as the *error-complexity measure*.

The complexity cost measure is the cost of one extra leaf in the tree, denoted by α. Then the total cost of a subtree T_t is

$$\sum_{t_i \in T_t} r(t_i)p(t_i) + \alpha|T_t|$$

and of the node t, if such subtree is pruned is

$$r(t)p(t) + \alpha$$

where $r(t)$ is defined by Eq. (2.1). These are equal when

$$\alpha = \frac{r(t)p(t) - \sum_{t_i \in T_t} r(t_i)p(t_i)}{|T_t| - 1}.$$

Then α gives a measure of the value of the cost-complexity term for the subtree T_t. The pruning algorithm computes α for each subtree (except the first, when $|T| = 1$) and selects the smallest α for pruning. Repeating this process until there are no subtrees left yield a series of nested pruned trees, which form the lower convex hull shown in Fig. 6.

Then, for each subtree T in the convex hull, $\hat{L}_{n,m}(T)$ is measured and the tree yielding the lowest one is selected as the final tree. At the beginning, for small trees $\hat{L}_{n,m}(T)$ is large and it decreases

as the tree size grows, as $\hat{L}_n$ does. But usually there is a point where $\hat{L}_{n,m}(T)$ stops decreasing and starts to increase, while $\hat{L}_n(T)$ always decreases. Such point is considered the start of overfitting point. Fig. 7 shows such behavior for a typical R/D plot.

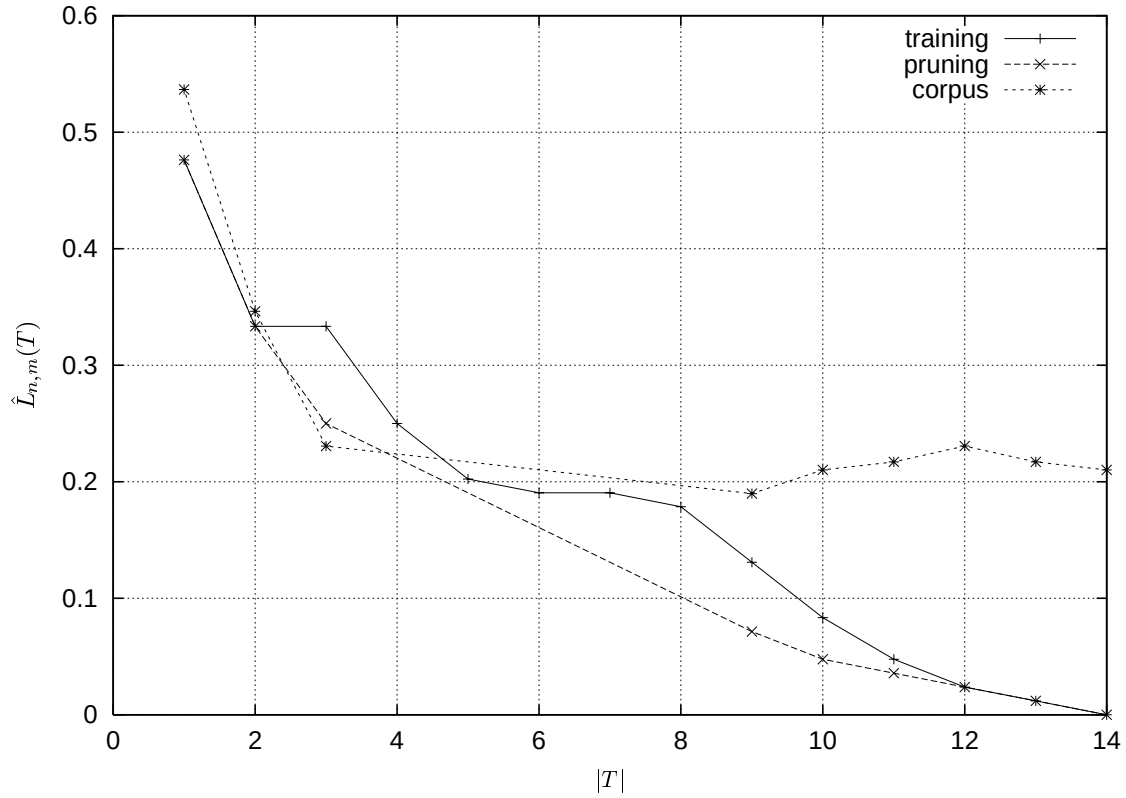

Figure 7: Overfitting occurs when $\hat{L}_{n,m}(T)$ stops decreasing and it begins to grow.

As stated in (Murthy, 1997, p. 21), pruning is more beneficial with increasing skewness in class distribution and/or increasing sample size. Other pruning algorithms are described in (Mingers, 1989b). Another technique known as *averaging* is described in (Oliver and Hand, 1995).

2.7 Practical criteria

In order to build good decision trees, there are several aspects that have to be taken into account:

1. As important as obtaining a misclassification error as low as possible, is that this misclassification error is a reliable estimation. Therefore, a training set and a corpus set should be

drawn from the original data set, the former will be used for training purposes and the later for evaluation purposes. A rule of thumb (Breiman et al., 1984) is to split the original data set in 2/3 of the cases for the training set and 1/3 of the cases for the corpus set. Eqs. (2.5) and (2.6) may be used to determine the confidence interval of the estimated misclassification error, although they may be useless unless m (the number of vectors in the corpus set) is high. N-fold cross-validation is a good choice for computing an estimation of misclassification error.

2. The splitting criterion is an important issue because tree performance directly lies on the quality of the first splits. A bad split in the first levels may cause a poor generalization error. In Sect. 4.2 we test several splitting criteria in order to rank them according to their performance on several standard classification data sets. Both Gini and the R-norm criteria seem to yield the best results. In our honest opinion, the R-norm criterion should be more deeply analyzed as it could be used in an adaptive method which may take advantage of extra information such as node position and intrinsic characteristics.

3. Sometimes a leaf too mixed is split using a variable which is not really representative, generating two new leaves that do not contribute to reduce tree impurity and causing also a bad split that may contribute to poor performance for the corpus set. This happens when a random data set is chosen for splitting. An extra classification feature may be used to detect and avoid this problem. Suppose we add a random variable $\mathcal{X} \sim \mathcal{N}(0,1)$ as a new classification feature. Then, when the best possible split has been found, reject to split when $\mathcal{X}$ is part of such split. The intuitive concept is clear: if a random split is better than any other split, the data set is too mixed and it cannot be split. This may be combined with a constraint upon the minimum number of elements in a leaf. When a tree is built following these criteria, those leaves that were rejected for splitting may represent sets of mislabeled data, so they can be used to detect outliers in the training set. Removing outliers may improve classification performance and the understanding of the data set properties.

4. Pruning always finds the best subtree for a given test set, so it should be always carried out in order to find the smallest subtree with the smallest misclassification error. More than any other criterion, pruning is the critical issue when building decision trees (and with any other classification method where a large system is built and then is pruned back, such as neural networks, for example). The right size of a classification system is a key issue in its error generalization performance, as we will study in Sect. 5.1.1.

2.8 Known problems

Even after pruning, resulting decision trees may be too large and, therefore, too much specific. The deeper a split in the decision tree is, the more specific probably is. This may generate unrealistic

leaves which represent particular characteristics from the training and corpus sets, but not a real characteristic of the underlying distribution of (X, Y), causing overfitting.

Furthermore, large decision trees are likely to use all the classification features present in the training set, even if they are not directly related to the class which we are trying to find. For example, suppose we are using decision trees to classify hyperspectral images obtained from an HIRIS or AVIRIS sensor (Goetz and Herring, 1989), with 192 or 224 spectral bands respectively. If a large tree is built, all 192 or 224 bands will be used sooner or later anywhere in the tree, which is clearly a mistake. We would like to be sure that only the best classification features will be selected for splitting.

Another problem related to decision trees is the number of classes present in the training set. When the number of classes is high, decision trees become too specific to training set characteristics, as stated in (Kim and Landgrebe, 1991), because of the poor representation of the intrinsic characteristics of each class when the number of elements is small.

Our goal is, therefore, to build a better classification system based on decision trees that does not suffer from overfitting (or, at least, a system that tries to minimize it), it has a reduced classification cost, it uses a reduced set of classification features and, finally, it performs better when the number of classes is high. Our approach is to break the classification process in a sequence of partial decisions, using a measure of how mixed (or impure) a data set is. We call this ensemble *progressive decision trees*, and it will be discussed in the next chapter.

Chapter 3

Progressive decision trees

Innovation: Toujours dangereuse
Gustave Flaubert, *Le Dictionnaire des Idées Reçues*

As stated in the previous chapter, decision trees suffer from several drawbacks that limit their use in some scenarios: the most important is that generated trees may be too large and, therefore, too specific, becoming a classification system with a bad generalization error. Furthermore, classification cost may be extremely high when large data sets are used during the training stage. Trees also become unstable when the number of classes is high and the size of the training set is limited. We are trying to build a classification system that would (ideally) allow us:

1. To improve generalization error reducing overfitting.
2. To reduce training and pruning algorithms costs.
3. To treat elements from several classes as elements of a single superclass which would be solved by another classifier later.

In this chapter we present a novel ensemble of decision trees which allow us to partially fulfill the requirements stated above. We call this ensemble *progressive decision trees*.

It is important to specify what we mean by progressive: classification is no longer a one-stage process but a multi-stage one. Instead of having a single tree which classifies the whole input space, we break the classification problem in a sequence of partial classification sub-problems which can be independently solved and then combined into a more robust classifier. Each tree is a partial classification system which acts in a lazy scenario: it only classifies those samples which are easily classified by a few splits. We prefer the term *partial* rather than *lazy* because the latter was already used in (Friedman, Kohavi and Yun, 1996) with a different meaning. The term progressive was also used because our initial work was related to document layout recognition (Minguillón et al., 1999), where an input image was classified at a coarse level, and then progressively refined.

As stated in (Liu and Setiono, 1998), when orthogonal hyperplanes are used to build univariate decision trees, three problems arise: *replication*, *repetition* and *fragmentation*. These problems were defined in (Pagallo and Haussler, 1990). Replication occurs when similar subtrees are replicated in a decision tree, yielding to an inefficient partition of the input space. Repetition occurs when classification features are repeatedly tested more than once along a path in a decision tree. Fragmentation occurs when data are gradually partitioned into small fragments (Brodley and Utgoff, 1992). Replication and repetition always imply fragmentation, but fragmentation may also occur if too many features need to be tested. The fragmentation problem is also related to poor generalization error performance (see Sect. 5.1.1), as the number of smaller leaves which are created during the growing stage increases yielding to a poor margin distribution (see Sect. 3.8.2). In the following sections we will describe how progressive decision trees deal with such problems. All the experiments and theoretical formulations related to our ensemble are presented in the next two chapters, in order to facilitate the reading of this chapter.

3.1 Progressive decision trees

In this section we define the basic framework for our classification system. As we use CART (Breiman et al., 1984) as the basic tool for constructing progressive decision trees, we will use the notation presented in the previous chapter. In Chapter 5 we will give a more formal framework for progressive decision trees trying to find relationships between our classification system and other classification systems that have been recently developed.

3.1.1 Labeling rule

We introduce an extra class, called the *mixed* class, which will be used as a class representing a mixture of elements of several classes. For the moment, we will denote this class as M. The new labeling rule is then

$$l'(t) = \begin{cases} l(t) & \text{if } r(t) \leq \epsilon \\ M & \text{otherwise.} \end{cases} \tag{3.1}$$

where $l(t)$ and $r(t)$ are defined in Eq. (2.1) and ϵ is the desired threshold for considering a label too impure in a misclassification cost sense. The basic idea is that $l(t)$ is rejected if $r(t)$ fails to be under a certain threshold ϵ. The concept of *rejection* is also described in (Bottou and Vapnik, 1992) where classifiers are studied under the best compromise between locality and capacity. It is fully described in (Baram, 1998b)[1], under a partial classification framework. The concept of rejection is also related to the concept of *specialist* described in (Blum, 1997) and further studied in (Freund, Schapire, Singer and Warmuth, 1997), where each specialist (that is, a base-level classifier) decides whether to give an opinion or not depending on the context. A similar approach is also used in

[1] A brief paper containing corrections to this one may be found in (Baram, 1998a).

(Gavin, Puzenat and Zighed, 1999), where misclassification error is redefined to take into account only classified samples.

3.1.2 Growing and pruning basics

The growing algorithm is basically the same as the one described in Sect. 2.3 for classical decision trees. Every time a leaf t is split, a new subtree is created. This subtree is trained using only those vectors in D_n that were in t, that is, in the region of the input space which t represents. This training does not depend on other vectors that were in other regions, so it is easy to see that, for each leaf t that is going to be split, a new thread might be created, sharing the root of the tree T in order to compute global tree measures such as misclassification error or average rate, for example. This process is stopped when a certain stopping condition is reached (it will be discussed later in Sect. 3.4).

On the other hand, when such tree is pruned back using the pruning algorithm described in Sect. 2.6, and several leaves are classified as mixed because they are too impure, what we do is to join all these leaves in a single data set which represents the whole input space but a finite number of holes (which may be finite or infinite, though). This new leaf is labeled as mixed, so the process is repeated recursively, as a second pruning stage.

The whole process is started again joining all leaves labeled as mixed in a new training set, so a new decision tree is built. A two-dimensional example is shown in Fig. 8, where a simple tree T_1 with two regions labeled as mixed only classifies the right uppermost corner of the input space.

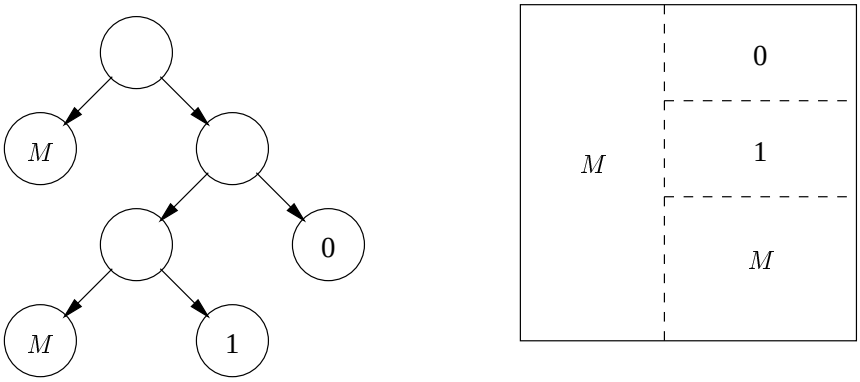

Figure 8: Mixed regions of a progressive decision tree.

Then, the two regions labeled as mixed generate a decomposition of the original input distribution (X, Y), as follows,

$$(X, Y) \doteq (X, Y)^{(\overline{0})} = (X, Y)^{(1)} \oplus (X, Y)^{(\overline{1})}$$

where $(X,Y)^{(1)}$ is the distribution of those samples falling in the regions classified by T_1, and $(X,Y)^{(\overline{1})}$ is the distribution of those samples not classified by T_1. Analogously, we can decompose the original training set as follows,

$$D_n \doteq D^{(\overline{0})}_{n_{\overline{0}}} = D^{(1)}_{n_1} \oplus D^{(\overline{1})}_{n_{\overline{1}}}, \quad n = n_{\overline{0}} = n_1 + n_{\overline{1}},$$

that is, the training set is decomposed in two sets. Then, a new tree T_2 may be trained using $D^{(\overline{1})}_{n_{\overline{1}}}$ generating a new partition, as shown in Fig. 9. In general, at stage $i+1$, T_{i+1} decomposes the training set in two new sets, as follows,

$$D^{(\overline{i})}_{n_{\overline{i}}} = D^{(i+1)}_{n_{i+1}} \oplus D^{(\overline{i+1})}_{n_{\overline{i+1}}}, \quad n_{\overline{i}} = n_{i+1} + n_{\overline{i+1}}. \tag{3.2}$$

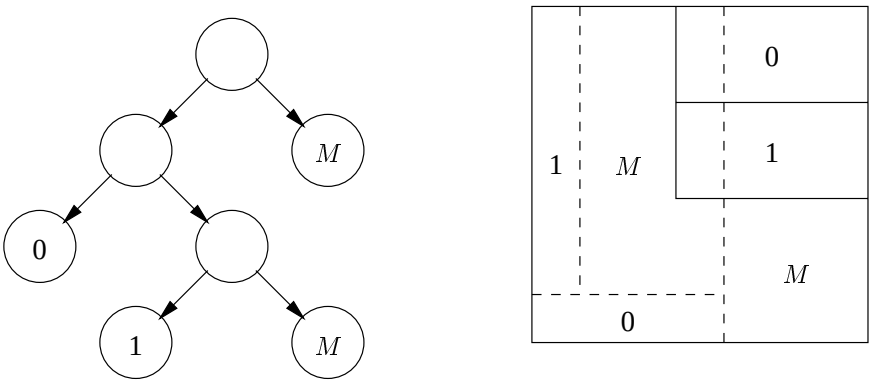

Figure 9: Partition refinement.

Initially, the whole input space (the original training set D_n or $D^{(\overline{0})}_{n_{\overline{0}}}$) may be classified using a trivial tree T_0 with a single leaf labeled as mixed. A first tree makes a refinement of such input space, classifying a percentage of the input space with a low misclassification error (because leaves too impure are labeled as mixed and, therefore, not classified). Regions of the input space left unclassified form the new training set $D^{(\overline{1})}_{n_{\overline{1}}}$, so this process may be repeated until classification requirements (in both classification percentage and accuracy) are satisfied, or until it is not possible to do further refinements. We will discuss both cases later in Sect. 3.4.

Intuitively, this refinement process may be seen as a simple but elegant way to handle boundary problems: the best splits (those in the top of the tree) separate the easy cases from the input data set, and (hopefully) before the splits become too specific, all data sets labeled as mixed are joined together and a new tree is created, trying to separate the new easy cases. We avoid the use of too deep trees that may cause overfitting, but maintaining a good classification performance (Holte, 1993; Auer, Holte and Maass, 1995) at a reduced cost (in both training and classification

senses). As the size of the consecutive training sets $n_{\overline{i}}$ decreases at each stage, both training and classification cost also decrease.

When ϵ is small, most leaves are labeled as mixed and then joined, pruning back the decision tree and creating a small decision tree. If ϵ is too small, a degenerated decision tree with a single leaf labeled as mixed would be created, unless pure regions would be present in the training set. Such a trivial tree does nothing, so it is useless and its use should be avoided. The final decision tree needs at least three leaves, in order to be a non trivial tree or the first level of a classical decision tree.

On the other hand, when ϵ is large enough (but not too much), large trees may be obtained even after pruning and joining leaves labeled as mixed. In this case, the decomposition described by Eq. (3.2) may be seen as a naive classification of the samples in two sets: the easy samples and the hard samples (in terms of classification cost and accuracy). The underlying idea is that easy samples are classified with a high confidence while hard samples are left unclassified. Eq. (3.2) may be refined again as $D_{n_{\overline{1}}}^{(\overline{1})}$ is the new training set for a new classifier.

3.1.3 Confusion matrix

The confusion matrix for progressive decision trees has an extra column with respect to the confusion matrix defined in Sect. 2.5.1, as we need to count how many elements are not classified (that is, classified as mixed):

$j\backslash k$	0	1	...	$K-1$	M	$\sum$
0	$n_{0,0}^T$	$n_{0,1}^T$	...	$n_{0,K-1}^T$	$n_{0,M}^T$	n_0
1	$n_{1,0}^T$	$n_{1,1}^T$	...	$n_{1,K-1}^T$	$n_{1,M}^T$	n_1
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
K-1	$n_{K-1,0}^T$	$n_{K-1,1}^T$	...	$n_{K-1,K-1}^T$	$n_{K-1,M}^T$	n_{K-1}
$\sum$	n_0^T	n_1^T	...	n_{K-1}^T	n_M^T	n

Table 2: Confusion matrix for a progressive decision tree.

We can define the percentage of input vectors classified by T as

$$P_T = \frac{\sum_{k=0}^{K-1} n_k^T}{n} = 1 - \frac{n_M^T}{n}.$$

Although Tab. 2 is related to the training set, an equivalent confusion matrix may be computed for the corpus set.

3.2 Decision graphs

As two or more leaves are joined to create a new leaf which is recursively split, decision trees become decision acyclic graphs (DAG) (Oliver, 1993). As stated in (Pagallo and Haussler, 1990), *conjunctions* (regions defined by intersections) can be described efficiently by decision trees, while *disjunctions* (regions defined by unions) require a large tree to be described. This is directly related to the replication problem, as the same subtree describing a conjunction needs to be built for each disjunction. We will use the same example in (Oliver, 1993) to illustrate this problem: suppose we are dealing with four boolean attributes (A, B, C and D) and a goal function f which can be described as

$$f = (A \wedge B) \vee (C \wedge D).$$

A possible perfect decision tree for such data set would use A as the classification feature for the first split (actually it could have selected any because of problem inherent symmetry), while the second splits (one for $\overline{A}$ and another for A) would use C and B respectively, as shown in Fig. 10 (D could also have been used instead of C). Notice that the subtree describing the term $(C \wedge D)$ requires two identical subtrees, which is inefficient, causing the replication problem.

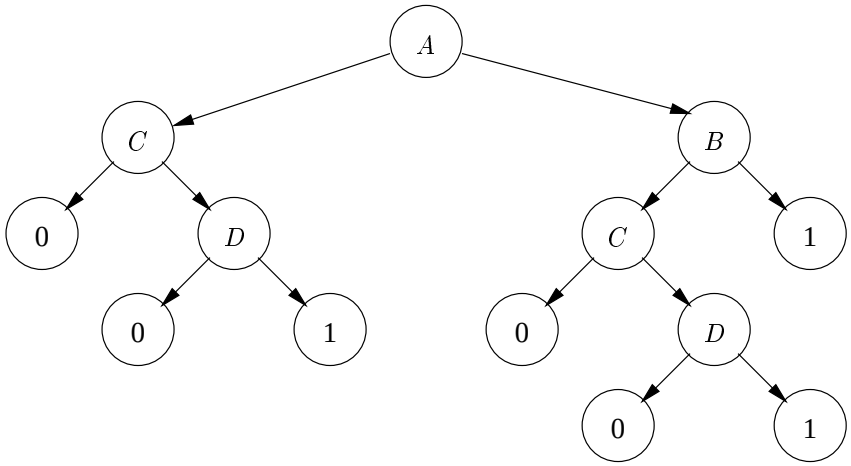

Figure 10: A decision tree for $(A \wedge B) \vee (C \wedge D)$.

Regarding progressive decision trees, suppose we stop the growing algorithm when there is a leaf (or maybe two) labeled different than mixed. Then we join all mixed leaves (except in the trivial case) and then we repeat the process until a perfect decision tree is found. Suppose also that we set $\epsilon = 0$, so all leaves will be labeled as mixed unless they are pure.

Suppose A is also chosen the first classification feature. When the initial data set is split using A, both leaves would be labeled as mixed. Suppose now the right leaf is split using B, generating

two new leaves, but in this case the right leaf is not labeled as mixed, but to 1 (as it only contains elements from class 1). Then, the other two leaves (representing $\overline{A}$ and $A \wedge \overline{B}$) are joined together and the process is repeated again. In this case C (but also D by symmetry) is chosen as the new classification feature, creating a perfect leaf and another leaf which, finally, is split using D as the classification feature, and the process stops here because all input data is correctly classified. The final decision tree (a decision graph, actually) is shown in Fig. 11, and it is also the minimal tree (or graph) which represents f.

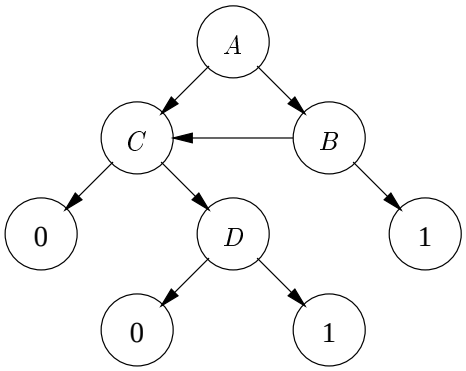

Figure 11: A progressive decision tree (DAG) for $(A \wedge B) \vee (C \wedge D)$.

Although this example is far from a real scenario, it shows the possibilities of progressive decision trees. Notice that, in order to achieve the desired decision tree shown in Fig. 10, B must be chosen as the classification feature to split the right leaf obtained in the first split where A was used. If not, the replication problem would not be avoided. As we do early stopping, the node selection criterion becomes critical. Notice also that, in this case, B is the split that produces the most perfect possible subsets, though.

Progressive decision trees deal with the replication problem trying to join those regions that should not be treated separately. Joining also fights against the fragmentation problem, as small regions are joined together in single large region.

3.3 Joining mixed leaves

Suppose we grow a complete decision tree of maximum depth $\overline{d}$ and then we prune it back, name it T. If $\overline{d}$ is small compared to the real decision tree average rate R, probably there will be a lot of leaves in T labeled as mixed. We can join all these leaves together and then start a new training process, following the process previously described.

This process has a drawback, though. When $\overline{d}$ is not too small, the partition induced by T may

possibly generate unconnected regions labeled as mixed that should not be joined together, as shown in Fig. 12. This problem is also related to other problems regarding inefficient splits, as discussed in Sect. 3.6.2.

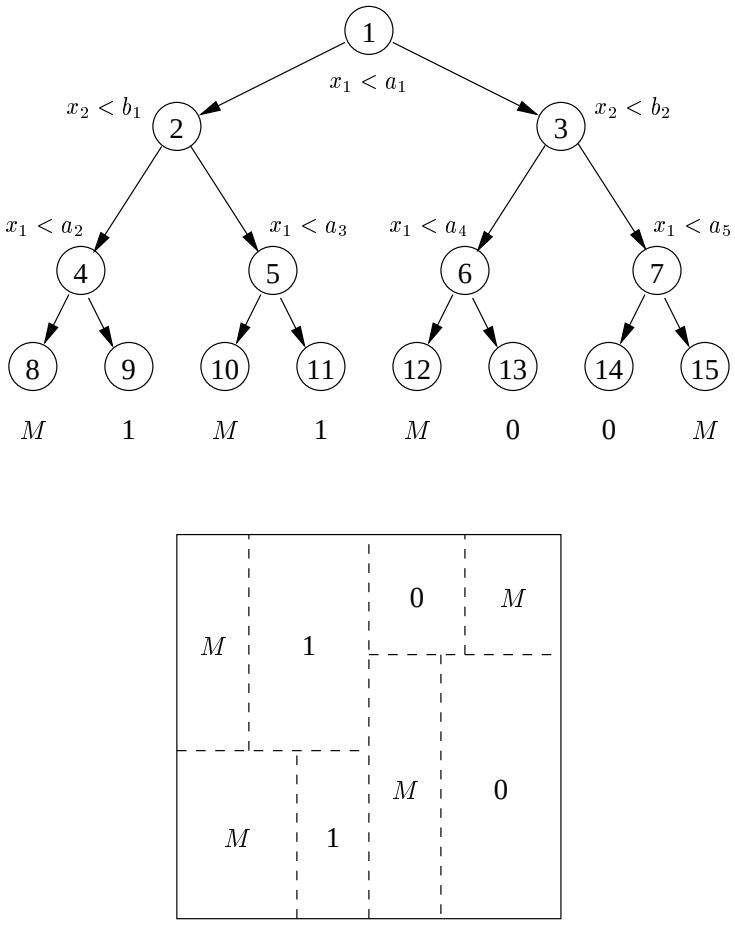

Figure 12: Joining may involve unconnected regions labeled as mixed.

Although joining all mixed leaves together is a good approach to avoid both replication and fragmentation problems, our experiments show that often the first splits in the second tree (which is trained with all data in the regions labeled as mixed) reproduce approximately the same splits that the first tree, trying to separate again the mixed regions. This may happen when either $\overline{d}$ or ϵ are not small. Therefore, the first splits of each progressive decision tree should be inspected trying to detect coincident splits which are superfluous.

3.3.1 Detecting connected regions

Once the decision tree has been built, it is easy to detect connected and unconnected regions labeled as mixed. Suppose by now that splits are orthogonal. For each leaf labeled as mixed, generate a list containing all representative splits, that is, only those splits that are boundaries of such region. For example, in Fig. 12:

- Leaf 8: $x_1 < a_2$, $x_2 < b_1$ (split $x_1 < a_1$ is superseded by $x_1 < a_2$)
- Leaf 10: $x_1 < a_3$, $x_2 \geq b_1$
- Leaf 12: $x_1 \geq a_1$, $x_1 < a_4$, $x_2 < b_2$
- Leaf 15: $x_1 \geq a_5$, $x_2 \geq b_2$

Then, for each pair of leaves containing at least one common boundary (that is, the same split), try to find any non empty intersection. For example, leaves 8 and 10 share the same split $x_2 < b_1$, and the intersection of $x_1 < a_2$ and $x_1 < a_3$ is obviously non empty. On the contrary, leaves 12 and 15 share the same split $x_2 < b_2$ but the intersection of $x_1 \geq a_1$, $x_1 < a_4$ and $x_1 \geq a_5$ (provided $a_5 > a_4$) is empty. This process is easily extendible to d dimensions in a recursive fashion, although it may be computationally very expensive for large trees and large d.

For non orthogonal splits, the problem is how to find non empty intersections efficiently. Furthermore, two regions not sharing any split may be connected, which is impossible for the orthogonal splits case. As we will see in Sect. 3.6, this may also be detected during the process of combining decision trees.

3.4 Growing and pruning parameters

For the moment we have only the intuitive idea behind progressive decision trees. Nevertheless, the framework defined in the previous sections arises several questions related to the parameters which are inherent to the training process (both growing and pruning):

1. How deep a single tree has to be?
2. How to set ϵ?
3. Is pruning necessary?
4. How many trees (denoted by t) are needed?
5. Which leaves must be joined?

In the following sections we will try to give several answers to these questions using both empirical and theoretical criteria. These answers would determine the growing and pruning processes for progressive decision trees, the subject of study in this thesis. It is easy to see that these questions are related one to each other, so their answers will be related too, showing that probably there is an underlying more general framework that we have to establish.

3.4.1 Minimal progressive decision trees

Suppose now we set $\epsilon = 0$ (although this is not really important now) and that our stopping rule is "stop the training process when at least there is one leaf not labeled as mixed", that is, when a leaf contains only elements from one class. Suppose, by simplicity, that the last split (which caused a pure leaf to be found) does not create two pure leaves, but only one. Suppose also that we force that at each step of the training stage, a new classification feature is used.

It is easy to see that because all regions will be labeled as mixed but one, these regions are all connected, so we have a tree that separates one region labeled as mixed from another not labeled as mixed. The path from the root of the tree to the leaf not labeled as mixed defines the boundaries of such leaf, and therefore, a very simple tree with only one decision node and two leaves may be built, using a special type of decision node, which is no longer a hyperplane but a hyperrectangle (notice that it may be finite or infinite, though). Of course, we assume that only orthogonal hyperplanes are used for node splitting, although the method could be also generalized to use general hyperplanes.

Hyperrectangles in $\mathbb{R}^d$ are an interesting subject of study because they approximate the distribution of d independent multivalued random variables, as stated in (Auer, Long and Srinivasan, 1997). Our method is greedy but only computes one side of the hyperrectangle at each iteration. Another greedy method which computes the best possible hyperrectangle at each iteration is described in (Devroye et al., 1996, p. 348). This method is proved to be *consistent*[2], although the authors remark that it serves only as a prototype and should not be considered under a practical approach. The consistency of an unsupervised greedy growing algorithm that produces tree structured classifiers from labeled training data is described in (Nobel, 1994). The *patient rule induction method* algorithm (PRIM) (see (Hastie et al., 2001, p. 279)) also finds hyperrectangles in the feature space.

Therefore, progressive decision trees may be seen as an easy (and maybe naive) approximation to the problem of complex boundaries construction using hyperrectangles, but under a suboptimal approach, trying to find a compromise between its computational complexity and its performance. In Sect. 5.2.4 we will describe under a more theoretical approach the use of progressive decision trees as a single decision tree built using hyperrectangles.

[2] see (Devroye et al., 1996, p. 91) for a definition of consistent.

3.4.2 General case

But using hyperrectangles to create holes which are perfect is not a good way to build decision trees, as these holes are probably too specific to the training set, causing overfitting. The five questions should be treated as a whole, trying to find relationships between them, using also previous knowledge and intuition:

- The basic idea of partial classification is based on a fast, first opinion with a large margin of confidence, and if this opinion does not satisfy a certain criterion, a second opinion is searched and so on. Therefore, the first trees in such ensemble should be small, trying to solve the easiest cases only with a high confidence. The last tree, which has to deal with the hardest samples must be a normal decision tree, as largest as needed.

- Following the same approach, ϵ should be very small for the first trees and grow accordingly to each tree accuracy. If ϵ is too small, it will be impossible to build small decision trees (unless the classification problem is easy to solve), and if it is too large, the confidence in our prediction will be low. Experiments in Sect. 4.5.2 show ϵ cannot be very different from the expected misclassification error to ensure a reasonable tradeoff between classification accuracy and percentage of classified vectors.

- When trees are small, pruning is not an issue as it has a low computational cost. Furthermore, very small trees are usually not pruned at all. On the other hand, for large trees pruning is the most effective tool for building good decision trees, so it should be used.

- Usually only two or three trees are needed to obtain a good progressive decision tree. Nevertheless, the number of trees depends also on the size of the data set, although increasing the number of decision trees does not ensure increasing classification performance, but the contrary. Gama and Brazdil (2000) also use only two or three levels for cascading as a good tradeoff between classification performance and complexity.

- The simplest way to deal with the problem of joining leaves is to join them all, and then to study the ensemble between the two decision trees. In fact, if the first splits of the second decision tree reproduce some of the first splits in the first decision tree, this does not have a negative impact on classification accuracy, but on classification cost, as similar questions are asked twice or more times for several elements.

In the following chapter we will try to justify all the statements above empirically defining appropiate experiments for selecting each parameter.

3.5 Combining paradigms

The main idea of progressive decision trees is partial classification: a tree no longer classifies all input data but only a percentage, hopefully yielding a lower misclassification error, and at a reduced cost. Then, a set of small progressive decision trees are combined trying to improve classification performance while reducing classification cost. In general, one or more base-level models are combined using a meta-level model to improve classification system performance. As stated in (Todorovski and Džeroski, 2000), there are three combining paradigms for the meta-level model: voting, stacking (Wolpert, 1992) and cascading (Gama and Brazdil, 2000). Both voting and stacking may be considered *multiexpert systems*, while cascading is better studied as a *multistage system* (Alpaydin and Kaynak, 1998; Kaynak and Alpaydin, 2000). A comprehensive compilation of papers on multiple classifier systems may be found in the series (Kittler and Roli, 2000; Kittler and Roli, 2001; Kittler and Roli, 2002). An excellent recent book which covers several combining paradigms is (Hastie et al., 2001).

Suppose decision trees are used as base-level classifiers. In a voting scheme, each decision tree makes a prediction for each input vector, and the prediction receiving the most votes is chosen as the final prediction. Voting may be weighted or not, depending on the meta-level model. In stacking, a learning algorithm is used to induce a meta-level model (a tree of trees) for combining the predictions of the decision trees. Cascading is an iterative process of combining classifiers: at each iteration, the training set is extended with the predictions obtained in the previous iteration. Other authors have also studied the relationship between decision trees and several combining paradigms (Drucker and Cortes, 1996; Quinlan, 1996a; Kearns and Mansour, 1999; Quinlan, 1999; Drucker, 2000). Trees have also been used as meta-level models in (Chan and Stolfo, 1995).

Combining classifiers tries to extract the best from each one: as stated in (Gama and Brazdil, 2000), the combined error rate depends on the error rate of individual classifiers and the correlation among then. We will define classifier correlation later in Sect. 3.8, where it will be considered as an empirical measure of classifier performance, altogether with a bias-variance (Dietterich and Kong, 1995; Kohavi and Wolpert, 1996; Domingos, 2000) decomposition of the misclassification error and a classical Mahalanobis distance measure.

In the following sections we will discuss our ensemble under these combining paradigms, studying which is the most appropriate under the point of view defined by the questions defined in Sect. 3.4 and giving some empirical reasons for their better performance than classical decision trees. The theoretical details of our ensemble of progressive decision trees will be discussed in Chapter 5.

3.5.1 Voting

Voting is the most common method used to combine classifiers. As pointed out by Ali and Pazzani (1996), this strategy is motivated by the Bayesian learning theory which stipulates that in order to

maximize the predictive accuracy, instead of using a single learning model, one should ideally use all models in the hypothesis space.

Voting involves two different (but related) processes: first, a set of different classifiers must be constructed using the available data set, and second, the output of these classifiers must be combined into a single classification system. As several classifiers working in parallel are combined into a single one, voting is considered to be a multiexpert approach to classifier combination. Therefore, a voting classification system is completely defined by:

1. The way the base-level classifiers are generated from the training set. As the training set is unique and the learning algorithms are deterministic, this means that different training sets must be generated from the input training set.

2. The way the base-level classifiers are combined into a single classification system. It may be a simple majority voting scheme or a more complex weighted voting scheme depending on each base-level classifier performance, for example.

Different choices for the questions described above generate different voting methods. The most well known voting methods are bagging and boosting, which will be briefly described in the following sections.

3.5.1.1 Bagging

Bagging was introduced by Breiman (1996a), and it is based on the manipulation of the input training set using subsampling with replacement, so each new training set has the same number of examples but some of them may be missing respect from the original training set and others may appear several times. Bagging receives its name from the procedure of "bootstrap aggregating", which consists in creating the replicates of the training set using subsampling with replacement. This procedure is fully described in (Efron and Tibshirani, 1993).

For each training set, a classifier is generated, and all classifiers are combined in a single classification system, usually using a uniform voting scheme. Bagging is, in fact, the most simple way to build a multiple classifier system, and, despite of its simplicity, it may achieve very good results when is used with unstable base-level classifiers, such as decision trees, for example. For unstable we mean that the learning algorithm used to build the classifier is very sensitive to small perturbations in the training set.

According to this definition, our ensemble cannot be studied under the bagging framework, because the training sets are not built using subsampling but depending of the output of the previous classifier, and the voting scheme of our ensemble is clearly not uniform.

3.5.1.2 Boosting

Freund and Schapire developed boosting as a new tool for creating ensembles of classifiers, see (Freund, 1995; Quinlan, 1996a; Freund and Schapire, 1999) for boosting basics. Boosting refers to a general method of producing a very accurate prediction classifier by combining less accurate classifiers. The main idea is that each classifier which satisfies being a *weak learner* —that is, its accuracy is only slightly better than chance— may be boosted when combined altogether with many other weak learners.

Basically, boosting uses a set of weights for the training set, and then a classifier is built taking into account this set. Initially, all weights are equal. The input vectors are classified and then all vectors wrongly classified are boosted in order to have more weight, creating a new training set (actually, creating a new set of weights). In the final voting scheme, each classifier is weighted according to its performance. This process (generating a new set of weights, computing a new combination scheme) is repeated until the boosting ensemble test error is sufficiently low enough or it shows an asymptotic behavior. Surprisingly, boosting does not increase test error even for a large number of classifiers, contradicting Occam's razor. A good explanation of why boosting does improve error generalization performance can be found in (Schapire et al., 1998).

Our method may be seen as a basic binary weighted boosting method. Initially, all vectors in the training set D_n are given the same weight. Then, all vectors that were classified (with a misclassification error lower than the given threshold) are no longer considered, while the rest are used in a second training process. A weight 0 is assigned to the former, and a weight 1 to the latter. The main difference with boosting is the way partial decision trees are combined, though, as voting is not weighted based upon classifier performance but order of appearance.

We propose two variations of our ensemble: sequential, which comes up from the definition of progressive classification trees, and parallel/sequential, which tries to solve the multiclass classification problem and to improve classification accuracy, but increasing classification cost. These ensembles will be described in the next section, Sect. 3.6, altogether with their advantages and drawbacks.

3.5.1.3 Voting progressive decision trees

Actually, we are not trying to combine classifiers using a voting scheme because our classification system weights classifiers depending on its order of appearance, and not on its performance. Furthermore, training sets are also created according to the output of the previous decision tree. Nevertheless, progressive decision trees may be also studied under a voting approach if a valid voting weights scheme is formulated. In Sect. 5.3.1 we present such a valid weight scheme, although it is not really useful for practical purposes.

Both bagging and boosting reduce generalization error mainly due to the reduction in the variance (see Sect. 3.8), and only when the learning algorithm is unstable (which is the case for decision trees). As stated in (Bauer and Kohavi, 1999), although boosting is on average better than bagging, it is

not uniformly better than bagging. Boosting tends to reduce bias while bagging tends to reduce variance. A comparison of boosting, bagging and another method based on randomization can be found in (Dietterich, 2000), and another method based also on randomization can be found in (Ho, 1998). A recent paper on ensemble pruning may be found in (Prodromidis and Stolfo, 2001).

3.5.2 Stacking

Generalized stacking, described by Wolpert (1992), is based in the concatenation of several classifiers in two levels. The basic idea is that several classifiers are trained using the original training set (it does not matter how they are trained, but only the fact that several classifiers are available); they form the set of base-level classifiers. Then, a meta-level classifier is used to determine how the outputs of the base-level classifiers should be combined, trying to extract information about the patterns present in the mistakes produced by the base-level classifiers.

Trees have been previously used in a stacking paradigm by Chan and Stolfo (1995), where several classifiers were combined using decision trees following two strategies: *arbiters* and *combiners*. An arbiter (together with an arbitration rule) decides a final classification outcome based upon the base level predictions. In a combiner, the predictions of the learned base level classifiers on the training set form the basis of the meta level learner training set. Trees become a good tool for stacking because of their inherent hierarchical structure. A recent paper on arbiters and combiners may be found in (Ortega, Koppel and Argamon, 2001).

Our ensemble may be described using a very simple (but forced) stacking framework as follows: the first base-level classifier is trained with the whole training set. The second base-level classifier is trained with all the vectors left unclassified by the first one, and so on, that is, each base-level classifier is trained with the vectors left unclassified by the previous one. The meta-level classifier does not need training at all (so it cannot be considered a true stacking ensemble): the output for an input vector is the first output different than mixed in the ordered sequence of base-level classifiers.

3.5.3 Cascading

Cascading, which is fully described in (Gama and Brazdil, 2000), is also based in the concatenation of several classifiers, but using a different approach. In this case, all useful information that can be collected from the output of a given classifier is joined altogether with the input training set (that is, adding new classification features) creating a new training set which can be used to train a new classifier, and so on.

Cascading is completely different to any voting or stacking ensemble, which are multiexpert systems. These use all collected information from the base-level classifiers in the meta-level classifier, and then a final output is generated. On the other hand, there is not such a framework in cascading, which is a multistage system: a classifier tries to classify the input vectors, computing additional information which may be useful for the next classifier in the sequence. The final classifier uses all

information that has been computed during the previous classification stages and then the final class is generated. We will describe the details of this ensemble with more detail in Sect. 3.7.

It is easy to see that our ensemble is a particular case of cascading, as we share the same basic idea: a classifier tries only to classify those vectors which seem easy to classify, and another classifier will do the rest. The main differences are basically two: first, we do not delay the generation of the class of an input vector until the last decision tree, but we allow any tree to classify an input vector when it is confident enough. Second, no additional information is carried over from one stage to the next.

3.6 Cascading progressive decision trees

In this section we describe two ways of generating each classifier taking part in a cascading ensemble. In the *sequential ensemble*, one decision tree is built at each classification stage, while in the *parallel/sequential ensemble* two or more trees are built. By the moment, we do not consider including additional information. In Sect. 3.7 we will analyze progressive decision trees under the approach described in (Gama and Brazdil, 2000).

3.6.1 Sequential ensemble

The simplest way to combine two progressive decision trees is shown in Fig. 13, where two trees named T_1 and T_2 are joined replacing each leaf of T_1 labeled as mixed by a copy of T_2.

This is, of course, the most natural and intuitive way to see progressive decision trees. A first tree tries to classify the input data set. All vectors left unclassified by the first tree are sent to the second tree, and so on. The whole tree could be seen as a network or as a directed acyclic graph, although it is easier to study it as a tree. In other words, a first tree generates a space partition and then, all regions labeled as mixed are joined together and create a new input space. It is better to say, though, that the input space is the same (actually $\mathbb{R}^d$) but the underlying probability of (X, Y) has changed. Therefore, a new tree may be built for this new training set taking advantage of the new properties of such training set.

Although partial decision trees may be small (as small as T_1 and T_2 were in the previous example, for instance), the resulting tree can be really large. If the number of leaves labeled as mixed in T_1 is m_1, the resulting tree has $|T_1| - m_1 + m_1|T_2|$ nodes with a total of $|\tilde{T}_1| - m_1 + m_1|\tilde{T}_2|$ leaves. When several trees are combined, this number may grow exponentially. Therefore, any tree inspection that could reduce resulting tree size should be taken into account.

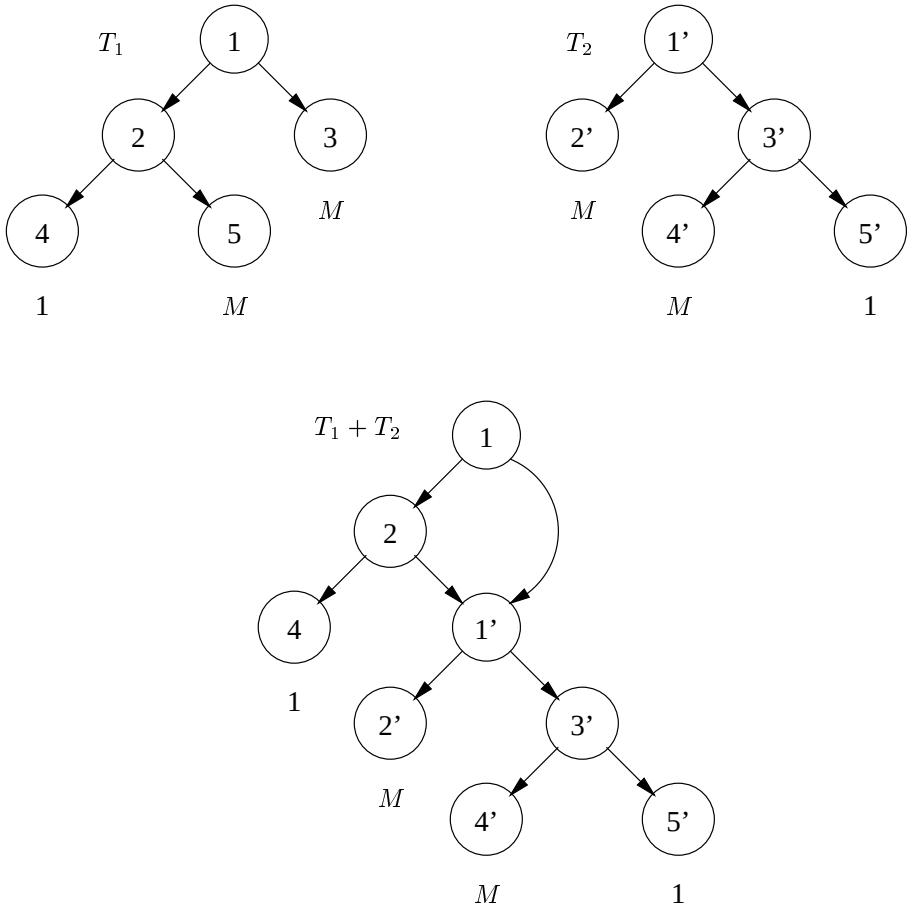

Figure 13: Sequential ensemble of progressive decision trees. The resulting tree becomes a decision graph.

3.6.2 Sequential ensemble problems

But resulting tree size is not the only problem we have to handle. Sequential ensembles may generate sub-optimal trees when the first split of the second tree reproduces approximately one split of the first tree. This is caused by a bad policy of joining mixed leaves, but it is easy to detect and to solve when trees are combined. This problem does not affect classification performance, but it increases classification cost.

Similar problems appear when the first tree is not correctly simplified, as shown in Fig. 14, where a simple tree has been built as the first stage of a progressive classification system. If both internal nodes 1 and 2 use the same classification feature x_1, such a tree may be obviously further simplified removing the first split. This may happen when a classification feature predominates more than the

others and it has a bimodal distribution, inducing the two first best splits $x_1 \leq h_1$ and $x_1 \leq h_2$. This is a clear example of the repetition problem.

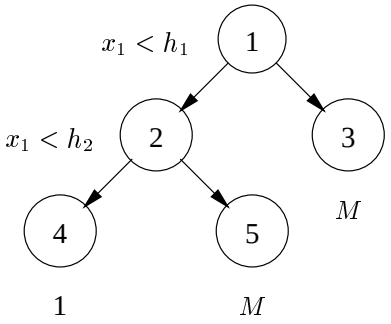

Figure 14: A tree that can be further simplified.

When the number of classes is high, it is usually impossible to create a small decision tree which has at least a leaf labeled as class k for each $k \in \{0, \ldots, K-1\}$. Of course, T must have at least K leaves, which may cause T to be too deep for some classes, specially when classes are not equally distributed in the training set.

3.6.3 Parallel/sequential ensemble

As stated previously, when the number of classes is high, it is difficult to create small progressive decision trees that classify all classes. Instead of creating larger progressive decision trees which could be too specific, it is better to take another approach more directly related to class clustering.

The classical approach is to build a decision tree for each pair of classes, trying to maximize the differences between classes. This leads to $\binom{K}{2}$ classifiers, which may be very expensive for large K. Our approach is to create a small progressive decision tree for each class, that is, a tree that only classifies elements of one class, trying to label as mixed all elements from the rest of classes. Therefore, only K small progressive decision trees need to be built (for the first stage). Then, a method to compute the assigned label combining the K outputs is needed. This is the basic procedure for constructing a multiclass classifier using only basic two-class classifiers, as stated in Vapnik (1998, p. 438). Several authors have recently proposed several methods for classifier combination (Woods, Jr. and Bowyer, 1997; Kittler, Hatef, Duin and Matas, 1998; Ho, Hull and Srihari, 1994) which take into account the problem of building classifiers when more than two classes are present. A paper which unifies previous works related to this subject is (Allwein, Schapire and Singer, 2000).

Notice that we use the extra mixed class M in order to allow our classification system to do partial classifications, simplifying also the combination scheme: after a vector has been classified by the K classifiers and K labels are obtained, we label such vector as class j if all classifiers generated

the mixed class M as output but the classifier j. If two or more classifiers generate a label other than M, or when all classifiers generate the mixed class as output, then the vector is labeled as mixed. Notice that this is a very restrictive labeling policy, and it could generate the mixed label only if all decision tree regions overlap. It could be improved by using a weighted voting scheme based on estimated class probabilities. The main advantage is that different $\overline{d}$ and ϵ may be used for each class.

Then, the set of progressive decision trees and the combiner may be seen as a single progressive classification system, and therefore they may be used as a single progressive decision tree.

3.6.4 Parallel/sequential ensemble problems

When the number of classes is large, some classes usually represent only a small percentage of the training set. In order to build a specific decision tree for such classes we need to descend deeply to separate one class from the rest (unless a class is really a cluster and is easily separable, which is unlikely). Furthermore, region decision for such a tree may be covered by other decision trees associated to more populated classes, so the label of such tree will never be chosen as the output label.

Classification cost is also an important issue when using this kind of ensemble. A decision tree must be built for each class, and classification cost is also multiplied by the number of classes, as the input vector must be classified by each tree.

Although this kind of ensemble has interesting properties, we do not use it in this thesis because it increases classification cost and the number of parameters involved in. The combination of specialized experts using complex schemes is an active research field nowadays (Kittler and Roli, 2000; Kittler and Roli, 2001; Kittler and Roli, 2002).

3.7 Cascading generalization

As described in Sect. 3.1.2, each decision tree is trained using a training set which is formed by the vectors not classified by the previous decision tree. This is the only information that we are carrying over from one stage to the next. Nevertheless, when an input vector is classified as mixed it falls in a leaf which describes a region of the input space: leaf probability, depth, misclassification error and class probabilities are important information that might be used in the following decision tree. This approach is used by Gama and Brazdil (2000), where class probabilities are carried over from one classification system to the next as new information. This is known as cascading with additional information.

Regarding progressive decision trees, in order to make a complete comparison we propose to study three different types of cascading ensembles:

Type A: No additional information is carried over from one decision tree to the next, and only vectors classified as mixed form the new training set. This is the approach we have described in this chapter, and it is shown in Fig. 15. Following from Eq. (3.2), the training set generated at stage $i+1$ (by a decision tree T_{i+1}) may be computed as:

$$D_{n_{\overline{i+1}}}^{(\overline{i+1})} = D_{n_{\overline{i}}}^{(\overline{i})} \ominus D_{n_{i+1}}^{(i+1)}.$$

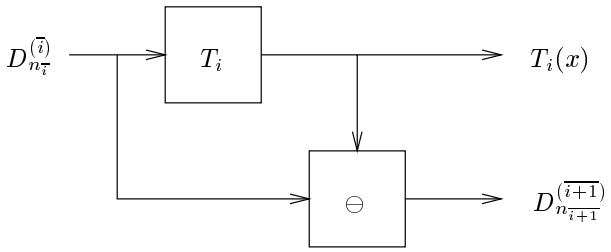

Figure 15: Stage architecture for a type A progressive decision tree.

Type B: Additional information is carried over from one decision tree to the next, and all vectors in the previous training set form the new training set:

$$D_{n_{\overline{i+1}}}^{(\overline{i+1})} = \sharp(D_{n_{\overline{i}}}^{(\overline{i})}, \Lambda^{i+1}(D_{n_{\overline{i}}}^{(\overline{i})}))$$

where $\sharp$ represents the transpose concatenation operator, and Λ^{i+1} represents the additional information operator associated to tree T_{i+1}, as shown if Fig. 16. $\Lambda^{i+1}(x), x \in D_{n_{\overline{i}}}^{(\overline{i})}$ gives all information related to the final decision taken about the label x is classified with. As stated in (Ting and Witten, 1997), it is better to use class probabilities (which are continuous) rather than class predictions (which are discrete). On the other hand, suppose D_n is a training set containing n vectors of dimension d and D'_n another set of n elements of dimension d'. Then, $\sharp(D_n, D'_n)$ is a new training set containing n elements of dimension $d + d'$. As D'_n does not contain labels, these are carried over from D_n. Notice that in this case, $n_{\overline{i+1}} = n_{\overline{i}} = \ldots = n$.

Type C: Finally, if we combine the two types described above, as shown in Fig. 17, additional information is carried over and only those vectors left unclassified by the previous classifier form the new training set, as follows:

$$D_{n_{\overline{i+1}}}^{(\overline{i+1})} = \sharp(D_{n_{\overline{i}}}^{(\overline{i})} \ominus D_{n_{i+1}}^{(i+1)}, \Lambda^{i+1}(D_{n_{\overline{i}}}^{(\overline{i})} \ominus D_{n_{i+1}}^{(i+1)})).$$

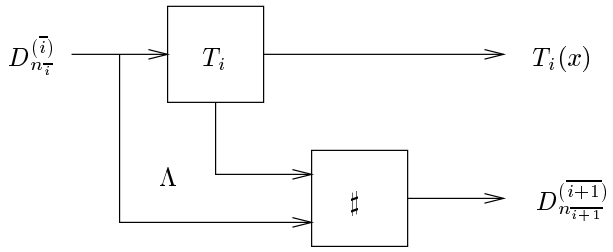

Figure 16: Stage architecture for a type B progressive decision tree.

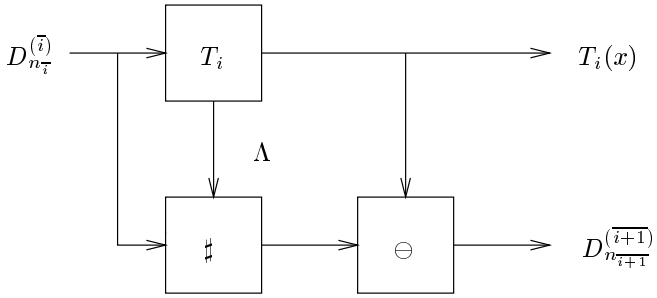

Figure 17: Stage architecture for a type C progressive decision tree.

3.7.1 Additional information

Suppose an input vector x is being classified by a given decision tree T_i. What information is generated during this process? Is it useful for the next decision tree in a cascading ensemble? The first question is easy to answer:

1. Until x reaches a leaf, it is classified according to each decision taken in a decision node. The distance from x to the splitting hyperplane may be a good indicator of the confidence x has been classified with. The path itself defines a sequence of partial classifications (each decision node has been a leaf in a certain moment during the construction of the decision tree).

2. Once x has reached a leaf, and it is classified according to a labeling rule, several information may be useful: the class assigned to x, the probability of x belonging to each class, leaf probability, leaf depth, and the margin of the decision.

The second question does not have a simple answer. Adding more information does not ensure improving classification performance, unless this information is really significant for the problem we

are trying to solve. Furthermore, under a statistical framework, increasing dimensionality usually requires an exponential increase of the available number of samples, which is not the case in a cascading ensemble. This negative effect is known as *curse of dimensionality*, see (Duda et al., 2000, pp. 169–170), for example. Increasing dimensionality has also a negative impact on both training and classification costs, so this possibility should be used carefully.

Experiments that were previously done to the experiments described in Sect. 4.5 showed that adding too much additional information caused a decrease in classification performance, or that such information was never used for splitting. Other measures we could use as additional information are leaf impurity and misclassification error, leaf depth, and the probability of falling in such leaf, but they did not yield to a better performance in general. Therefore, we only use leaf class probabilities $\{p(j|t)\}$ and two measures related to the margin, Θ_1 and Θ_2, defined as follows:

$$\Theta_1(p) = 2\max\{p(j|t)\} - 1,$$

and

$$\Theta_2(p) = \max\{p(j|t)\} - \max_2\{p(j|t)\},$$

where $\max_2$ return the second higher value in $\{p(j|t)\}$.

Notice that for $K = 2$ (actually, when the number of present classes in a leaf is two), both measures agree. Θ_2 measures how good a decision is with respect to the second option in a given leaf, and it is always non-negative. Θ_1 measures the same but with respect to the rest of the classes present in a given leaf, and it may be negative. It is easy to see that Θ_1 is the probability of x being correctly classified minus the probability of x being misclassified, the definition of margin given in (Schapire et al., 1998), as we will see in Sect. 3.8.2. We introduce the use of Θ_2 because it gives useful information about the rest of classes. Suppose $K = 3$, and $p = \{0.5, 0.499, 0.001\}$. In this case, the choice is between the first two classes, $\Theta_1(p) = 0$ and $\Theta_2(p) = 0.001$, showing that there is almost a tie. If $p = \{0.5, 0.25, 0.25\}$, $\Theta_1(p)$ is zero again, but $\Theta_2(p) = 0.25$, showing that there is a large margin in the decision.

The new attributes computed by the first tree express relations between the input variables, which are impossible to find using only orthogonal decision trees. These new attributes may be seen as complex relationships which may help to build more complex boundaries.

3.8 Empirical support for progressive decision trees

Our goal is to give empirical support for our ensemble: although experiments show that it is possible to improve classification accuracy while reducing classification cost (see (Minguillón et al., 1999), for example), we want to study why this happens. Our intuition is that the use of small decision trees captures the best splits reducing overfitting, while maintaining a good classification accuracy.

In fact, it is not easy to have a model or theory which allows us to establish relationships between the different parameters involved, and decision trees need an important problem tuning effort in order to obtain good results, as stated in (Murthy, 1997), where data specific preprocessing plays an important role in improving classification performance. The usual way is to define a set of measures over the classification system under a set of assumptions and study the influence of each parameter separately over the proposed measure.

3.8.1 Empirical measures

We will use the same approach used by Gama and Brazdil (2000), where three measures are used to determine whether a classifier ensemble may improve or not classification performance. These measures are error correlation, Mahalanobis distance and bias-variance decomposition. We will compare the three types of cascading ensembles described above using the later, which seems to be the most effective tool for algorithm analysis. We will try also to find answers to the questions listed in Sect. 3.4 using the information extracted from the bias-variance decomposition.

3.8.1.1 Error correlation

The basic idea of combining classifiers is that when a classifier makes a mistake, maybe many other classifiers do not, so a simple voting system could perform better. Suppose we are combining only two classifiers, g_1 and g_2. If both g_1 and g_2 make the same mistakes, the fact of combining two classifiers does not improve performance. Therefore, it would be interesting that an ensemble of classifiers will have the *diversity* property, as defined in (Ali and Pazzani, 1996). The concept of *error correlation* is a metric used to measure the degree of diversity in an ensemble.

Regarding progressive decision trees, error correlation could be defined using a similar approach: we define error correlation between two progressive classification trees T_i and T_{i+1} as the probability that T_{i+1} makes the same mistake than T_i would have made if the mixed class was not used. Following similar notation than (Gama and Brazdil, 2000), we define the error correlation between two consecutive decision trees T_i and T_{i+1} as:

$$\phi_{i,i+1} = \mathbf{P}\{T_{i+1}(x) = T_i(x) | T_i'(x) = M \wedge T_i(x) \neq f(x)\} \tag{3.3}$$

where $T_i'(x) = M$ is the condition that describes the fact that T_i decides not to classify x and, therefore, it is classified as mixed by labeling rule defined in Eq. (3.1) and then it is wrongly classified again by T_{i+1} making the same mistake.

We decided to discard error correlation as an analytical tool because it does not give us information about the real behavior of each ensemble, but only about how a decision tree is affected by the decisions taken by the previous one. Nevertheless, error correlation is directly related to the first two questions in Sect. 3.4, as the second part of Eq. (3.3) is directly related to ϵ. Suppose $K = 2$, and the confusion matrices for T_i and T_{i+1} are:

$j\backslash k$	0	1	M	$\sum$
0	$n_{0,0}^{T_i}$	$n_{0,1}^{T_i}$	$n_{0,0,M}^{T_i} + n_{0,1,M}^{T_i}$	n_0
1	$n_{1,0}^{T_i}$	$n_{1,1}^{T_i}$	$n_{1,0,M}^{T_i} + n_{1,1,M}^{T_i}$	n_1
$\sum$	$n_0^{T_i}$	$n_1^{T_i}$	$n_M^{T_i}$	n

confusion matrix for T_i

$j\backslash k$	0	1	M	$\sum$
0	$n_{0,0}^{T_{i+1}}$	$n_{0,1}^{T_{i+1}}$	$n_{0,0,M}^{T_{i+1}} + n_{0,1,M}^{T_{i+1}}$	n'_0
1	$n_{1,0}^{T_{i+1}}$	$n_{1,1}^{T_{i+1}}$	$n_{1,0,M}^{T_{i+1}} + n_{1,1,M}^{T_{i+1}}$	n'_1
$\sum$	$n_0^{T_{i+1}}$	$n_1^{T_{i+1}}$	$n_M^{T_{i+1}}$	n'

confusion matrix for T_{i+1}

where $n_{0,0,M}^{T_i}$ denotes the number of elements of class 0 which are classified as 0 by T_i but, because of its insufficient confidence, are finally classified as mixed, and so. Notice that several relationships may be established between some entities in both confusion matrices, for instance:

$$n' = n'_0 + n'_1 = n_{0,0,M}^{T_i} + n_{0,1,M}^{T_i} + n_{1,0,M}^{T_i} + n_{1,1,M}^{T_i} = n_M^{T_i},$$

that is, the second tree is trained using those vectors in the original training set not classified by the first tree. Notice that, according to Eq. (3.2), n should be denoted as $n_{\overline{i-1}}$ and n' as $n_{\overline{i}}$. Error correlation may be computed as follows:

$$\begin{aligned}\phi_{i,i+1} &= \mathbf{P}\{T_{i+1}(x) = 1 | T'_i(x) = M \wedge T_i(x) = 1 \wedge f(x) = 0\} + \\ &\quad \mathbf{P}\{T_{i+1}(x) = 0 | T'_i(x) = M \wedge T_i(x) = 0 \wedge f(x) = 1\} \\ &= \frac{n_{0,1,M}^{T_i}(n_{0,1}^{T_{i+1}} + n_{0,1,M}^{T_{i+1}})}{n(n_{0,0,M}^{T_i} + n_{0,1,M}^{T_i})} + \frac{n_{1,0,M}^{T_i}(n_{1,0}^{T_{i+1}} + n_{1,0,M}^{T_{i+1}})}{n(n_{1,0,M}^{T_i} + n_{1,1,M}^{T_i})}\end{aligned}$$

If ϵ is too small, T_i will be a small tree, and most of the input vectors will be classified as mixed. Then, error correlation only measures $T_i + 1$ performance, not the real ensemble properties. Error correlation is a useful measure when more than two classifiers are involved in a voting scheme, as we expect such scheme to weight errors in order to correct the mistakes made by one classifier by the average of the rest. When only two classifiers are sequentially combined, error correlation becomes almost meaningless, so we decided to discard it as an analytical tool.

Analogously, we could define a new measure involving those vectors not classified by T_i that are also not classified by T_{i+1}. We call it *mixed class correlation* and it is defined as follows:

$$\overline{\phi}_{i,i+1} = \mathbf{P}\{T_{i+1}(x) = M | T'_i(x) = M\} \tag{3.4}$$

Nevertheless, as only those vectors not classified by T_i form the training set for T_{i+1}, this measure only represents the percentage of input vectors not classified by T_{i+1}, and gives no information about the ensemble in itself. This measure would have sense in the second kind of cascading ensemble described in Sect. 3.7, where all input vectors are carried over from one classifier to the next with additional information.

3.8.1.2 Mahalanobis distance

Mahalanobis distance (Jobson, 1992) is a powerful tool that tries to explain mathematically the shape and distance between clusters in a multidimensional space. Classification becomes a clustering

problem. Each class is supposed to be a single cluster (which may be not true, though), so we can define a centroid for each class. Two measures are defined: the *intra-class* distance, and the *inter-class* distance. The first distance measures how close to the centroid of a class the elements of such class are, while the second distance measures the distance between class centroids.

Mahalanobis distance is computed as follows: let $\overline{x}_k$ be the mean vector for all elements of class k in the training set D_n, n_k the number of elements of class k (so $\sum_{i=0}^{K} n_{k_i} = n, k_i \in \{0, \ldots, K-1\}$), and S_k the covariance matrix for class k. Then, the intra-class distance from a vector x of class k to its class centroid is computed as:

$$(x - \overline{x}_k)^T S_k^{-1} (x - \overline{x}_k),$$

and the inter-class distance between classes k_1 and k_2 is computed as:

$$(\overline{x}_{k_1} - \overline{x}_{k_2})^T S_{k_1,k_2}^{-1} (\overline{x}_{k_1} - \overline{x}_{k_2}),$$

where S_{k_1,k_2} is the pooled covariance matrix of classes k_1 and k_2, which is computed as:

$$S_{k_1,k_2} = \frac{S_{k_1}(n_{k_1} - 1) + S_{k_2}(n_{k_2} - 1)}{n_{k_1} + n_{k_2} - 2}.$$

Under a classification framework, it is interesting to have compact classes (that is, with small intra-class distance) which are well separated one from each other (that is, with large inter-class distance). The basic idea is again to use a first classifier to separate the easiest classes and then to refine such classification using a sequence of specialized classifiers.

When additional information is added to an input vector, increasing its dimensionality, both the class centroid and covariance matrix also increase (in the sense of that columns and rows are added). This obviously changes the values obtained for both the intra-class and inter-class distances. Even if we do not use additional features, as Mahalanobis distance is computed using input data statistics, it is changed when we alter the underlying probability distribution because of the removal of those vectors already classified at a given stage.

Nevertheless, it is difficult to interpret variations using the Mahalanobis distance, specially when the number of classes is high. When the number of classes is 2, it is easy to show that the inter-class distance grows, but for K classes, a total of $\binom{K}{2}$ should be computed and then plotted. Furthermore, the assumption that classes form clusters is completely unreliable, so we discarded the use of the Mahalanobis distance as an analytical tool.

3.8.1.3 Bias-Variance decomposition

The bias-variance decomposition is probably the most effective tool for an empirical analysis of the performance of a classifier, as it measures how good a learning algorithm is and how much affected by small perturbations in the training set is. As stated in (Domingos, 2000), it is also related to error generalization and margin measures defined in (Schapire et al., 1998).

We will follow the bias-variance decomposition described by Domingos (2000)[3], which unifies previous works by Dietterich and Kong (1995), Kohavi and Wolpert (1996), Breiman (1996b) and Friedman (1997).

The basic idea consists of decomposing the expected error into three components: intrinsic noise, bias and variance. As stated in (Kohavi and Wolpert, 1996), the intrinsic noise is a lower bound on the expected cost of any learning algorithm, and it is independent from the learning algorithm. Usually, the intrinsic noise is assumed to be zero. As it is estimated using bootstrapping, it would be zero as all samples in the training set are unique (that is, there are no two equal samples in the training set with different labels). This assumption it is not really important as we want to study the decomposition in itself, and not the quantities.

Bias is defined as the loss incurred by the main prediction relative to the optimal prediction, and it is denoted by $\hat{B}$. Variance is the average loss incurred by predictions relative to the main prediction, and it is denoted by $\hat{V}$. In other words, the bias measures the accuracy or quality of the prediction (high bias implies a poor prediction), while the variance measures the precision or specificity of the prediction (a high variance implies a weak prediction). There is a trade-off between bias and variance: classifiers which tend to reduce bias have a large variance and vice versa. For classification purposes, a low variance is usually more important than a low bias.

Regarding decision trees, it is well known that they are unstable classifiers, as small variations in the training set may cause large variations in the final prediction. This fact shows that large decision trees have a large variance but bias is kept low. Our experiments in Sect. 4.4 also show this fact. As stated in (Gama and Brazdil, 2000), it is better to combine classifiers with different behavior from a bias-variance decomposition analysis, using algorithms with low variance at the first stages and algorithms with low bias at the final stages. These facts are consistent with our ensemble:

- Small decision trees created using an impurity criterion cannot describe all the complexity of the problem being solved, but only a first approach. Their bias is high, but they are less sensitive to training set changes, showing a low variance.

- Large decision trees can generate very complex boundaries, reducing bias, but variance grows because the final splits are very sensitive to small changes in the training set.

Both facts may be studied under a margin distribution approach: small trees which partition the input space in large regions yield a large margin, while large trees with too many leaves generate partitions with too many small regions which have a poor margin distribution.

This gives us information about the first and the fourth questions defined in Sect. 3.4, as tree sizes and number of trees are directly related. Our experiments show that only two or three trees are needed to create a good progressive decision tree for most data sets. Increasing the number of trees reduces comprehensibility and it makes the parameter tuning problem (specially ϵ and $\overline{d}$)

[3] There is available software at `http://www.cs.washington.edu/homes/pedrod/bvd.c`

more difficult. As stated in (Gama and Brazdil, 2000), different algorithms must be combined (or algorithms with different bias-variance behavior). If we use only decision trees, we need to create decision trees with such bias-variance different behavior, which is not always possible depending on data size and number of classes, as only two parameters (in our experiments) are allowed to change, ϵ and $\overline{d}$. These fact reduces our options to only two or three decision trees.

Our goal is to use the bias-variance decomposition to measure how progressive decision trees deal with the classification problem. We expect our ensemble to outperform classical decision trees and we are interested in studying where this better performance is achieved, if reducing bias and/or variance. We are also interested in comparing the three kinds of cascading ensembles (defined in Sect. 3.7) under this framework, in order to study whether classification cost may be reduced or not.

3.8.2 Margin distribution

A classifier g may be studied as a function which maps an input vector $x \in \mathbb{R}^d$ to a label $y \in \{0, \ldots, K-1\}$. Suppose that $K = 2$ and that we use $\{-1, +1\}$ to identify the two possible labels instead. Now, the output generated by the classifier may be described as $y = \text{sgn}(g(x))$, where $g(x)$ is now a function which maps x to $\mathbb{R}$. Then, the magnitude $|g(x)|$ may be seen as a measure of confidence in the prediction, and it is known as the *margin* of x, as defined in (Schapire et al., 1998). In fact, the margin is defined as $yg(x)$, so it is negative when the classifier makes a mistake and positive when it is right.

Intuitively, the largest the average margin, the better the classification accuracy. Voting systems such as bagging and boosting benefit from a better margin distribution. For other ensemble methods such as stacking or cascading there are no previous works measuring margin distribution, though. The reason is that margin distribution is usually measured for ensembles that make linear combinations of several functions such as the voting schemes, and all the analytical tools are defined using the concept of the *convex hull* defined by such linear combination.

Margin is a powerful analysis tool as it relates each classifier performance to global generalization error and it explains intuitively why combining slightly better classifiers performs better. The problem is that margin is not related to decision trees, as these are built using criteria related to splitting data sets, not maximizing the distance between the elements separated by a hyperplane. Furthermore, margin is also related to voting schemes where classifier results are averaged, which is not our case, as progressive decision trees fall into the cascading category of ensembles. Therefore, we decided to discard margin as a measure for analyzing decision trees.

As we will see in Chapter 5, all approaches trying to incorporate margin analysis into decision trees do not produce good error generalization bounds, although they allow us to study decision trees under a theoretical framework.

3.9 Summary

We finish this chapter with a compilation of all the knowledge we have acquired when defining progressive decision trees, and their advantages and drawbacks with respect to classical decision trees.

Progressive decision trees deal with the replication, repetition and fragmentation problems following the same approach: joining regions which had been split previously, giving to the classical decision tree growing algorithm new chances to find better splits. Fragmentation is combated by joining, so new large regions are created. This also minimizes both the repetition and replication problems, as complete subtrees are joined together, so there is no need to represent the same subtree twice. Decision trees become decision graphs which are richer structures for representing more complex boundaries.

Now we can also give some partial answers to the questions defined in Sect. 3.4, using all the knowledge we have acquired through this chapter, while other questions will be answered using the results of the experiments described in the next one. The basic structure of our ensemble is a special case of cascading, so we have generalized the cascading paradigm in order to explain the three types of progressive decision trees defined in Sect. 3.7. Within this paradigm, the parameters (or questions) involved with are:

1. *How deep a single tree has to be?* The bias-variance decomposition helps us to answer this question: as stated in (Gama and Brazdil, 2000), in a cascading ensemble is interesting to combine classification systems with different bias-variance decomposition behavior, so simple decision trees should be used at the first stages and complex decision trees at the last ones. This is coherent with the intuitive idea behind progressive decision trees: only the first splits are necessary (that is, small decision trees), in order to avoid overfitting.

2. *How to set ϵ?* It is harder to find a question for this answer than for the other ones. The value of ϵ should not be too large (at least when compared to $\hat{L}_n$) or the first decision tree would be almost a classical decision tree. On the other hand, if ϵ is too small, the first decision tree may be also too small, even a trivial decision tree (a single leaf labeled as mixed). In the following chapter we will try to find a better answer for this question using an exhaustive empirical evaluation of decision trees varying this parameter.

3. *Is it pruning necessary?* When $\overline{d}$ is small, pruning is not necessary (because nothing is pruned back). Nevertheless, pruning small decision trees is not a problem in a training cost sense, so we do pruning even for small decision trees. On the other hand, for large trees pruning is the best method to obtain trees with a right size, avoiding overfitting without compromising performance.

4. *How many trees are needed?* Usually, the bigger the training set is, the larger the number

of small decision trees can be. But, as we always use the same learning algorithm (decision trees), it is difficult to create different (in a bias-variance decomposition sense) consecutive decision trees, so usually only two or three decision trees are needed. Furthermore, at each stage the problem of building a decision tree becomes harder and harder. We will try to give a better answer in the following chapter by varying the number of decision trees in the ensemble. Nevertheless, as only different classifiers should be combined, and it is difficult to create very different classifiers using small decision trees, the number of trees will be probably small.

5. *Which leaves must be joined?* In order to avoid inefficient ensembles, only connected leaves (in a region sense) should be joined. This may be efficiently done for small trees. Nevertheless, if we are only interested in studying generalization error performance, this is not a key issue so it can be ignored.

In the following chapter we will define a set of experiments to determine some of the parameters related to progressive decision trees. Finally, in Chapter 5 we will try to give some theoretical support to all the intuition behind progressive decision trees using previous results for combinations of classifiers.

Chapter 4

Experimental results

Pratique: Supérieure à la théorie
Gustave Flaubert, *Le Dictionnaire des Idées Reçues*

In this chapter, we present the results obtained for several kinds of decision trees and cascading ensembles using a standard database of data sets commonly accepted as a standard by the machine learning community. We also present in this chapter the results obtained with real data sets that have been previously used in other projects: a document layout recognition system, remote sensing of hyperspectral images, and a brain tumor classification system. A recent paper on web mining can be found in (Mor and Minguillón, submitted).

This chapter is the main core of this thesis, as we provide empirical support for our ensemble comparing the three kinds of progressive decision trees defined in Sect. 3.7, using a parameter-based approach: each parameter is studied independently, trying to extract any useful information for constructing progressive decision trees. As we use classical decision trees as base-level classifiers, we also study the more relevant parameters in the training process of such classifiers. To do so, we use the empirical measures defined in Sect. 3.8 as basic tools for comparing classifiers.

Our goal is to study cascading ensemble performance using the bias-variance decomposition (Domingos, 2000) as the basic tool for determining when cascading improves generalization error and why. A paper containing the most important contributions of this chapter may be found in (Minguillón and Pujol, submitted).

4.1 Data sets used for experiments

4.1.1 Document layout recognition

The first time we used progressive decision trees it was in a project related to document layout recognition systems (see (Minguillón et al., 1999)). Our goal was to build a simple but fast classification

system that could be used as a first stage of a document processing and storage package. Nevertheless, the meaning of the word "progressive" in such system is slightly different, as a hierarchical structure was built in order to reduce classification cost taking advantage of classification features redundancy. The main idea is that image is segmented in large blocks, and then a first decision tree tries to classify each block. If a block cannot be correctly classified (under an accuracy constraint) in this first stage, such block of size $N \times N$ is subdivided in four sub-blocks of size $N/2 \times N/2$ and the process is repeated recursively for each sub-block. The maximum and minimum block size were set to $N = 64$ and $N = 8$ respectively, the JPEG standard (Wallace, 1991) block size.

The data set used in this paper was donated by Hewlett-Packard, and it was composed by 24 images of 1275×1755 pixels (8.5×11.7 inches, scanned at 150dpi), which were labeled at 64×64, 32×32, 16×16 and 8×8 block size resolutions, doing block padding when necessary. Due to the difficulty of the labeling process, we suspect there is a small percentage of mislabeled blocks in the smaller block sizes, although this percentage is probably below 5%. A total of 16 images were used in the training set and the other 8 images were used for testing purposes.

A progressive decision tree was built using four small trees, one tree for each block size resolution. Tab. 3 compares the size of the four small decision trees with a large decision tree created using the target block size (8×8).

Block size	$\lvert T \rvert$	$\lvert \tilde{T} \rvert$	R	$\overline{d}$
Progressive classification scheme				
64×64	11	6	2.76696	4
32×32	27	14	4.16826	6
16×16	21	11	3.71954	6
8×8	35	18	4.73452	8
Non-progressive classification scheme				
8×8	1441	721	8.55861	38

Table 3: Tree sizes for the document layout recognition system.

Notice that $\overline{d}$ (actually limited by and also the maximum depth allowed during training) grows for smaller block sizes. The reason why we need deeper decision trees at each stage is that, firstly, a increasing number of blocks is present in the training set used at each stage, and secondly, at each stage the classification problem becomes more complex as the easiest samples have been already processed by the previous stage.

The last decision tree increases misclassification error although the total error remains below the one-stage decision tree, as shown in Tab. 4. This is probably caused by two facts: first, most errors during the labeling process were done at the 8×8 block size, and second, such block size is too small to differentiate between the different classes (even for a human).

Block size	Processed / total blocks	Classified blocks (stage)	Classified blocks (total)	$\hat{L}_{n,m}$
Progressive classification scheme				
64×64	3360 / 3360	41.4%	41.4%	0.089
32×32	7856 / 13440	33.0%	60.1%	0.047
16×16	21052 / 53760	66.9%	86.8%	0.042
8×8	27892 / 215040	100.0%	100.0%	0.065
Non-progressive classification scheme				
8×8	211200 / 211200	100.0%	100.0%	0.078

Table 4: Classifier performance for the document layout recognition system.

Nevertheless, the most remarkable fact is that only 60160 blocks need to be classified by the progressive classification scheme, while the one-stage decision tree needs to classify 211200 blocks, so the classification cost is reduced. Furthermore, the multi-stage classification system generates larger homogeneous areas which are easier to interpret in a document layout recognition system.

This first experiment shows that it is possible to exploit image redundancy at different block sizes, improving performance and reducing classification cost using small progressive decision trees.

4.1.2 Hyperspectral imaging

For several reasons, hyperspectral imaging might be considered one of the most important applications of pattern recognition nowadays. Ecology, environmental studies, precision farming, and even military applications take advantage of the information present in all the bands captured by a hyperspectral sensor: each kind of material has a typical spectral signature that may be used to identify it. The problem is that this signatures have a large dimensionality (from a few bands up to hundreds of bands), and they are affected by the distortion caused by the atmosphere, so costly corrections are needed. Large dimensionality is usually faced using a few parts of the complete spectral signature or doing a dimensionality reduction through principal component analysis (Jobson, 1992) or independent component analysis (Hyvärinen, Karhunen and Oja, 2001), for example.

We also used progressive decision trees for hyperspectral imaging classification, see (Minguillón, Pujol, Serra and Ortuño, 2000; Minguillón, Pujol, Serra, Ortuño and Guitart, 2000). Although the main goal of the original paper was to establish a relationship between lossy compression and classification accuracy, a set of decision trees was also constructed in order to compare the influence of lossy compression on both kinds of decision trees.

In this experiment, a large labeled data set is available from CREAF,[1] which was first described in (Baulies and Pons, 1995). Usually, hyperspectral images lack of ground truth data, which is not

[1] Centre de Recerca Ecològica i Aplicacions Forestals, Universitat Autònoma de Barcelona.

the case for this data set. Nevertheless, this data set has been labeled using other criteria than exact the representation of terrain (such as sociological and terrain use, for example), so labels do not exactly match reality and, therefore, it is impossible to find very accurate decision trees. Our estimate is that around a five or ten percent of labels is misclassified. In our paper we used progressive decision trees not only to build a classification system but also to detect those areas most likely to be mislabeled.

The original hyperspectral image is 710 pixels wide by 4558 pixels long, although there are no labels for the whole image, and some parts of the image are missing due to ortocorrection. For example, if a complete (we have both data and labels) 400×2400 image is selected, 960000 labeled pixels are available for training (864000) and testing purposes (96000). Using Eq. (2.5) and assuming an error of $\epsilon = 0.01$,

$$\mathbf{P}\{|\hat{L}_{n,m} - L_n| > 0.01 \,|\, D_n\} \leq 2e^{-2\,96000\,0.01^2} = 9.17436 \times 10^{-9}.$$

which is, in a practical sense, zero. There is no need to use more labeled pixels for testing purposes, as Eq. (2.5) ensures we have enough. Notice that, when compared to the data sets described in Sect. 4.1.4, this data set is huge. Algorithms such as the nearest neighbor classifier cannot be used due to excessive computational costs.

In (Minguillón, Pujol, Serra and Ortuño, 2000) we proceeded as follows: first we built a decision tree for such data set (T_1) , and then we re-labeled it using the new labeling rule defined in Eq. (3.1), trying to obtain a good compromise between the percentage of classified pixels and classification performance (T_2). Our goal was to identify the most likely pixels to be mislabeled in the training set. Tab. 5 shows the properties of both trees (P_T is the percentage of input vectors processed by tree T). Notice that both trees are very large, almost ten questions need to be answered before a class is assigned to an input vector.

tree	$\lvert\tilde{T}\rvert$	R	P_T	$1 - \hat{L}_{n,m}$
T_1	836	9.82891	1.0	0.837352
T_2	650	9.60482	0.722118	0.907975

Table 5: Decision trees built for the hyperspectral imaging experiment.

Then we proceeded to build a progressive decision tree using only two stages, as shown in Tab. 6. Our goal was to obtain a tree similar to T_2, but much smaller, while maintaining classification performance.

Both trees (T_2 and T_3) were then compared using several compression algorithms and both classification percentage and accuracy were measured. As expected, T_3 performed much better than T_2, basically due to its small size. See (Minguillón, Pujol, Serra and Ortuño, 2000) for experiment details and (Minguillón, Pujol, Serra, Ortuño and Guitart, 2000) for other similar experiments.

tree	$\lvert\tilde{T}\rvert$	R	P_T	$1-\hat{L}_{n,m}$
T_{3A}	9	3.01563	0.523329	0.943835
T_{3B}	8	2.13816	0.383473	0.800825
T_3	44	4.83963	0.70612	0.906444

Table 6: Progressive decision trees built for the hyperspectral imaging experiment.

These experiments are part of a largest project which is still on development. Our goal is to build a system capable of compressing hyperspectral images maintaining a reasonable classification accuracy. More information can be found at `http://ccd-hyper.uab.es`, a site devoted to projects involving hyperspectral imaging.

4.1.3 Brain tumor classification

Another remarkable project where we have used small decision trees combined with other classifiers in a cascading ensemble is in the INTERPRET project, which is devoted to the classification of brain tumors using *in vivo* magnetic resonance spectra. A brief description of the INTERPRET project and preliminary results may be found in (Tate et al., 2002).

In (Minguillón, Tate, Arús and Griffiths, 2002) a multi-stage classification system which uses a simple combination of voting and cascading is described. Each stage tries to reproduce the path clinicians follow to determine the class of a tumor sample: samples are first classified as tumor or normal. Tumor samples are then classified as benign or malignant. Each one of these two classes is refined trying to identify tumor degree or grade. Finally, each grade is split trying to identify tumor family. This scheme partially reproduces the World Health Organization tumor classification structure, which helps clinicians to feel confident with the classification system.

Each stage is trained separately, using the best classification features which are selected using a very simple algorithm based on variable importance determined by a combination of classical correlation analysis and splitting discrimination power. As every classifier uses its own set of classification features, different information may be extracted and then combined into a more robust classification system.

At each stage, three different classifiers are built: a small decision tree, a LDA classifier and a k-NN classifier. The LDA classifier is the first stage of a cascading ensemble with the decision tree, which uses all the additional information computed by the LDA algorithm (pseudo-class probability). Then the outcome of the decision tree and the k-NN classifier are combined using a very simple voting scheme: if both labels agree, generate such label as the outcome of the stage, or a special mixed label otherwise. Fig. 18 shows the architecture of each classification stage.

Fig. 19 shows a simplified version of the World Health Organization (WHO) tree structure which

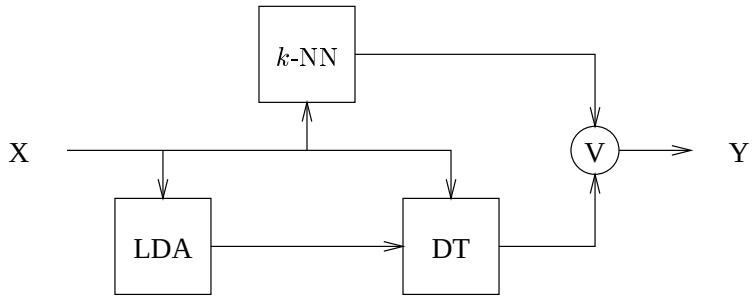

Figure 18: Cascading and voting architecture for each classification stage.

is commonly used by clinicians to diagnose brain tumor category and class. See (Minguillón, Tate, Griffiths, Majos, Pujol and Arús, 2002) for details. We have reproduced this structure using different training sets at each stage following the stage architecture shown in Fig. 18, as previously described. Classification accuracy for each category class is also shown. Notice that classification performance is overall good, although there are several tumor types which are very difficult to classify.

The use of the mixed class allows clinicians to discard those samples which may be more difficult to classify, maybe because they have been acquired using a bad configuration of the acquisition protocol parameters.

This project is also on development yet. Our goal is to build a decision support system for tumor diagnosis that clinicians might use confidently. More information about this project can be found at `http://carbon.uab.es/INTERPRET`.

4.1.4 UCI standard data sets

In this thesis we have used several data sets that form a standard database which is used by the machine learning scientific community (Blake and Merz, 1998), known as the UCI Machine Learning database. An extension of the this database is under development, and it is available at (Bay, 1999). Table 7 shows the selected data sets, the number of instances, the number of classification features and the number of classes, once they have been preprocessed. The *baseline error* (achievable with the simplest classifier which always labels any input vector with the label of the most populated class) is also shown. A brief description of each of these data sets may be found at the end of this thesis, in Appendix A.

All of these data sets have been selected from a larger list which is available at (Blake and Merz, 1998), because they do not present instances with missing values, and all attributes are numerical or binary. The number of instances is in the form n/m, where n and m are the number of samples used for the training set and corpus set respectively, trying to respect the proportions given in the original data sets, if possible, or using the rule $(2/3, 1/3)$ as described in Sect. 2.7 otherwise.

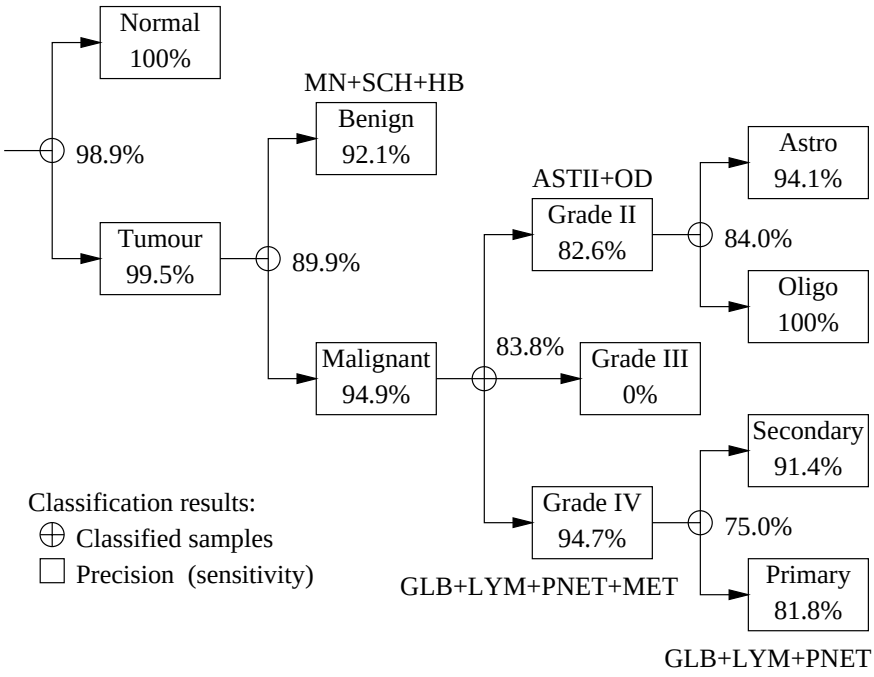

Figure 19: WHO tree and classification accuracy for each tumor family: Non tumors, Tumors, Benign, Malignant, Low Grade, Medium Grade, High Grade, Primary and Secondary.

The exception is the data set named *waveform* because is computer generated and, therefore, any combination of n/m samples is available (see Appendix A for more information about this data set). The number of features is in the form d ($d*$) where d and $d*$ are the original and real (that is, used) number of classification features. The same scheme is also used for the number of classes. Discarded classification features are references, identification numbers, and so. Discarded classes were those not present in the training set or small subclasses which were grouped in a large class (the *lrs* data set).

Although other databases are also available, such as the ELENA database (Avilés-Cruz, Guérin-Dugué, Voz and Cappel, 1995), for example, we focused in the UCI database because it is the most widely used database across the literature, and most of its data sets are not synthetic but they come from real problems. In fact, small databases such as the ELENA database are becoming part of the UCI database.

data set	number of instances	number of features	number of classes	baseline error
covtype (CO)	11340 / 3780	54	7	0.85714
glass (GL)	142 / 72	10 (9)	7 (6)	0.64486
ionosphere (IO)	200 / 151	34	2	0.35897
iris (IR)	102 / 48	4	3	0.66667
letter (LE)	16000 / 4000	16	26	0.95935
liver (LI)	230 / 115	6	2	0.42029
lrs (LR)	354 / 177	102 (93)	100 (10)	0.48588
optdigits (OP)	3823 / 1797	64	10	0.89822
page (PA)	3648 / 1825	10	5	0.10232
pendigits (PE)	7494 / 3498	16	10	0.89592
pima (PI)	576 / 192	8	2	0.34896
sat (SA)	4435 / 2000	36	7 (6)	0.76177
segmentation (SE)	210 / 2100	19	7	0.85714
sonar (SO)	104 / 104	60	2	0.46635
thyroid (TH)	144 / 71	5	3	0.30233
vehicle (VE)	564 / 282	18	4	0.74232
vowel (VO)	528 / 462	12 (11)	11	0.90909
waveform (WA)	n / m	21	3	0.66667
wdbc (WD)	380 / 189	30	2	0.37258
wine (WI)	120 / 58	13	3	0.60112

Table 7: Standard UCI data sets for experiments.

4.2 Splitting criteria

In this experiment we show the relative importance of the splitting criterion used for building decision trees. We test several impurity based splitting criteria and the twoing criterion, which were described in Sect. 2.4.3. Before we try these splitting criteria on the UCI standard data sets, we will try to study their behavior in a simple synthetic data set which would allow us to understand them better.

4.2.1 Synthetic data sets

Fig. 20 shows the synthetic data set created for this experiment. There are two classes in a two dimensional space, 10000 elements for each class. The elements of the first class are in a circle of radius 1 around the origin, while the elements of the second class are in a square of side 2 centered in $(1, 1)$. The total area is $\pi + 4$ and the common area is $\pi/4$, so the expected misclassification error should be around 0.11, approximately. Nevertheless, we are more interested in the splits computed

by each splitting criterion than in the classification accuracy itself.

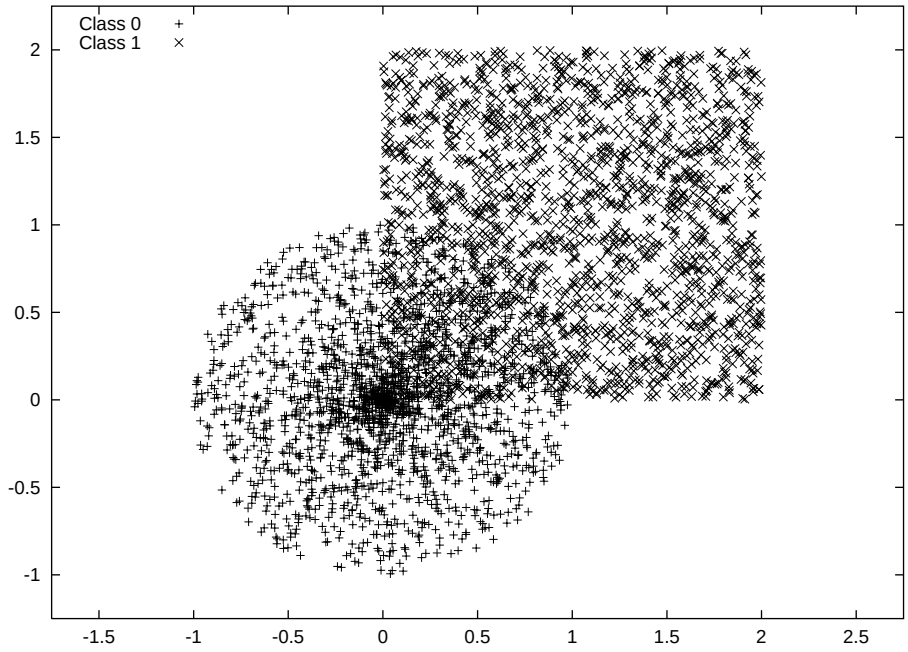

Figure 20: Synthetic data set used for splitting criteria evaluation.

We grow a decision tree with a maximum depth $\overline{d} = 4$, because we are only interested in the first splits. It is easy to see that the common area (actually the 1×1 corner square containing it) can be separated using four splits, although only three are needed to separate the pure areas from another area containing the common area. In this experiment we want to study the form of the splits created by each of the splitting criteria. We number each node in the tree using a positional order: root is node number 1, and the offspring of node i are nodes $2i$ and $2i + 1$.

Tab. 8 shows the computed splits for each splitting criterion. If node i has "—" as split, it means that it is not split (it is a pure node) and, therefore, nodes $2i$ and $2i + 1$ would also have "—" as split, respectively.

Notice that Bayes, Gini, R-norm ($R = 2$) and Twoing criteria generate the same tree with the same splits. Entropy generates a similar tree but the small change in the second split makes it a more balanced tree. But the most remarkable fact is that the tree generated by R-norm ($R = 1/2$) is the closest to the tree that a human would have generated: the split number 1 (which can be considered $x_1 \leq 1$) removes (in the sense of correct classification) the rightmost half of the square, the split number 2 (which can be considered $x_2 \leq 0$) removes the lower part of the circle, while the split number 5 removes the upper left corner of the square. A further split would separate the left

i	Bayes	Entropy	Gini	R-norm (1/2)	R-norm (2)	Twoing
1	$x_2 \leq 0.41$	$x_2 \leq 0.41$	$x_2 \leq 0.41$	$x_1 \leq 0.99$	$x_2 \leq 0.41$	$x_2 \leq 0.41$
2	$x_1 \leq 0.95$	$x_1 \leq 0.99$	$x_1 \leq 0.95$	$x_2 \leq 0.001$	$x_1 \leq 0.95$	$x_1 \leq 0.95$
3	$x_1 \leq -0.001$	$x_1 \leq -0.001$	$x_1 \leq -0.001$	—	$x_1 \leq -0.001$	$x_1 \leq -0.001$
4	—	—	—	—	—	—
5	$x_2 \leq -0.006$	—	$x_2 \leq -0.006$	$x_2 \leq 1$	$x_2 \leq -0.006$	$x_2 \leq -0.006$
6	—	—	—	—	—	—
7	—	—	—	—	—	—

Table 8: Splits generated by each splitting criterion.

upper part of the circle from its right upper part, the common area. As stated in Sect. 2.4.3, the impurity criteria more distant to the Bayes error criterion are precisely the Entropy and the R-norm ($R = 1/2$) criteria.

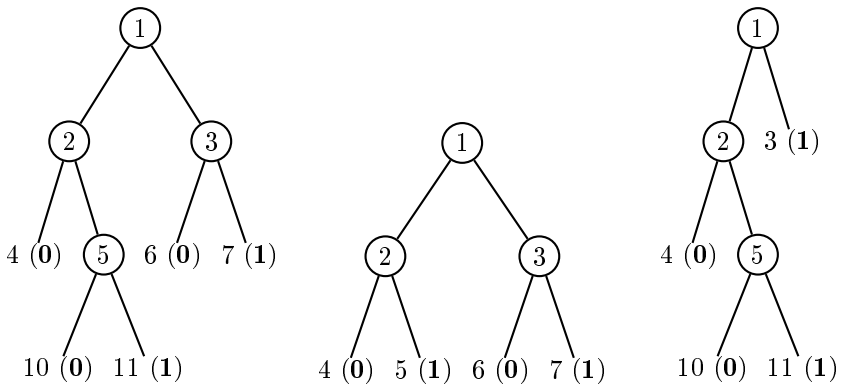

Figure 21: Trees generated by different splitting criteria for the synthetic data set. Labels are shown in parenthesis.

Computed trees are shown in Fig. 21, where the class assigned to a leaf is shown in parenthesis. Notice that the first and the second trees share the same structure (except a further split), while the third tree is slightly different. The main difference is that the latter tries to surround the mixed area instead of splitting it optimally: this is interesting for our purposes as we want to isolate mixed areas in order to perform a later classification. Furthermore, as we will see in the following section, the R-norm splitting criterion using ($R = 1/2$) achieves the best results. A theoretical study of

the splitting ability for the Gini, Entropy and R-norm ($R = 1/2$) splitting criteria may be found in (Kearns and Mansour, 1999), although the R-norm splitting criteria is not mentioned there.

4.2.2 UCI standard data sets

For each data set, 10 independent experiments are carried out using a combination of pruning and 3-fold cross validation (so a total of 30 runs were made of each classifier on each data set). We follow the (2/3,1/3) rule of thumb as stated in Sect. 2.7, regardless the original size of training and corpus sets as defined in Tab. 7. A total of 3000 samples (1000 for each class) were generated for the *waveform* data set. For each experiment we measure average rate R (also called *dynamic complexity* in (Auer et al., 1995)), maximum depth $\overline{d}$ (another tree complexity measure) and misclassification error $\hat{L}$. In order to make honest comparisons, we also compute the standard deviation $\sigma_{\hat{L}}$ of the average misclassification error. We also include the results of a k-nearest neighbor classifier (k-NN) using the same 3-fold cross validation scheme.

Two remarkable facts can be extracted from this experiment. First, there is not a clear winner in terms of classification accuracy. Small training set size is also an important factor that must be taken into account. Accuracy variability is too high in order to choose a splitting criterion, although R-norm using $R = 1/2$ yields the best classification accuracy in ten out of twenty data sets. Furthermore, both R-norm ($R = 1/2$) and Entropy criteria (the two splitting criteria more distant to the Bayes error) yield the best results in fourteen out of twenty data sets. More precisely, if we compute a test of similarity between the best and the second best splitting criterion for each data set, using a significance level of $p = 0.05$ the null hypothesis (splitting criteria perform similarly) cannot be rejected but for the *optdigits* and *pendigits* data sets.

And second, there is a clear loser, namely the Bayes error or misclassification error criterion, which never achieves the best accuracy for any data set. It is also remarkable that the Bayes error is not only the splitting criterion which yields the worse classification accuracy, but it also produces the biggest trees in average. On the other hand, the Twoing criterion only is chosen once (for the *pima* data set) although the Gini criterion yields the same classification accuracy with slightly bigger trees.

Nevertheless, if we rank each splitting criterion according to its performance from 1 (the best) to 6 (the worst) and we compute the total rank, the best is entropy (47 points), followed by R-norm ($R = 1/2$) (53 points), Gini and Twoing tie in the third position (62 points), then R-norm ($R = 2$) (78 points) and finally Bayes error (108 points).

These results are coherent with Breiman et al. (Breiman et al., 1984) conclusions, where it is stated that the Bayes risk is not a good splitting criterion except for the first splits. Our conclusion is that the splitting criteria may not be critical for the general case, but it may be critical for a given data set in terms of relative accuracy. It seems also plausible to discard both the Bayes error and the Twoing criteria (which is surprising for the later).

		data set				
criterion		**CO**	**GL**	**IO**	**IR**	**LE**
Bayes error	R	19.4523	4.96479	2.81581	1.92033	14.0046
	$\overline{d}$	39.3	7.7	3.93333	2.56667	34.5333
	$\hat{L}$	0.23778	0.29061	0.09031	0.056	0.16486
	$\sigma_{\hat{L}}$	0.00695	0.04692	0.02159	0.02703	0.00724
Entropy	R	10.2542	4.54812	2.62422	1.892	10.1426
	$\overline{d}$	21.7333	7.2	4.0	2.9	18.5
	$\hat{L}$	**0.21211**	0.30704	0.08889	0.048	0.13514
	$\sigma_{\hat{L}}$	0.00581	0.05325	0.02202	0.01904	0.00479
Gini	R	10.7638	4.40469	3.14729	1.875	12.3958
	$\overline{d}$	21.5	6.9	4.26667	2.86667	28.0333
	$\hat{L}$	0.21415	**0.28920**	0.08775	0.048	0.14137
	$\sigma_{\hat{L}}$	0.00474	0.05101	0.02397	0.01904	0.00652
R-norm (1/2)	R	9.94822	4.84178	2.4396	1.899	10.0297
	$\overline{d}$	24.5	7.93333	4.0	3.13333	18.9667
	$\hat{L}$	0.21467	0.31315	0.09544	**0.04733**	**0.13394**
	$\sigma_{\hat{L}}$	0.00527	0.05316	0.02593	0.01965	0.00420
R-norm (2)	R	10.168	4.5561	3.15356	1.859	11.8516
	$\overline{d}$	18.7333	6.86667	4.16667	2.83333	24.4333
	$\hat{L}$	0.21929	0.29108	**0.08746**	0.04933	0.14353
	$\sigma_{\hat{L}}$	0.00585	0.04648	0.02353	0.01843	0.00561
Twoing	R	10.7966	4.2176	3.15812	1.901	10.6409
	$\overline{d}$	21.1333	6.86667	4.26667	2.96667	20.2333
	$\hat{L}$	0.21343	0.29014	0.08775	0.048	0.14140
	$\sigma_{\hat{L}}$	0.00600	0.05825	0.02457	0.01904	0.00397
k-NN	k	1	109	1	7	1
	$\hat{L}$	*0.16715*	*0.28732*	0.14245	*0.03533*	*0.04803*
	$\sigma_{\hat{L}}$	0.00537	0.04027	0.03601	0.01979	0.00249

Table 9: Influence of splitting criteria. The best classification accuracy for each data set is shown in bold (when k-NN is the best it is also shown in italic).

		data set				
criterion		**LI**	**LR**	**OP**	**PA**	**PE**
Bayes error	R	2.69029	4.26987	8.67055	10.886	10.0365
	$\overline{d}$	4.16667	6.1	16.4333	14.8667	22.7667
	$\hat{L}$	0.33217	0.16478	0.11408	0.03026	0.05128
	$\sigma_{\hat{L}}$	0.04124	0.03149	0.00924	0.00425	0.00395
Entropy	R	3.78333	2.90979	6.90676	5.55227	7.60365
	$\overline{d}$	6.86667	5.53333	11.2	9.43333	13.2667
	$\hat{L}$	0.31217	**0.15669**	0.09863	0.02982	0.03997
	$\sigma_{\hat{L}}$	0.02087	0.02625	0.00681	0.00340	0.00355
Gini	R	3.64478	3.32119	7.62978	6.04423	8.74615
	$\overline{d}$	6.23333	6.2	12.9	8.73333	15.5667
	$\hat{L}$	0.30435	0.16309	0.10641	**0.02976**	0.04200
	$\sigma_{\hat{L}}$	0.02589	0.02673	0.00750	0.00339	0.00322
R-norm (1/2)	R	4.77232	3.43399	6.74966	4.58786	7.07989
	$\overline{d}$	8.86667	6.53333	10.9667	10.7	13.7
	$\hat{L}$	0.31195	0.15838	**0.09358**	0.03050	**0.03733**
	$\sigma_{\hat{L}}$	0.03361	0.02688	0.00603	0.00327	0.00394
R-norm (2)	R	3.31	3.12411	7.92363	7.4493	9.01268
	$\overline{d}$	5.53333	5.43333	13.4	9.76667	15.6667
	$\hat{L}$	**0.30348**	0.15763	0.10452	0.03104	0.04447
	$\sigma_{\hat{L}}$	0.03444	0.03009	0.00704	0.00339	0.00342
Twoing	R	3.65261	3.18305	7.26314	6.46513	8.10721
	$\overline{d}$	6.3	5.6	11.5333	9.2	13.6
	$\hat{L}$	0.30435	0.15932	0.10392	0.03037	0.04350
	$\sigma_{\hat{L}}$	0.02657	0.02998	0.00636	0.00339	0.00469
k-NN	k	7	7	3	3	1
	$\hat{L}$	0.32580	*0.13559*	*0.01372*	0.04420	*0.00694*
	$\sigma_{\hat{L}}$	0.3181	0.02227	0.00234	0.00433	0.00114

Table 10: Influence of splitting criteria. (continued)

		data set				
criterion		**PI**	**SA**	**SE**	**SO**	**TH**
Bayes error	R	3.94557	13.4144	5.79582	2.47585	3.2547
	$\overline{d}$	6.13333	26.4667	11.93333	3.16667	4.33333
	$\hat{L}$	0.25013	0.14236	0.04593	0.26135	0.07230
	$\sigma_{\hat{L}}$	0.02164	0.00850	0.00833	0.04900	0.02715
Entropy	R	3.26068	5.91273	5.01742	2.05421	2.45587
	$\overline{d}$	7.13333	13.2333	12.6	3	3.23333
	$\hat{L}$	0.23971	0.13097	0.03965	0.25652	**0.06291**
	$\sigma_{\hat{L}}$	0.02375	0.00736	0.00662	0.05200	0.02377
Gini	R	3.00319	6.52182	5.55398	2.92441	2.21643
	$\overline{d}$	6.26667	13.3333	13.6667	2.93333	3.86667
	$\hat{L}$	0.23802	0.13212	0.04242	0.25894	0.07371
	$\sigma_{\hat{L}}$	0.02175	0.00951	0.00687	0.04715	0.02845
R-norm (1/2)	R	3.59062	5.81943	4.87383	2.91812	2.3784
	$\overline{d}$	9.53333	14.3333	12.5333	5.03333	3.63333
	$\hat{L}$	0.24245	**0.13051**	**0.03779**	**0.25266**	0.06338
	$\sigma_{\hat{L}}$	0.02050	0.00916	0.00674	0.04639	0.02460
R-norm (2)	R	3.00319	7.84027	5.62714	2.17657	3.05892
	$\overline{d}$	6.13333	14.1667	13.2667	2.93333	4
	$\hat{L}$	0.24154	0.13596	0.04238	0.25411	0.07136
	$\sigma_{\hat{L}}$	0.01953	0.00851	0.00697	0.04904	0.02792
Twoing	R	2.99687	6.27682	5.269	2.21643	2.96502
	$\overline{d}$	6.2	12.6667	13.3	2.93333	3.9
	$\hat{L}$	**0.23802**	0.13330	0.04251	0.25894	0.07230
	$\sigma_{\hat{L}}$	0.02175	0.00848	0.00900	0.04715	0.02690
k-NN	k	13	3	1	1	1
	$\hat{L}$	0.26042	*0.09364*	0.04251	*0.18986*	*0.05916*
	$\sigma_{\hat{L}}$	0.02752	0.00575	0.00578	0.04272	0.02873

Table 11: Influence of splitting criteria. (continued)

		data set				
criterion		**VE**	**VO**	**WA**	**WD**	**WI**
Bayes error	R	6.88653	8.83677	7.16342	2.31746	2.75424
	$\overline{d}$	12.0333	15.8333	10.3333	3.2	3.46667
	$\hat{L}$	0.3	0.23566	0.228	0.06720	0.08079
	$\sigma_{\hat{L}}$	0.02624	0.02598	0.01422	0.01823	0.04051
Entropy	R	5.81283	6.94818	5.79432	1.99471	2.36525
	$\overline{d}$	13.2333	11.4	12.1667	3.96667	3
	$\hat{L}$	0.27400	**0.21919**	0.21727	0.05908	0.06328
	$\sigma_{\hat{L}}$	0.03388	0.00242	0.01117	0.01623	0.03302
Gini	R	5.5347	7.81151	5.7013	2.39533	2.44605
	$\overline{d}$	10.5	13.3667	9.73333	3.63333	3.16667
	$\hat{L}$	0.28664	0.22333	0.2181	0.05979	0.08418
	$\sigma_{\hat{L}}$	0.02167	0.02358	0.01085	0.01570	0.04231
R-norm (1/2)	R	6.42553	6.91591	6.66717	2.23501	2.33051
	$\overline{d}$	15.3	10.7	15.9333	5.86667	2.9
	$\hat{L}$	**0.26667**	0.22697	0.22443	**0.05750**	**0.05311**
	$\sigma_{\hat{L}}$	0.02631	0.02473	0.01059	0.01921	0.02132
R-norm (2)	R	5.58889	7.70338	6.14303	2.57628	2.42345
	$\overline{d}$	9.86667	13.1333	9.63333	3.86667	3.13333
	$\hat{L}$	0.29397	0.22838	**0.21563**	0.06085	0.08475
	$\sigma_{\hat{L}}$	0.02328	0.02592	0.00981	0.01532	0.03501
Twoing	R	5.92683	7.09374	5.6802	2.48757	2.24322
	$\overline{d}$	11.1333	11.9667	9.83333	3.8	3.03333
	$\hat{L}$	0.28227	0.22758	0.21697	0.05908	0.08136
	$\sigma_{\hat{L}}$	0.01685	0.02618	0.01127	0.01516	0.04514
k-NN	k	3	1	19	9	1
	$\hat{L}$	0.35390	*0.02596*	*0.14337*	0.06631	0.26215
	$\sigma_{\hat{L}}$	0.02406	0.01060	0.01047	0.01666	0.04946

Table 12: Influence of splitting criteria. (continued)

Regarding the k-NN classifier, experiments show the importance of data structure in $\mathbb{R}^d$. When data is highly clustered, k-NN achieves the best results, even closer to the L^* bound (as in the *waveform* data set, for example). It has also good performance for small data sets with a high dimensionality where input training vectors are separated one from each other. On the other hand, for some data sets the k-NN classifier does not provide a good accuracy, or it needs a large k which is computationally expensive, causing also overfitting. The k-NN classifier is unfeasible for very large data sets, such as the hyperspectral image defined in Sect. 4.1.2, for example. Nevertheless, a k-NN classifier may be used in a cascading framework as the last classifier, trying to extract any information present in input data not captured by the previous classifiers, reducing also k-NN cost because only a fraction of data is used; such an ensemble is described in (Kaynak and Alpaydin, 2000).

4.3 Limited training

In this experiment we test tree performance under a maximum depth constraint. Complete decision trees of maximum depth $\overline{d} \in \{2, 3, 4, 5, 6\}$ are grown and then pruned back. Only the best results (in classification accuracy) are shown, and ties are broken in favor of the criterion yielding the smallest tree. We are interested not only on the performance but also on the splitting criterion achieving the best performance. As we are trying to know the splitting criterion which minimizes $\hat{L}$, we do not show results for $\overline{d} = 1$, because what the Bayes error criterion exactly does is to minimize classification error. We do not include the Twoing criterion, as it cannot be expressed or approximated as an impurity based criteria. Tabs. 13 to 17 show the results for each maximum training depth, respectively.

Notice that for some data sets, $R = \overline{d}$, so no pruning is done. This is a clear sign that the training tree has been stopped too early and pruning is ineffective. On the other hand, $R < \overline{d}$ for a few cases, showing that grown trees were deep enough to contain good pruned subtrees. As expected, $\hat{L}$ decreases as $\overline{d}$ increases, showing an exponential decay when the number of leaves (which is directly related to $\overline{d}$) is large enough to represent all classes present in the training and corpus sets. Only in a few cases $\hat{L}$ shows a bizarre behavior, although it is not significant under a statistical basis, and it is directly related to the experiments described in Sect. 4.4, because it can be explained as a reduction of variance for a given depth.

Nevertheless, the most remarkable fact is that the best splitting criterion changes with the maximum depth allowed. For very small trees, the Bayes error criterion and the R-norm ($R = 2$) criterion yield the best results. As stated in (Brodley, 1995), it is interesting to use splitting criteria which minimize misclassification error near the leaves, that is, it is interesting to be greedy when we are close to take a final decision, whilst it is better to use splitting criteria which minimize tree impurity near the root of the decision tree, in order to avoid overfitting. Our experiments show also this behavior, as depicted in Fig. 22, where splitting criteria are sorted according to its distance to

data set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	**criterion**
covtype	2	2	0.51694	0.00671	Entropy
glass	2	2	0.36526	0.04892	Bayes error
ionosphere	1.78419	2	0.10228	0.02511	R-norm (2)
iris	1.66667	2	0.06067	0.02555	Entropy
letter	2	2	0.86337	0.00301	Bayes error
liver	1.53333	2	0.33884	0.04345	R-norm (2)
lrs	2	2	0.23729	0.02862	Bayes error
opdigits	2	2	0.65227	0.01089	R-norm (2)
page	2	2	0.04781	0.00387	Bayes error
pendigits	2	2	0.61795	0.00763	R-norm (2)
pima	1.40436	1.43333	0.25703	0.02705	Bayes error
sat	2	2	0.35666	0.01741	R-norm (1/2)
segmentation	2	2	0.44338	0.01562	R-norm (2)
sonar	1.30483	1.43333	0.28019	0.04869	R-norm (2)
thyroid	1.88732	2	0.09202	0.02666	Bayes error
vehicle	2	2	0.43416	0.03585	Bayes error
vowel	2	2	0.72768	0.02681	Bayes error
waveform	2	2	0.29947	0.01085	Bayes error
wdbc	1.67011	1.93333	0.07090	0.01462	R-norm (2)
wine	1.97034	2	0.08023	0.02549	R-norm (1/2)

Table 13: Training with a limited maximum depth, $\overline{d} \leq 2$.

data set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	**criterion**
covtype	2.9919	3	0.40305	0.00617	Entropy
glass	2.96596	3	0.33380	0.05910	Bayes error
ionosphere	2.07308	2.4	0.09630	0.02010	Gini
iris	1.806	2.43333	0.04467	0.02172	R-norm (2)
letter	3	3	0.76351	0.00362	Entropy
liver	1.85014	2.56667	0.32522	0.03395	R-norm (2)
lrs	2.32429	3	0.17703	0.03295	R-norm (2)
opdigits	3	3	0.43401	0.01768	Entropy
page	2.99888	3	0.04031	0.00419	R-norm (2)
pendigits	3	3	0.39915	0.00952	Entropy
pima	1.34264	1.93333	0.25508	0.02653	Gini
sat	2.90635	3	0.21152	0.00810	Bayes error
segmentation	2.84496	3	0.17476	0.02360	R-norm (1/2)
sonar	1.78985	2.1	0.26522	0.04809	Entropy
thyroid	2.23192	2.6	0.06761	0.02657	Entropy
vehicle	2.75886	3	0.34705	0.03047	R-norm (1/2)
vowel	3	3	0.61475	0.03166	Bayes error
waveform	2.88348	3	0.26653	0.01492	R-norm (2)
wdbc	1.92275	2.96667	0.06473	0.01666	Entropy
wine	2.30932	2.8	0.05254	0.02289	R-norm (1/2)

Table 14: Training with a limited maximum depth, $\overline{d} \leq 3$.

data set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	**criterion**
covtype	3.74514	4	0.35114	0.00588	R-norm (2)
glass	3.52958	3.96667	0.29953	0.05586	Bayes error
ionosphere	2.15798	2.56667	0.09345	0.02195	Entropy
iris	1.822	2.6	0.04933	0.01982	Gini
letter	4	4	0.64884	0.00536	Entropy
liver	2.25058	3.16667	0.31797	0.03092	R-norm (2)
lrs	2.54699	3.96667	0.16234	0.02775	Entropy
opdigits	3.99031	4	0.27770	0.02233	R-norm (2)
page	3.02421	4	0.03533	0.00408	Entropy
pendigits	3.97952	4	0.22737	0.01387	R-norm (2)
pima	2.00254	2.9	0.24779	0.02193	Gini
sat	3.79877	4	0.18382	0.00854	Bayes error
segmentation	3.41846	4	0.09818	0.0112	R-norm (2)
sonar	2.0186	2.56667	0.25894	0.04788	Entropy
thyroid	2.44742	3.13333	0.06338	0.02513	Entropy
vehicle	3.55248	3.96667	0.30934	0.03266	Entropy
vowel	3.95874	4	0.50343	0.02572	R-norm (2)
waveform	3.5607	3.93333	0.24287	0.01089	R-norm (2)
wdbc	1.93774	3.4	0.06085	0.01463	Entropy
wine	2.32599	2.9	0.05311	0.02132	R-norm (1/2)

Table 15: Training with a limited maximum depth, $\overline{d} \leq 4$.

data set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	**criterion**
covtype	4.87383	5	0.32899	0.00736	Bayes error
glass	3.91596	4.83333	0.29390	0.05689	Bayes error
ionosphere	2.53704	3.16667	0.08889	0.02300	R-norm (2)
iris	1.866	2.83333	0.04867	0.02109	R-norm (1/2)
letter	4.8923	5	0.49782	0.00897	Entropy
liver	2.8271	3.86667	0.31159	0.02971	Gini
lrs	2.8807	4.66667	0.15650	0.02655	Entropy
opdigits	4.94136	5	0.19149	0.01368	R-norm (2)
page	4.15554	4.96667	0.03207	0.00350	R-norm (2)
pendigits	4.91714	5	0.14479	0.00629	R-norm (2)
pima	2.57292	3.96667	0.24258	0.02560	Gini
sat	4.55824	5	0.16123	0.00878	R-norm (2)
segmentation	3.77903	5	0.07126	0.00933	R-norm (2)
sonar	2.17681	3.16667	0.25749	0.04802	Entropy
thyroid	2.42981	3.16667	0.06338	0.02513	Entropy
vehicle	4.10396	4.96667	0.28865	0.02731	R-norm (1/2)
vowel	4.82177	5	0.41657	0.03137	R-norm (2)
waveform	4.24008	4.93333	0.22783	0.01277	R-norm (2)
wdbc	1.95917	3.8	0.05979	0.01617	Entropy
wine	2.32599	2.9	0.05311	0.02132	R-norm (1/2)

Table 16: Training with a limited maximum depth, $\overline{d} \leq 5$.

data set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	**criterion**
covtype	5.88324	6	0.30527	0.00446	Bayes error
glass	4.00751	5.5	0.28686	0.05289	Bayes error
ionosphere	2.53832	3.56667	0.08718	0.02360	Entropy
iris	1.918	3.23333	0.046	0.02010	R-norm (1/2)
letter	5.89922	6	0.40699	0.00885	Entropy
liver	3.14551	4.5	0.30522	0.02956	Gini
lrs	2.99652	5.16667	0.15424	0.02442	Entropy
opdigits	5.81313	6	0.14693	0.01066	R-norm (2)
page	4.75131	5.96667	0.03041	0.00357	Gini
pendigits	5.69666	6	0.09973	0.00917	Entropy
pima	2.67682	4.4	0.23958	0.02400	Gini
sat	5.15392	6	0.14943	0.00812	R-norm (2)
segmentation	3.95342	5.96667	0.05831	0.00790	R-norm (1/2)
sonar	2.85531	4.4	0.25266	0.04609	R-norm (1/2)
thyroid	2.4554	3.2	0.06244	0.02320	Entropy
vehicle	4.52264	5.9	0.28239	0.02723	R-norm (1/2)
vowel	5.55773	6	0.35152	0.03185	Entropy
waveform	5.01017	5.93333	0.22167	0.01120	R-norm (2)
wdbc	2.03589	4.13333	0.05838	0.01703	Entropy
wine	2.32599	2.9	0.05311	0.02132	R-norm (1/2)

Table 17: Training with a limited maximum depth, $\overline{d} \leq 6$.

the Bayes error criterion (see Fig. 3).

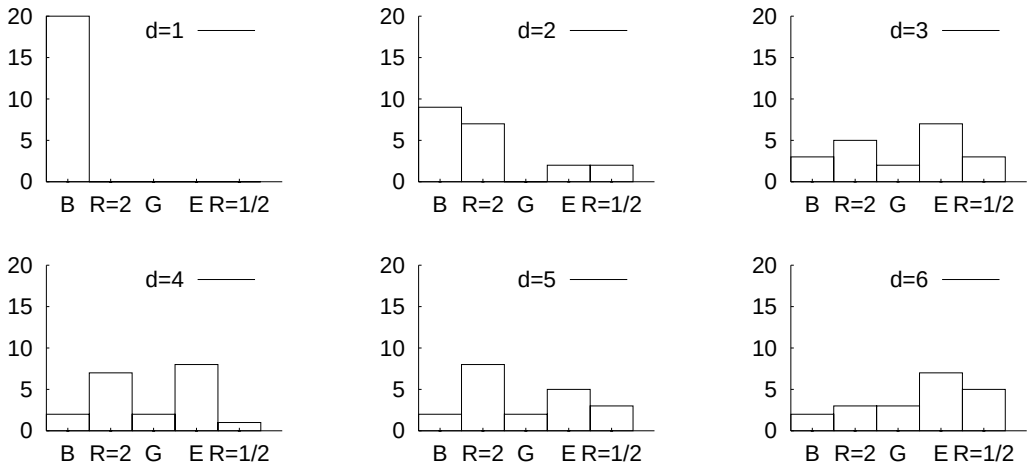

Figure 22: Number of times a splitting criterion is chosen as the best (B=Bayes error, R=R-norm, G=Gini, E=Entropy).

As $\overline{d}$ is allowed to grow, more and more data sets achieve better performance with the Entropy and R-norm ($R = 1/2$) splitting criteria than with the other ones. Notice that all splitting criteria are special cases of the R-norm splitting criterion, but the Gini splitting criterion. This can be approximated using $R \approx 1.61858$, though, so it can be considered another case. Actually, it is easy to see that the Gini criterion is the square of the R-norm criterion using $R = 1/2$, but a constant factor. It is fair to think that a family of splitting criteria such as the R-norm family could be used to achieve the best performance if the optimal value for R was computed. Furthermore, such value could be computed at each node depending on its intrinsic characteristics (including node position in the tree, such as depth, for example), and then use an adaptive scheme for the splitting criterion. This is the approach used in (Brodley, 1995), although in a more unsophisticated way.

4.4 Bias-variance decomposition

In this experiment we compute bias-variance decomposition for decision trees. Our goal is to study bias and variance behavior (see Sect. 3.8.1.3) under several experiment setups in order to extract information that can be useful for our purposes.

We randomly split each data set (all available instances shown in Tab. 7) in two sets: a training set and a test set, following the $(2/3, 1/3)$ rule. Then, the training set is used to generate 25 bootstrap replicates (subsampling with replacement from the training set, see (Efron and Tibshirani, 1993)), and a classifier is built for each bootstrap replicate. Then, performance is measured for each classifier

using the test set. This process is repeated five times, and then all results are averaged. We do not use 100 replicates as described in (Domingos, 2000) because, as stated in (Breiman, 1996a), it is usually (almost) useless to go beyond 25 replicates for subsampling purposes.

As we want to compare classical decision trees to progressive decision trees, first of all we measure the bias-variance decomposition for a single tree grown without any depth limitation and using the best splitting criterion found in the experiment described in Sect. 4.2 (we used the Gini criterion for the *pima* data set).

set	**criterion**	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$
CO	Entropy	9.56141	19.736	0.24422	0.00394	0.17520	0.06902
GL	Gini	4.44458	6.96	0.35617	0.02599	0.29014	0.06603
IO	$R = 2$	2.91573	3.904	0.10789	0.00544	0.07521	0.03268
IR	$R = 1/2$	1.81212	2.64	0.04235	0.00555	0.02353	0.01882
LE	$R = 1/2$	9.74719	17.472	0.16474	0.00264	0.07584	0.08890
LI	$R = 2$	4.69259	7.688	0.36265	0.01326	0.28522	0.07743
LR	Entropy	2.84827	5.368	0.16886	0.00722	0.13672	0.03214
OP	$R = 1/2$	6.56378	10.376	0.11645	0.00352	0.04066	0.07579
PA	Gini	5.39101	7.456	0.03211	0.00216	0.02796	0.00415
PE	$R = 1/2$	6.83542	13.264	0.04873	0.00072	0.01752	0.03121
PI	Gini	2.49658	4.576	0.23666	0.01014	0.20938	0.02728
SA	$R = 1/2$	5.62438	12.808	0.14339	0.00247	0.10326	0.04013
SE	$R = 1/2$	4.31219	10.6	0.04969	0.00632	0.03065	0.01904
SO	$R = 1/2$	2.44432	4	0.28150	0.02132	0.19130	0.09020
TH	Entropy	2.35748	3.008	0.08811	0.00403	0.06667	0.02144
VE	$R = 1/2$	5.63901	12.352	0.29770	0.00760	0.24610	0.05160
VO	Entropy	6.67239	10.4	0.27682	0.00755	0.09454	0.18228
WA	$R = 2$	5.95249	8.944	0.23230	0.00423	0.16783	0.06447
WD	$R = 1/2$	1.97176	4.192	0.06910	0.01750	0.04737	0.02173
WI	$R = 1/2$	2.24136	2.888	0.06374	0.01035	0.02667	0.03707

Table 18: Bias-variance decomposition for a single tree using the best splitting criterion for each data set.

Notice that we are using a different approach to measure generalization error performance, so results shown in Tab. 18 differ from those shown in Sect. 4.2.2. N-fold cross validation always yields lower misclassification error than bootstrapping (but for the *iris* and *pima* data sets), while the latter usually yields a smaller misclassification error variance. On the other hand, bootstrapping produces smaller trees in average.

A remarkable fact is that usually bias is larger than variance, but for a few training sets. A large bias shows that decision trees built using orthogonal hyperplanes are not complex enough to capture

internal data structure, so there is room for improvement using ensemble methods such as boosting, for example. Variance is larger than bias in a few data sets which have a common characteristic: all of them have a large number of classes (except the *wine* data set, which may be not representative), and all classes are equally probable, so a large baseline error is present (see Tab. 7).

4.4.1 Splitting criterion dependence

In the previous experiment we used the best splitting criterion for each data set, which is unknown unless we compare them by building a classifier for each one. As we want to automate our ensemble, we will use entropy as the splitting criterion. Tab. 19 shows the bias-variance decomposition using the entropy criterion, following the same experiment design than the one described above.

set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$
CO	9.56141	19.736	0.24422	0.00394	0.17520	0.06902
GL	3.96833	5.952	0.35797	0.02940	0.28451	0.07346
IO	2.47415	3.48	0.10783	0.00397	0.09744	0.01039
IR	1.80073	2.584	0.04220	0.00675	0.02353	0.01867
LE	9.91032	17.904	0.16702	0.00146	0.07703	0.08999
LI	3.38671	5.744	0.33976	0.01345	0.27826	0.06150
LR	2.84827	5.368	0.16886	0.00722	0.13672	0.03214
OP	6.66539	10.344	0.12137	0.00265	0.04621	0.07516
PA	4.78741	7.84	0.03263	0.00118	0.02675	0.00588
PE	7.30555	12.408	0.05319	0.00079	0.01812	0.03507
PI	2.44464	4.672	0.23950	0.00970	0.21953	0.01997
SA	5.60507	11.36	0.14431	0.00229	0.10382	0.04049
SE	4.37432	10.208	0.04988	0.00594	0.02935	0.02053
SO	2.24472	3.256	0.26736	0.02700	0.19420	0.07316
TH	2.35748	3.008	0.08811	0.00403	0.06667	0.02144
VE	5.2307	10.04	0.29660	0.00874	0.25036	0.04624
VO	6.67239	10.4	0.27682	0.00755	0.09454	0.18228
WA	5.25705	10.136	0.23087	0.00404	0.16204	0.06883
WD	2.10501	3.568	0.06947	0.01441	0.05052	0.01895
WI	2.25573	2.984	0.06827	0.01014	0.02667	0.04160

Table 19: Bias-variance decomposition for a single decision tree using entropy as the splitting criterion.

Notice that $\hat{L}$ is smaller for several data sets (we used a different method for misclassification error estimation), showing the importance of reducing parameter dependence when building complex classification systems. As stated in Sect. 4.2.2, no splitting criterion is superior for all data sets

with statistical significance, so we will choose the entropy splitting criterion for our experiments.

4.4.2 Bias-variance and training maximum depth

As stated in 3.4, the first decision trees in the cascading ensemble should be small in order to avoid overfitting, but we do not have yet a good definition for small. As stated in (Gama and Brazdil, 2000), first classifiers should have a small variance and a (hopefully) moderate bias.

With this experiment we try to obtain any information that would be useful for determining the size of the first decision tree in a cascading ensemble. We study the bias-variance decomposition as a function of the maximum depth allowed during the training process, trying to find any relationship between $\overline{d}$, $\hat{V}$ and $\hat{B}$. We grow a small decision tree with maximum depth $\overline{d}$, we prune it back and then we measure the bias-variance decomposition as usual. Varying $\overline{d}$ we can study the behavior of decision trees in order to determine a good tradeoff for the bias-variance decomposition.

Another interesting fact that can be extracted from this experiment is that both bias and variance show a typical behavior. When $\overline{d}$ is too small, built trees have not enough leaves to represent all classes present in the training set, so bias is high. While $2^{\overline{d}} < K$, bias decreases fast as $\overline{d}$ increases, and when $2^{\overline{d}} \geq K$, bias still decreases but showing an exponential decay. If we use pruning, it is better to use $K+1$ instead of K as some leaves are pruned back causing the tree to have not enough leaves to represent all classes.

On the other hand, when $\overline{d}$ is zero, variance is obviously 0 (the output label is always the most populated class). Then, as $\overline{d}$ increases, variance also increases but it shows a peak around K for several data sets, which is surprising, because it does not seem to follow any criterion, although the peak is usually for values lower than K, meaning than very different decision trees are built, because they do not represent all classes in the training set. This is also a criterion for determining $\overline{d}$, as such high value for variance should be avoided.

Therefore, the bias-variance decomposition clearly shows three different parts: a first part with high bias and low variance, a second part, where bias starts decreasing and variance shows a peak, and a third part, where both bias and variance stabilize. This behavior has also been observed for other classification systems when varying one of parameters which determine the learning rate (Valentini and Dietterich, 2002).

This behavior gives us information about how the first decision tree in a cascading ensemble should be constructed: we know that a low bias is good but a low variance is also important. We need a first decision tree large enough to have a low bias but trying to keep variance low. For most data sets, bias predominates over variance in generalization error, so classification accuracy shows a behavior more similar to bias. Therefore, we should select the minimum depth where variance is still small in comparison with bias, and bias has already an exponential decay behavior.

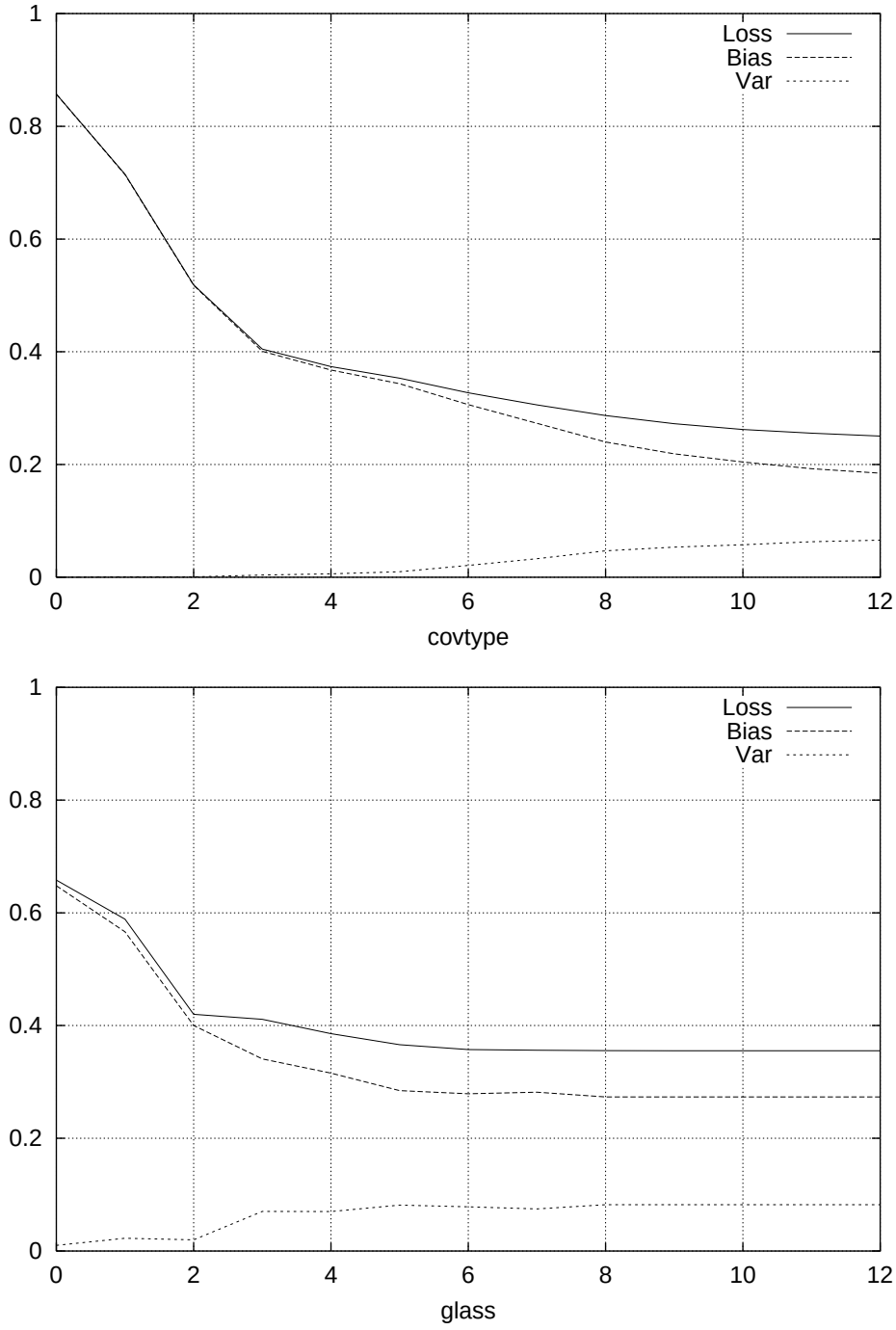

Figure 23: Bias-variance decomposition varying maximum training depth.

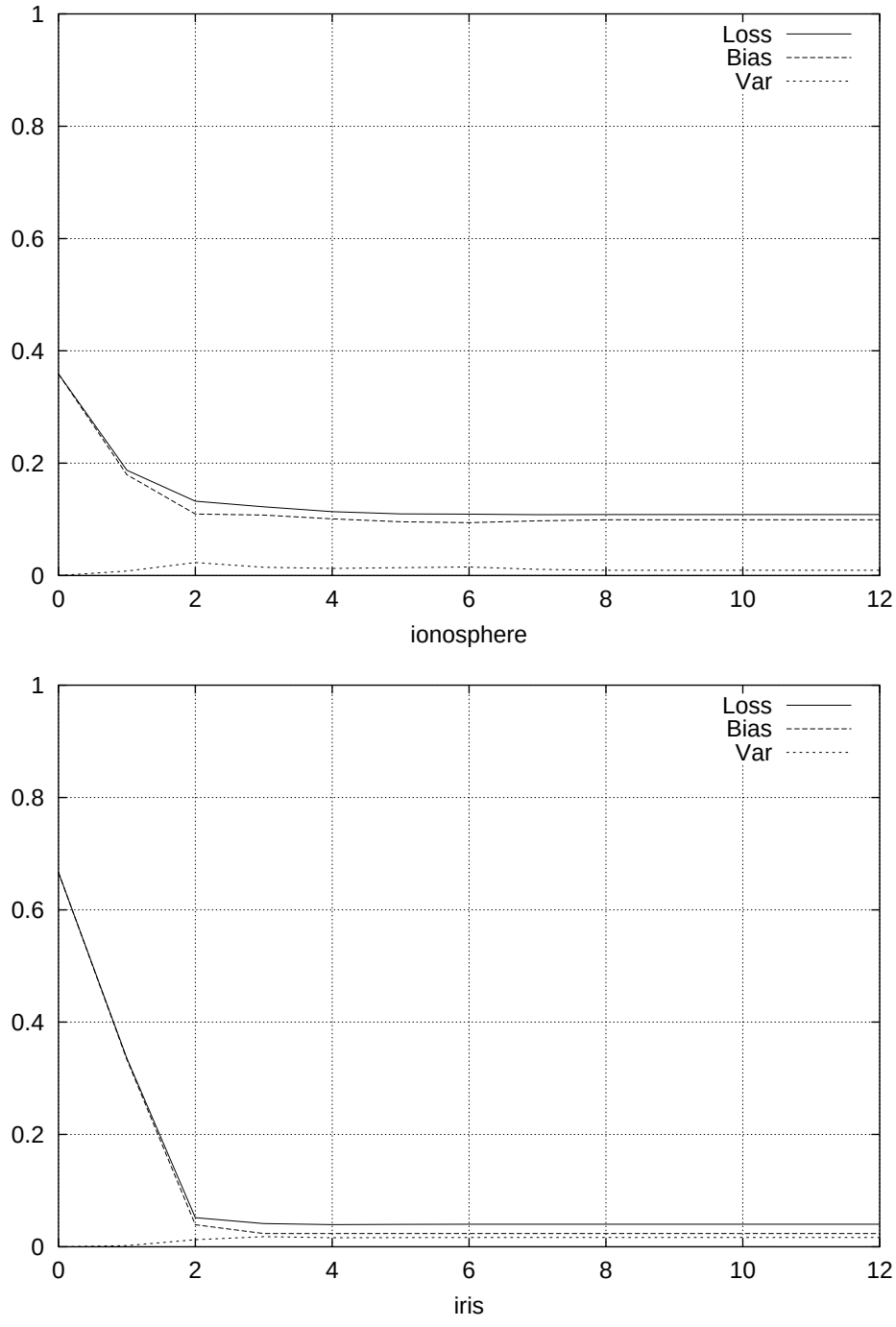

Figure 24: Bias-variance decomposition varying maximum training depth (continued).

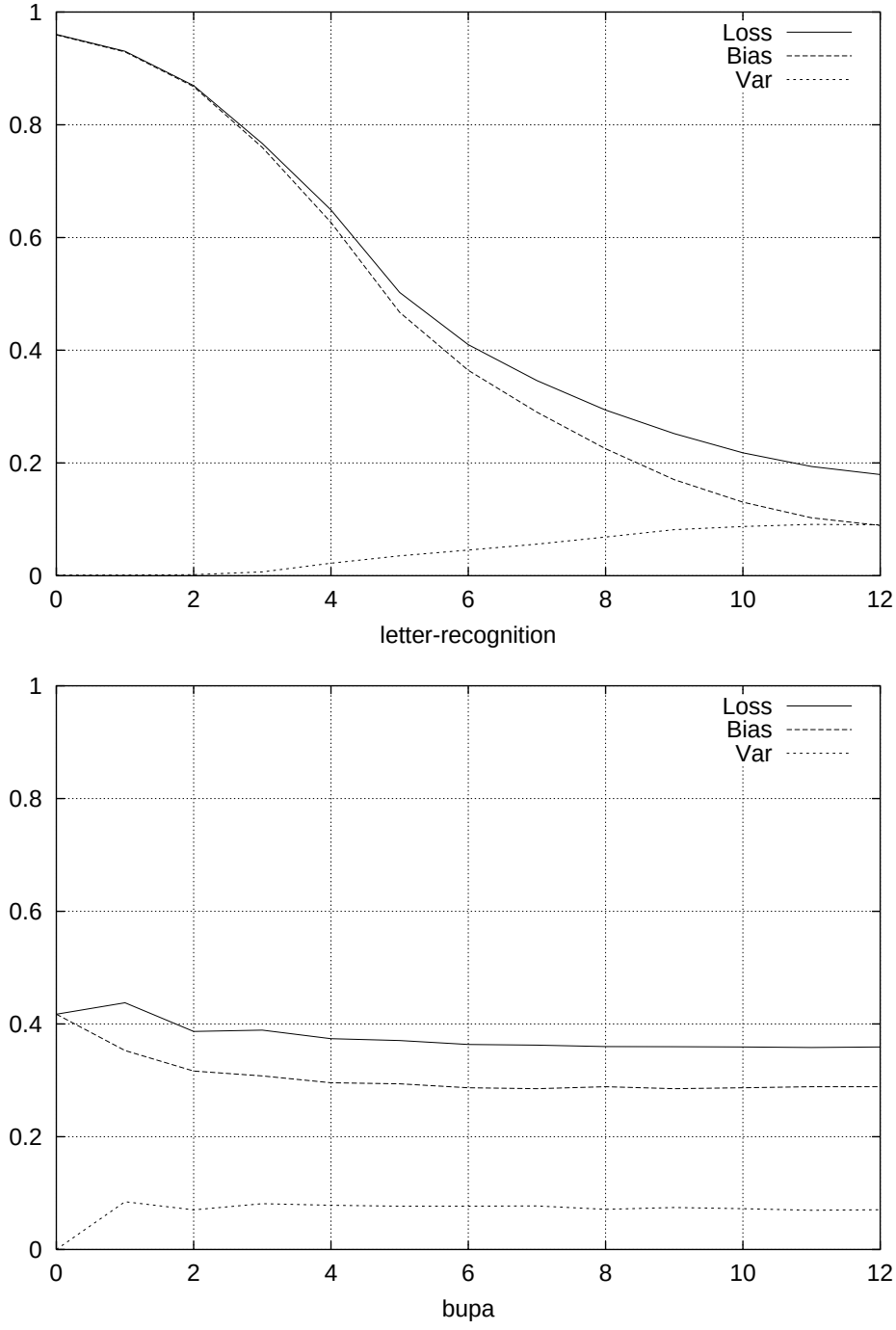

Figure 25: Bias-variance decomposition varying maximum training depth (continued).

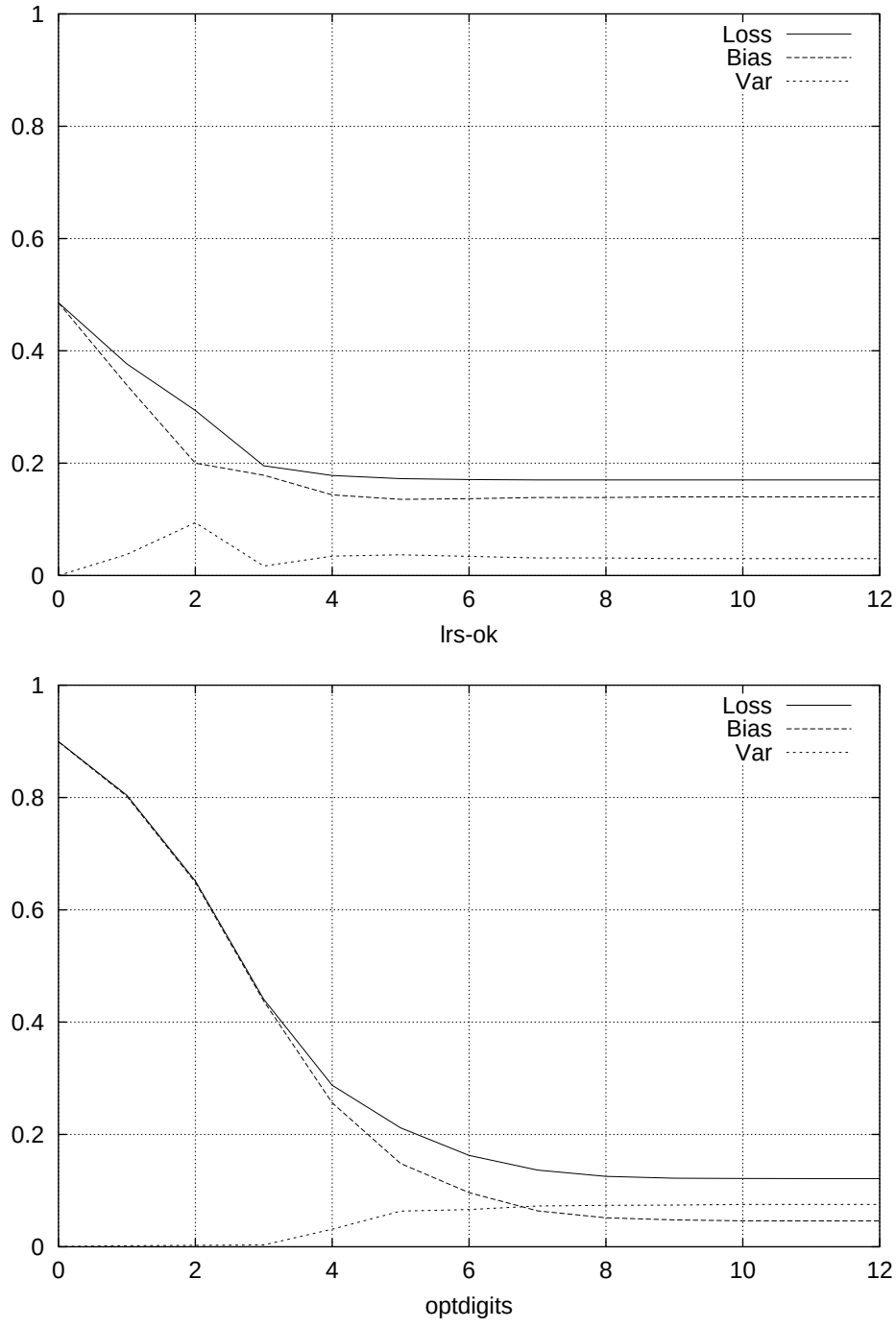

Figure 26: Bias-variance decomposition varying maximum training depth (continued).

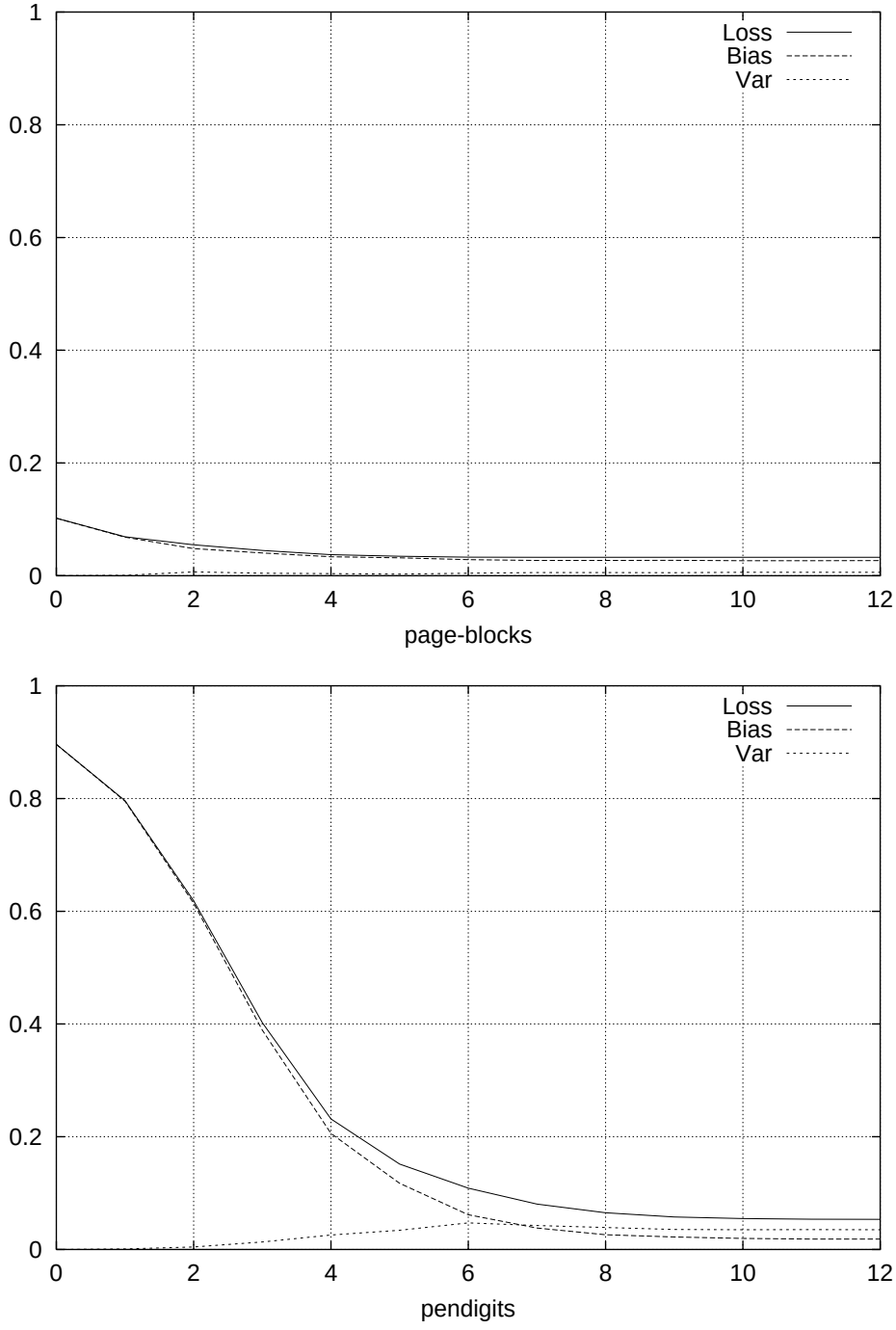

Figure 27: Bias-variance decomposition varying maximum training depth (continued).

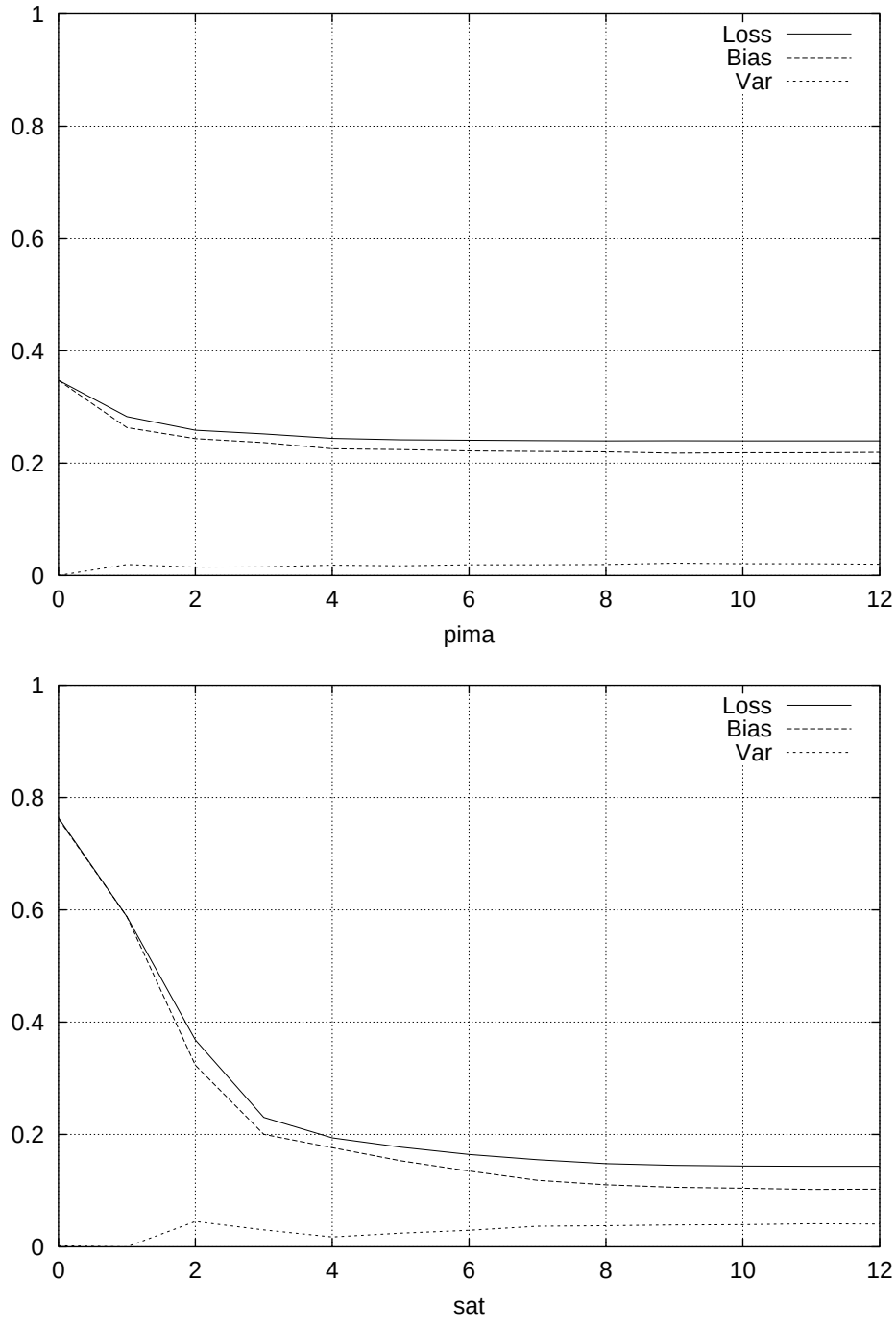

Figure 28: Bias-variance decomposition varying maximum training depth (continued).

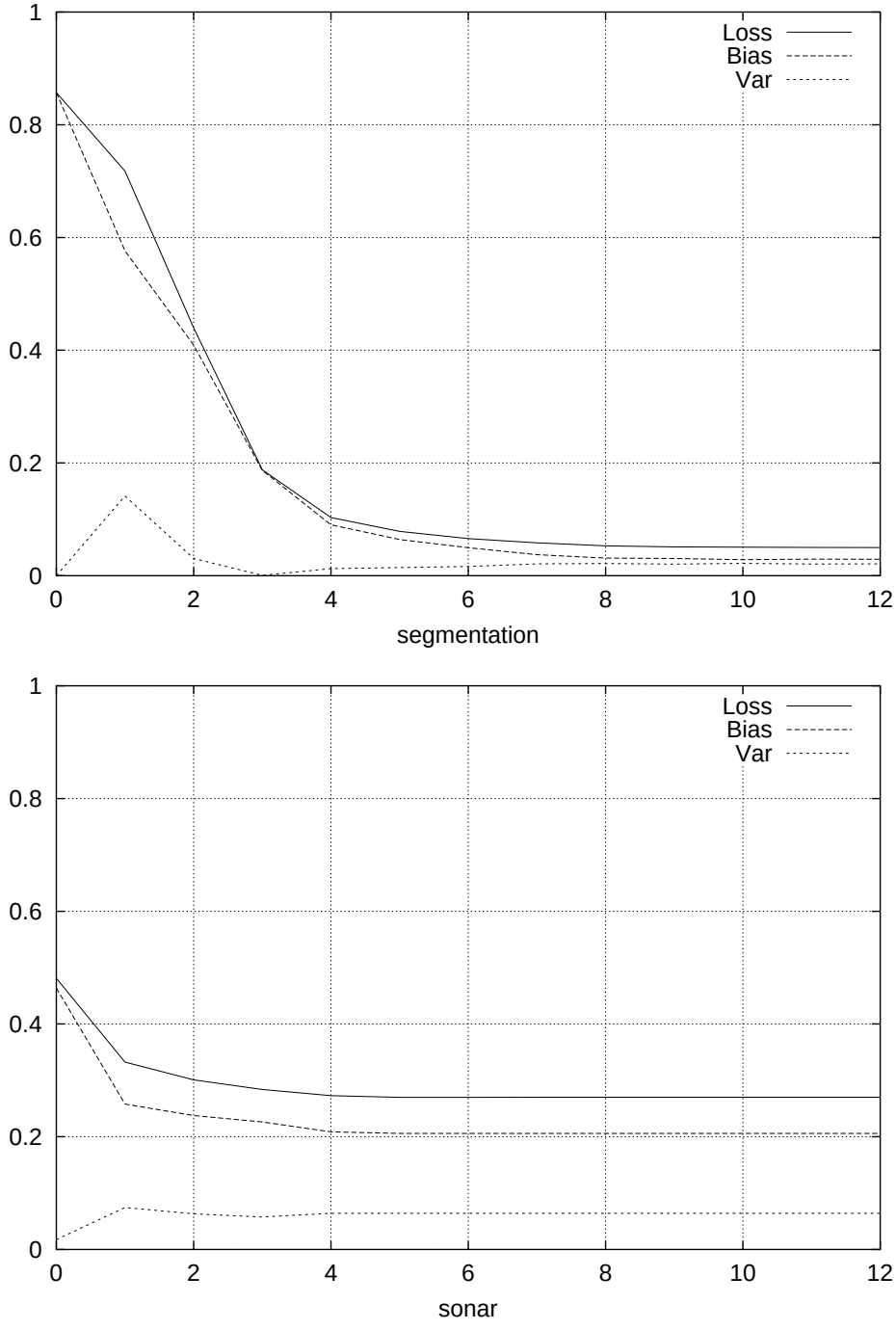

Figure 29: Bias-variance decomposition varying maximum training depth (continued).

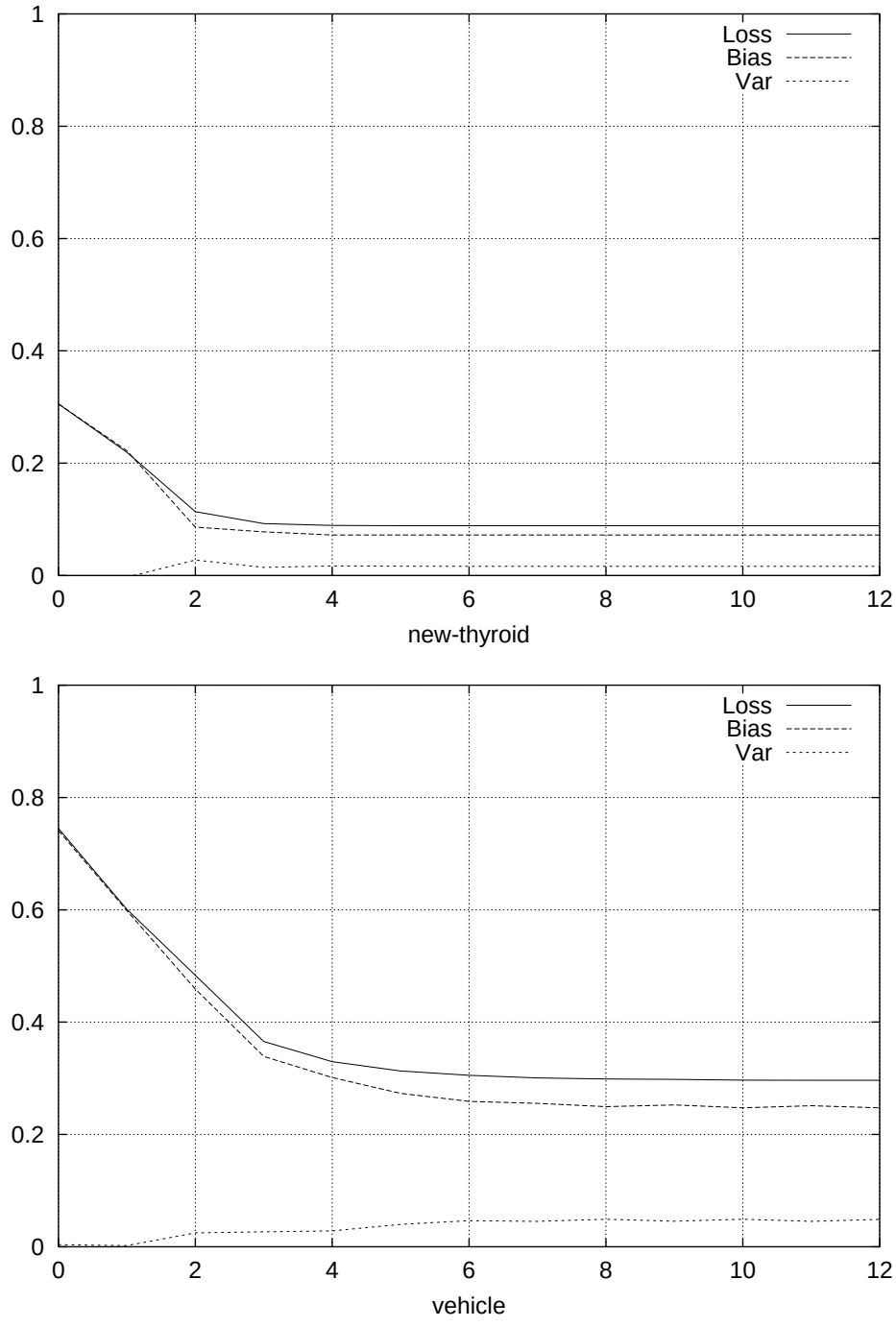

Figure 30: Bias-variance decomposition varying maximum training depth (continued).

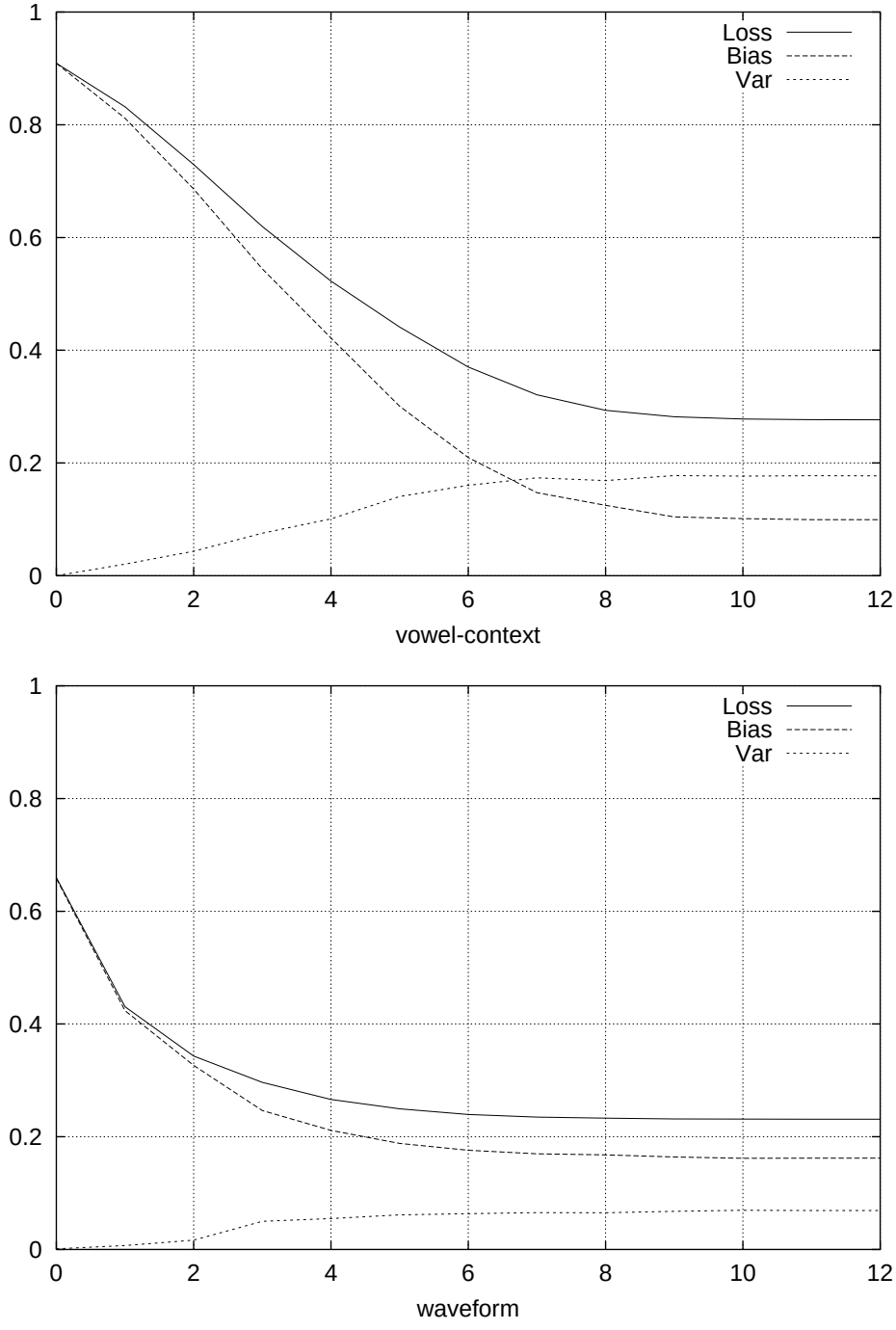

Figure 31: Bias-variance decomposition varying maximum training depth (continued).

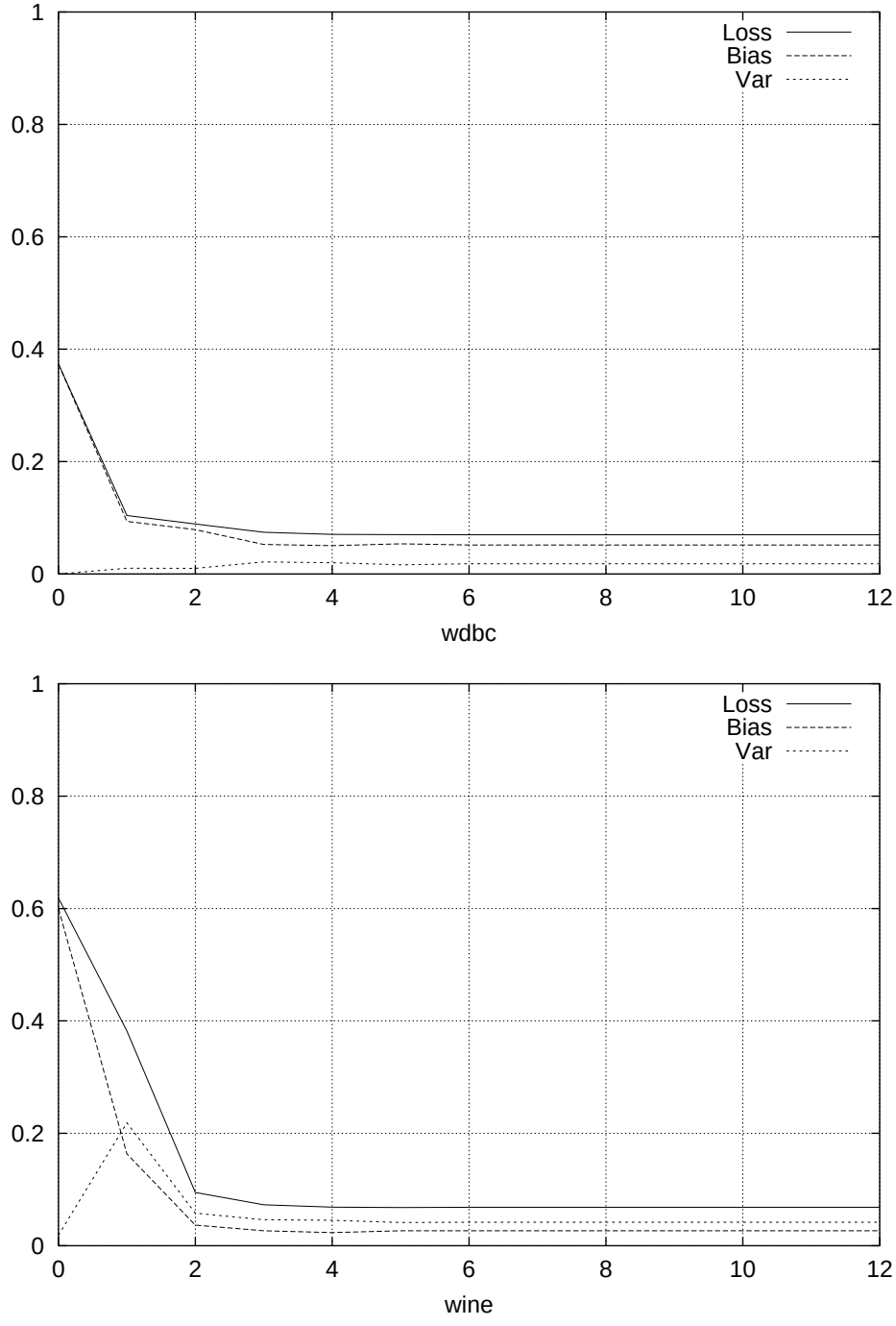

Figure 32: Bias-variance decomposition varying maximum training depth (continued).

4.5 Cascading decision trees

This section uses all the knowledge collected from the precious sections in order to compare the three kinds of cascading ensembles (defined in Sect. 3.7) between them and against a single decision tree (those built in Sect. 4.4). The three kinds of cascading ensembles were:

- Additional information, no mixed class (type B): a first decision tree is used to compute additional information that will be used by a second decision tree, and all vectors from the training set are used in both decision trees.
- No additional information, mixed class (type A): a first decision tree is used to remove those vectors in the training set which form the easiest part of the classification problem, and a second decision tree tries to classify the remaining vectors.
- Additional information, mixed class (type C): a first decision tree is used to remove the easiest cases but it also computes additional information which will be used by the second decision tree to classify the remaining vectors.

4.5.1 Additional information, no mixed class

The first experiment uses only the additional information computed by the first decision tree, which is used by the second decision tree, and no vectors are removed from the training set from one stage to the next. This is the basic approach described in (Gama and Brazdil, 2000).

In our first experiment we only want to measure whether cascading decision trees may be a good tool without any tuning: a first complete decision tree is built as usual, and then a second decision tree is built using the new classification features added by the previous one. Both training and classification costs increase, as two large decision trees must be built and then each sample must be classified twice. The second decision tree uses the $K + 2$ computed variables as new classification features. The basic idea is to see whether decision trees may be used in a cascading ensemble without degrading classification performance in an automated learning procedure.

4.5.1.1 Complete decision trees, best splitting criterion

Tab. 20 shows the performance of two decision trees ensembled using cascading. The average rate and maximum depth for the first decision tree (we will use T_1 to denote such tree) are shown in Tab. 18, as we use a complete decision tree for the first stage. A $+$ sign and a $-$ sign in $\hat{B}$ and $\hat{V}$ denote whether it increases or decreases when compared to T_1 results respectively. A positive $\Delta\hat{L}$ means that performance is degraded, while a negative value means that the cascading ensemble of the two trees outperforms a single tree.

As expected, both R and $\overline{d}$ decrease as the second decision tree uses the additional information provided by the first one. When the second decision tree uses one of the new variables for splitting,

set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$	$\Delta\hat{L}$
CO	7.00981	16.6	0.24148	0.00396	0.17103 −	0.07045 +	−1.12 %
GL	3.51026	6.048	0.35549	0.02707	0.28451 −	0.07098 +	−0.19 %
IO	2.21476	2.88	0.10571	0.00585	0.07350 −	0.03221 −	−2.02 %
IR	1.7867	2.504	0.04078	0.00516	0.02353 =	0.01725 −	−3.71 %
LE	9.01949	18.032	0.16650	0.00252	0.07533 −	0.09117 +	+1.07 %
LI	2.54624	4.096	0.34324	0.01176	0.25739 −	0.08585 +	−5.35 %
LR	2.54635	5	0.16981	0.00796	0.13446 −	0.03535 +	+0.56 %
OP	5.88032	9.952	0.11528	0.00359	0.03853 −	0.07675 +	−1.00 %
PA	3.70114	6.072	0.03203	0.00211	0.02774 −	0.00429 +	−0.25 %
PE	5.78445	11.8	0.04841	0.00069	0.01724 −	0.03117 −	−0.66 %
PI	1.75417	2.936	0.23525	0.00992	0.20781 −	0.02744 +	−0.6 %
SA	4.81591	11.12	0.14295	0.00288	0.10191 −	0.04099 +	−0.31 %
SE	4.01973	8.984	0.04883	0.00558	0.02987 −	0.01896 −	−1.73 %
SO	1.90365	2.968	0.27513	0.0224	0.18841 −	0.08672 −	−2.26 %
TH	2.01274	2.824	0.08378	0.00459	0.06389 −	0.01989 −	−4.91 %
VE	3.98577	8.168	0.29427	0.00753	0.23546 −	0.05881 +	−1.15 %
VO	5.64989	10.088	0.27561	0.00918	0.09636 +	0.17925 −	−0.44 %
WA	5.3315	8.184	0.22693	0.00471	0.16444 −	0.06249 −	−2.31 %
WD	1.60147	3.072	0.06846	0.01693	0.04526 −	0.02320 +	−0.93 %
WI	1.98991	2.496	0.06733	0.01060	0.03333 +	0.03400 −	+5.63 %

Table 20: Bias-variance decomposition for a type B cascading ensemble of two large decision trees. Signs +, =, − and Δ are referred to Tab. 18.

is in fact using a complete branch of the first decision tree from its root to a leaf. This is similar to the fact of using more complex decision functions in each internal node, increasing both tree complexity and boundary representation richness, and thus reducing the number of splits needed to represent such boundary.

Performance is slightly improved but for a few data sets (*letter*, *lrs* and *wine*), mainly because of bias reduction, while variance shows an unpredictable behavior. Nevertheless, performance improvement does not justify the increase of cost because two large decision trees must be built, and the increase of variance for several data sets shows that the first decision tree is too large. The main reason for bias reduction is that orthogonal splits are not rich enough to represent the boundaries defined by the elements of each class.

Using two complete decision trees is therefore a way of building more complex decision boundaries, but it is not the right approach as the problem of reducing misclassification error cannot be solved trying only to reduce bias, as variance must be also taken into account. Furthermore, neither training cost nor classification cost are reduced, so the small increase in classification performance

is not worth enough.

4.5.1.2 Complete decision trees, entropy splitting criterion

In this experiment we compute again the increase in classification performance but using always the entropy splitting criterion, in order to reduce parameter dependence. Tab. 21 shows the obtained results and it must be compared to Tabs. 19 and 20 in order to measure the increase of classification performance and parameter dependence, respectively.

set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$	$\Delta\hat{L}$
CO	7.00981	16.6	0.24148	0.00396	0.17103 −	0.07045 +	−1.12 %
GL	3.43151	5.48	0.35358	0.03097	0.27606 −	0.07752 +	−1.23 %
IO	1.93934	2.6	0.10400	0.00477	0.08889 −	0.01511 +	−3.55 %
IR	1.76917	2.44	0.04220	0.00714	0.02353 =	0.01867 =	0.0 %
LE	9.38852	22.8	0.16596	0.00123	0.07605 −	0.08991 −	−0.63 %
LI	2.47109	4.056	0.33384	0.01579	0.25913 −	0.07471 +	−1.74 %
LR	2.54635	5	0.16981	0.00796	0.13446 −	0.03535 +	+0.56 %
OP	6.05888	10.336	0.11924	0.00332	0.04247 −	0.07677 +	−1.75 %
PA	3.37098	6.696	0.03247	0.00097	0.02588 −	0.00659 +	−0.49 %
PE	6.23174	11.712	0.05215	0.00061	0.01708 −	0.03507 =	−1.96 %
PI	1.77048	3.056	0.23778	0.01003	0.21719 −	0.02059 +	−0.72 %
SA	4.74283	10.248	0.14300	0.00207	0.10242 −	0.04058 +	−0.91 %
SE	4.12135	8.896	0.04858	0.00564	0.02623 −	0.02235 +	−2.61 %
SO	1.73432	2.384	0.26272	0.02816	0.17101 −	0.09171 +	−1.74 %
TH	2.01274	2.824	0.08378	0.00459	0.06389 −	0.01989 −	−4.91 %
VE	3.71641	6.872	0.29367	0.01001	0.24468 −	0.04899 +	−0.99 %
VO	5.64989	10.088	0.27561	0.00918	0.09636 +	0.17925 −	−0.44 %
WA	4.7224	9.224	0.22509	0.00472	0.15365 −	0.07144 +	−2.50 %
WD	1.63829	2.544	0.06905	0.01460	0.04947 −	0.01958 +	−0.60 %
WI	1.99643	2.6	0.07280	0.00810	0.02333 −	0.04947 +	+6.64 %

Table 21: Bias-variance decomposition for a type B cascading ensemble of two large decision trees using the entropy splitting criterion. Signs +, =, − and Δ are referred to Tab. 19.

Notice that in this case, the bias-variance decomposition follows the predicted behaviour: misclassification error reduction is mainly caused by bias reduction, while variance usually increases (in 17 out of 20 data sets).

As we want to make fair comparisons across all data sets, from now and when we will use the entropy splitting criterion in order to minimize the dependence of classification performance on such parameter for the rest of the experiments.

4.5.1.3 Cascading three large decision trees

This experiment shows that using three large decision trees improves classification performance, mainly due to bias reduction. Tab. 22 shows the results obtained for this experiment. Final tree becomes more and more complex because uses variables from the previous two stages, which can be considered as complex boundaries. Furthermore, classification accuracy also improves when compared to a ensemble of two large decision trees and, therefore, there is possibly room for improving classification performance if four or more decision trees were used, although classification cost increases accordingly, and accuracy gain is not worth.

set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$	$\Delta\hat{L}$
CO	4.61748	12.968	0.23999	0.00399	0.16825 $--$	0.07174 $++$	-1.73 % $\downarrow$
GL	2.52004	4.416	0.35651	0.03069	0.27606 $-=$	0.08045 $++$	-0.41 % $\uparrow$
IO	1.13569	1.352	0.10202	0.00453	0.08889 $-=$	0.01313 $+-$	-5.39 % $\downarrow$
IR	1.64468	2.016	0.04424	0.00654	0.02353 $==$	0.02071 $++$	$+4.83$ % $\uparrow$
LE	7.71255	24.6077	0.16541	0.00119	0.07446 $--$	0.09095 $++$	-0.96 % $\downarrow$
LI	1.35183	1.856	0.33002	0.01554	0.24522 $--$	0.08480 $++$	-2.87 % $\downarrow$
LR	2.16638	4.416	0.16994	0.00853	0.12881 $--$	0.04113 $++$	$+0.64$ % $\uparrow$
OP	4.67795	9.184	0.11776	0.00348	0.03970 $--$	0.07806 $++$	-2.97 % $\downarrow$
PA	1.44619	4.624	0.03244	0.00072	0.02654 $-+$	0.00590 $+-$	-0.58 % $\downarrow$
PE	4.49384	9.32	0.05152	0.00068	0.01653 $--$	0.03499 $+-$	-3.14 % $\downarrow$
PI	1.20226	1.784	0.23644	0.00999	0.21016 $--$	0.02628 $++$	-1.28 % $\downarrow$
SA	3.69416	8.528	0.14083	0.00201	0.09795 $--$	0.04288 $++$	-2.41 % $\downarrow$
SE	3.38143	5.632	0.04808	0.00552	0.02467 $--$	0.02341 $++$	-3.61 % $\downarrow$
SO	1.14815	1.32	0.25867	0.02831	0.16232 $--$	0.09635 $++$	-3.25 % $\downarrow$
TH	1.3465	2.072	0.08222	0.00399	0.06111 $--$	0.02111 $-+$	-6.68 % $\downarrow$
VE	2.76416	4.864	0.28868	0.00999	0.23546 $--$	0.05322 $++$	-2.67 % $\downarrow$
VO	4.51533	8.688	0.27523	0.00866	0.09697 $++$	0.17826 $--$	-0.57 % $\downarrow$
WA	3.21346	6.912	0.21877	0.00415	0.14026 $--$	0.07851 $++$	-5.24 % $\downarrow$
WD	1.10493	1.264	0.06825	0.01424	0.04947 $-=$	0.01878 $--$	-1.76 % $\downarrow$
WI	1.65699	2.072	0.07267	0.00764	0.03334 $++$	0.03933 $--$	$+6.44$ % $\downarrow$

Table 22: Bias-variance decomposition for a type B cascading ensemble of three large decision trees using the entropy splitting criterion. Double signs $+$, $=$, $-$ and Δ are referred to Tabs. 19 and 21 respectively, and $\uparrow$ and $\downarrow$ are referred to Tab. 21 (they mean worse and better respectively).

Notice that both bias and variance show the same behavior when cascading the first decision tree with the second one than when cascading the second decision tree with the third one. Accuracy improvement is mainly due to bias reduction, while variance still increases, although it shows a more bizarre behavior. This may indicate that the third decision tree is useful for building more complex

boundaries, thus reducing bias, but it also increases unstability. Therefore, we discard testing more than three decision trees in a cascading ensemble, in order to avoid the construction of too unstable classifiers. This experiment also shows that orthogonal splits are not a good tool for representing complex boundaries.

4.5.1.4 Limited depth decision trees

As stated previously, under a bias-variance decomposition approach, T_1 should be a small tree, in order to reduce variance. But if we use orthogonal splits (which do not represent complex boundaries very well), T_1 should not be too small in order to produce reasonably accurate predictions. As T_2 uses the additional information provided by T_1, this should be large enough to produce "good" additional information. If not, we will be cheating our second tree and performance could be compromised.

As we use $\{p(j|t)\}$ as additional information, we must ensure the first decision tree has enough leaves to represent all classes. Therefore, we should discard using trees with maximum depth $\overline{d} < \log_2(K)$ where K is the number of classes, so we might take $\overline{d} = \lceil \log_2(K) \rceil$ as the minimum depth for the first decision tree. But if the first decision tree is pruned back, we cannot ensure to have enough leaves to represent all classes using $\overline{d} = \lceil \log_2(K) \rceil$, so we should also repeat the same experiment using $\overline{d} = \lceil \log_2(K) \rceil + 1$. Tab. 23 shows the better classification accuracy found when varying $\overline{d}$ from 0 to 12. Ties are broken in favor of the smallest decision tree (that is, the minimum $\overline{d}$).

A $\downarrow$ sign in Tab. 23 means that it is possible to tune the maximum depth of the first decision tree in the type B cascading ensemble to improve classification accuracy with respect to the type B cascading ensemble of two large decision trees, reducing also classification cost. This happens in 14 out of 20 data sets, with maximum depths in range $[0, 12]$, which are relatively small decision trees with a limited computational training cost. In all cases but two (the *iris* and *sat* data sets) classification accuracy is improved by reducing bias.

But, is $\overline{d}$ in Tab. 23 related to any criterion which can be computed or estimated from the data set? Tab. 24 compares the optimal $\overline{d}$ to several measures that can be computed from the original data set which is used to build the classification system: the minimum depth to ensure the decision tree has enough leaves to represent all classes, the entropy of the estimated class distribution in the training set, the maximum length of a Huffman code (Cover and Thomas, 1991) constructed for coding the classes present in the training set, and the average length of a single decision tree and its maximum depth (R and $\overline{d}$ in Tab. 19, respectively).

All of these measures have an information theory based meaning, as they compute the size of the decision tree for a given algorithm. Nevertheless, such algorithms are not directly related to the decision tree training algorithm, because the training process is not devoted to not separate classes at each stage perfectly. Furthermore, with only 20 data sets it would be very dangerous to infer any criterion, and for most data sets, the best $\overline{d}$ was not statistically significant in front of the second or even the third best maximum depth. Therefore, from a practical point of view, if we want the

set	$\overline{d}_1$	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$	$\Delta\hat{L}$
CO	12	7.3499	18.448	0.24334	0.00373	0.17413 −	0.06921 +	−0.36 % ↑
GL	5	3.60975	5.68	0.35279	0.02997	0.27042 −	0.08237 +	−1.48 % ↓
IO	6	1.97976	2.688	0.10373	0.00473	0.08889 −	0.01484 +	−3.80 % ↓
IR	3	1.79392	2.552	0.04188	0.00737	0.02353 =	0.01835 −	−0.76 % ↓
LE	3	9.98227	17.944	0.16623	0.00144	0.07419 −	0.09234 +	−0.47 % ↑
LI	8	2.44202	4.024	0.33294	0.01671	0.26783 −	0.06511 +	−2.01 % ↓
LR	5	2.64148	4.984	0.16936	0.00767	0.13447 −	0.03489 +	+0.30 % ↓
OP	3	6.67102	10.608	0.11870	0.00249	0.04034 −	0.07836 +	−2.20 % ↓
PA	7	3.51921	6.712	0.03228	0.00093	0.02566 −	0.00662 +	−1.07 % ↓
PE	9	6.49224	11.96	0.05222	0.00074	0.01822 +	0.03400 −	−1.82 % ↑
PI	6	1.9162	3.568	0.23744	0.00995	0.21719 −	0.02025 +	−0.86 % ↓
SA	9	4.92592	11.16	0.14284	0.00219	0.10401 +	0.03883 −	−1.02 % ↓
SE	11	4.124	8.936	0.04857	0.00562	0.02649 −	0.02208 +	−2.63 % ↓
SO	6	1.73432	2.384	0.26272	0.02816	0.17101 −	0.09171 +	−1.74 % =
TH	4	2.03279	2.84	0.08367	0.00449	0.06389 −	0.01978 −	−5.04 % ↓
VE	5	4.60828	9.448	0.29345	0.00987	0.24043 −	0.05302 +	−1.06 % ↓
VO	12	5.64556	10.056	0.27564	0.00912	0.09697 +	0.17867 −	−0.43 % ↑
WA	11	4.71566	9.208	0.22526	0.00464	0.15405 −	0.07121 +	−2.43 % ↑
WD	5	1.6652	2.664	0.06901	0.01461	0.04947 −	0.01954 +	−0.66 % ↓
WI	0	2.20816	2.936	0.07227	0.00938	0.02333 −	0.04894 +	+5.86 % ↓

Table 23: Bias-variance decomposition for a type B cascading ensemble of two decision trees using the entropy splitting criterion and the best $\overline{d}_1 \in [0,12]$ for the first decision tree. Signs +, =, − and Δ are referred to Tab. 19, and signs ↑ and ↓ are referred to Tab. 21.

first decision tree in a sequential ensemble to be a first approximation of the final classification, we must ensure that such decision tree has enough leaves, even if it is not the one yielding the best classification accuracy.

4.5.2 No additional info, mixed class

In the previous experiments, we have seen that the first decision tree must be large enough to provide a good approximation for class probabilities which are expected to be used by the second decision tree as new classification features to build more complex boundaries. In this case, no additional information is computed and carried over from one decision tree to the next one in the sequence, but only those vectors left unclassified by the first decision tree are used to train the second one, namely type A progressive decision trees.

As we have done in the previous experiments, we will start by studying the effect of ϵ and then,

set	$\overline{d}$	$\lceil \log_2(K) \rceil$	$\lceil \log_2(K) \rceil + 1$	$\lceil H \rceil$	$\overline{H}$	$\lceil R \rceil$	$\lceil \overline{d} \rceil$
CO	12	3	4	3	3	10	20
GL	5	3	4	3	5	4	6
IO	6	1	2	1	1	3	4
IR	3	2	3	2	2	2	3
LE	3	5	6	5	5	10	18
LI	8	1	2	1	1	4	6
LR	5	4	5	3	8	3	6
OP	3	4	5	4	4	7	11
PA	7	3	4	1	4	5	8
PE	9	4	5	4	4	8	13
PI	6	1	2	1	1	3	5
SA	9	3	4	3	4	6	12
SE	11	3	4	3	3	5	11
SO	6	1	2	1	1	3	4
TH	4	2	3	2	2	3	4
VE	5	2	3	2	2	6	11
VO	12	4	5	4	4	7	11
WA	11	2	3	2	2	6	11
WD	5	1	2	1	1	3	4
WI	0	2	3	2	2	3	3

Table 24: Comparison of $\overline{d}$ to several criteria related to data set characteristics.

once the best ϵ had been determined for each data set, we will study the combination of both parameters, $\overline{d}$ and ϵ using the results obtained previously.

4.5.2.1 Complete decision trees

As in the previous experiment, we will start combining two large decision trees, but we expect both trees to be smaller than those created in Sect. 4.5.1.2, as the first decision tree will be pruned back due to the use of the mixed class, and the second decision tree will have to classify less samples. Therefore, we grow a decision tree as usual but then we add a second pruning stage using the new labelling rule, defined by Eq. (3.1), recursively joining all leaves labelled as mixed. The only parameter here is ϵ, which determines the new labelling rule.

In this first experiment, we want to compare the two basic types of progressive decision trees without any fine-tuning. The basic idea is to see the relative importance of each parameter in the ensemble of type B or type A, $\overline{d}$ and ϵ respectively. Then we will study the best combination of both parameters at the same time, mainly to study how classification accuracy may be improved (or, at

least, maintained), when using smaller progressive decision trees.

As a first guess for ϵ, we will use a value closer to the misclassification error achieved by a single decision tree. Therefore, as misclassification error is a convex combination of the leaves partial misclassification errors, all those leaves making too many mistakes will be labelled as mixed and joined together. This setup allows us to study the proportion of samples falling in leaves which perform better than average (that is, the percentage of samples classified by the first decision tree), thus separating samples in two categories: better than average and worse than average. Tab. 25 shows the results for this experiment.

set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$	$\Delta\hat{L}$
CO	5.652	9.344	0.24323	0.00403	0.17326 −	0.06997 +	−0.41 % ↑
GL	0.72235	0.85714	0.35887	0.02927	0.28451 =	0.07436 +	+0.25 % ↑
IO	1.44582	2.12329	0.11508	0.00609	0.09573 −	0.01935 +	+6.72 % ↑
IR	1.26153	2.05085	0.04800	0.00590	0.02353 =	0.02447 +	+13.7 % ↑
LE	6.46263	9.096	0.16680	0.00135	0.07643 −	0.09037 +	−0.13 % ↑
LI	2.05682	3.06452	0.34254	0.01349	0.26783 −	0.07471 +	+0.82 % ↑
LR	1.03729	1.28448	0.16764	0.00816	0.13559 −	0.03205 −	−0.72 % ↓
OP	3.94312	5.424	0.12030	0.00299	0.04376 −	0.07654 +	−0.88 % ↑
PA	3.78171	6.672	0.03337	0.00057	0.02555 −	0.00782 +	+2.27 % ↑
PE	1.94782	2.76613	0.05290	0.00087	0.01735 −	0.03555 +	−0.55 % ↑
PI	2.29357	3.56637	0.24022	0.00835	0.21406 −	0.02616 +	+0.30 % ↑
SA	5.2186	8.704	0.14272	0.00239	0.10121 −	0.04151 +	−1.10 % ↓
SE	2.44838	3.54808	0.05016	0.00543	0.02805 −	0.02211 +	+0.56 % ↑
SO	1.54014	2.29412	0.26933	0.02603	0.18841 −	0.08092 +	+0.74 % ↑
TH	0.86621	1.05263	0.09022	0.00503	0.06944 +	0.02078 −	+2.39 % ↑
VE	2.86548	4.31683	0.29918	0.01028	0.25958 +	0.03960 −	+0.87 % ↑
VO	1.57001	1.70476	0.27619	0.00777	0.09212 −	0.18407 +	−0.23 % ↑
WA	2.302	3.39831	0.22941	0.00359	0.16224 +	0.06717 −	−0.63 % ↑
WD	1.13121	1.32836	0.07192	0.01487	0.05053 +	0.02139 +	+3.53 % ↑
WI	1.14018	1.32692	0.07320	0.00993	0.03000 +	0.04320 +	+7.22 % ↑

Table 25: Bias-variance decomposition for a type A cascading ensemble of two decision trees using the entropy splitting criterion and $\epsilon \approx \hat{L}$ for the first decision tree. Signs +, =, − and Δ are referred to Tab. 19, and signs ↑ and ↓ are referred to Tab. 21.

Notice that with this configuration for ϵ, classification accuracy is degraded for 12 out of 20 data sets, and that type B progressive decision trees outperform this ensemble (type A) in 18 out of 20 data sets. As ϵ is the only parameter we have to tune, it is obvious that using $\epsilon \approx \hat{L}$ is a simple but not a good criterion. In all cases but one (the *waveform* data set), an increase of classification accuracy is due to a reduction of bias.

Tab. 26 shows the percentage of classified samples (P_{T_1}) and the classification accuracy of the first decision tree ($\hat{L}_1$) in the cascading ensemble. The characteristics of the first decision tree (R_1 and $\overline{d}_1$) are also shown.

set	ϵ	R_1	$\overline{d}_1$	P_{T_1}	$\hat{L}_1$
CO	0.25	9.55809	19.736	0.95701	0.23387 $\downarrow$
GL	0.36	3.97253	5.952	0.98930	0.35649 $\downarrow$
IO	0.11	2.42933	3.384	0.90503	0.09466 $\downarrow$
IR	0.05	1.80073	2.584	0.86729	0.03411 $\downarrow$
LE	0.17	9.90946	17.904	0.97781	0.15513 $\downarrow$
LI	0.35	3.38744	5.744	0.92237	0.34045 $\uparrow$
LR	0.18	2.83643	5.328	0.92574	0.14458 $\downarrow$
OP	0.12	6.66117	10.336	0.95914	0.10583 $\downarrow$
PA	0.04	4.74536	7.624	0.91485	0.01813 $\downarrow$
PE	0.05	7.30473	12.408	0.98908	0.05024 $\downarrow$
PI	0.24	2.33497	4.472	0.70791	0.23783 $\downarrow$
SA	0.15	5.55941	11.28	0.88077	0.10983 $\downarrow$
SE	0.05	4.36087	10.144	0.96127	0.04270 $\downarrow$
SO	0.28	2.24472	3.256	0.95281	0.26495 $\downarrow$
TH	0.09	2.35748	3.008	0.96711	0.08604 $\downarrow$
VE	0.30	5.22411	10.016	0.91072	0.27725 $\downarrow$
VO	0.28	6.67227	10.4	0.98158	0.27109 $\downarrow$
WA	0.23	5.24605	10.104	0.93650	0.22051 $\downarrow$
WD	0.07	2.10387	3.552	0.95583	0.06456 $\downarrow$
WI	0.07	2.25071	2.976	0.91493	0.06030 $\downarrow$

Table 26: Percentage of classified vectors and classification accuracy for the first decision tree in Tab. 25. Signs $\uparrow$ and $\downarrow$ are referred to Tab. 19.

Notice that P_{T_1} is close to one for most data sets, showing that the first decision tree is big and complex enough to classify all the input data at the given accuracy threshold. As $\hat{L}_1$ is a convex combination of the partial misclassification error at each leaf, a large P_{T_1} and a $\hat{L}_1$ close to ϵ means that almost all leaves are pure enough (with respect to ϵ) and that only a few leaves remain too impure. On the other hand, a few data sets (*iris*, *pima* and *sat*) show lower values for P_{T_1}, due to the fact that the classification problem may be easily broken in an easy problem for the majority of the input samples, and a more difficult one for the rest. Notice also that T_1 could be used as a partial classification system with the reject option (Kittler et al., 1998), increasing classification accuracy by reducing the number of classified samples.

Therefore, we repeat the same experiment using a lower value for ϵ, we could use $\epsilon \approx \hat{L}/2$, for instance. The basic idea is to find a simple criterion related to $\hat{L}$, if possible. We use the factor $1/2$

because it is directly related to the error generalization bounds, as we will describe in Sect. 5.1.1.

set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$	$\Delta\hat{L}$
CO	7.38558	13.304	0.24286	0.00412	0.16833 −	0.07453 +	−0.56 % ↑
GL	1.84396	2.44186	0.36259	0.03222	0.27042 −	0.09217 +	+1.29 % ↑
IO	1.39676	1.93878	0.11337	0.00623	0.09231 −	0.02106 +	+5.14 % ↑
IR	1.13892	1.60825	0.04800	0.00436	0.02353 =	0.02447 +	+13.7 % ↑
LE	6.78746	10.008	0.16724	0.00138	0.07608 −	0.09116 +	+0.13 % ↓
LI	1.91157	2.91818	0.34484	0.01483	0.25565 −	0.08919 +	+1.50 % ↑
LR	1.49141	1.89167	0.16800	0.00823	0.13220 −	0.03580 +	−0.51 % ↓
OP	4.30877	6.16	0.12026	0.00321	0.04344 −	0.07682 +	−0.91 % ↑
PA	3.721	6.496	0.03313	0.00069	0.02555 −	0.00758 +	+1.53 % ↑
PE	2.22903	3.27419	0.05294	0.00084	0.01774 −	0.03520 +	−0.47 % ↑
PI	1.87514	3.032	0.23891	0.01063	0.21484 −	0.02407 +	−0.47 % ↑
SA	5.2385	9.752	0.14254	0.00230	0.09832 −	0.04422 +	−1.23 % ↓
SE	2.65425	4.04545	0.05006	0.00521	0.02597 −	0.02409 +	+0.36 % ↑
SO	1.29581	1.71765	0.27444	0.02940	0.18841 −	0.08603 +	+2.65 % ↑
TH	0.85465	1.01667	0.09100	0.00558	0.06944 +	0.02156 +	+3.28 % ↑
VE	3.76781	6.18333	0.29813	0.00952	0.26028 +	0.03785 −	+0.52 % ↑
VO	1.8132	2.16216	0.27649	0.00807	0.09273 −	0.18376 +	−0.12 % ↑
WA	3.74589	6.29839	0.22713	0.00342	0.15644 −	0.07069 +	−1.62 % ↑
WD	1.1465	1.57143	0.07259	0.01468	0.04947 −	0.02312 +	+4.49 % ↑
WI	1.28001	1.6875	0.07013	0.01025	0.02333 −	0.04680 +	+2.72 % ↓

Table 27: Bias-variance decomposition for a type A cascading ensemble of two decision trees using the entropy splitting criterion and $\epsilon \approx \hat{L}/2$ for the first decision tree. Signs +, =, − and Δ are referred to Tab. 19, and signs ↑ and ↓ are referred to Tab. 21.

In this case, results (shown in Tab. 21) are even worse than in the previous experiment. Variance is always worse (but in one case, the *vehicle* data set) than with a single decision tree, while bias is usually reduced except for a few data sets.

Notice that in this case the percentage of classified sample is much lower for some data sets, and both R and $\overline{d}$ are also smaller. Nevertheless, $\hat{L}$ shows a more unpredictable behavior, and it is reduced for some data sets and dramatically increased for other. Tab. 28 shows the results for the first decision tree in the ensemble. Notice that in this case, $\hat{L}_1$ is larger than ϵ for most data sets (17 out of 20), showing that the setup $\epsilon \approx \hat{L}/2$ is too restrictive, as it is impossible to use such criterion as a simple way to separate leaves in two classes. This is directly related to the concepts of margin and generalization error that we will study in Sect. 5.1.1, where we will show that obtaining a good generalization error performance is directly related to assuming a certain error when classifying the training set. Therefore, we will proceed as with type B progressive decision trees, trying to find the

set	ϵ	R_1	$\overline{d}_1$	P_{T_1}	$\hat{L}_1$
CO	0.13	9.51216	19.712	0.79415	0.20387 ↓
GL	0.18	3.95412	5.952	0.89645	0.34780 ↓
IO	0.06	2.19395	3.128	0.62366	0.13516 ↑
IR	0.03	1.66464	2.368	0.69098	0.02328 ↓
LE	0.09	9.90754	17.904	0.96501	0.15208 ↓
LI	0.18	3.11645	5.376	0.54010	0.39416 ↑
LR	0.09	2.78527	5.216	0.85293	0.13162 ↓
OP	0.06	6.65007	10.336	0.92811	0.10136 ↓
PA	0.02	4.69012	7.592	0.85998	0.01932 ↓
PE	0.03	7.3028	12.408	0.97753	0.04943 ↓
PI	0.12	1.89491	3.712	0.25378	0.43186 ↑
SA	0.08	5.44146	11.128	0.72265	0.08546 ↓
SE	0.06	4.34446	10.144	0.91918	0.03994 ↓
SO	0.14	2.14872	3.16	0.71641	0.32787 ↑
TH	0.05	2.35748	3.008	0.93411	0.08451 ↓
VE	0.15	5.1506	9.928	0.74979	0.24260 ↓
VO	0.14	6.67207	10.4	0.97268	0.26891 ↓
WA	0.12	5.14215	9.896	0.74776	0.19200 ↓
WD	0.04	2.07384	3.488	0.86337	0.07803 ↑
WI	0.04	2.19708	2.936	0.82040	0.08277 ↑

Table 28: Percentage of classified vectors and classification accuracy for the first decision tree in Tab. 27. Signs ↑ and ↓ are referred to Tab. 19.

best ϵ for each data set by exhaustive search in the $(0, 1/2)$ range.

Varying ϵ from 0 to 1/2 can be efficiently done starting with the complete decision tree built as usual, once the pruning algorithm has been applied, and then using the new labelling rule, starting with $\epsilon = 1/2$ and then reducing it and pruning back all mixed leaves at the same time, so there is no need to repeat the training process for each value of ϵ.

Notice that, for a few data sets, both R and $\overline{d}$ are zero, which means that a single decision tree outperforms any ensemble of two type A progressive decision trees. In such case, the results are the same than those shown by Tab. 19 as only one large decision tree is used. Type A progressive decision trees outperform classical decision trees in 14 out of 20 data sets, but they obtain poorer results than type B progressive decision trees. Only in two data sets (*letter* and *sat*), type A progressive decision trees obtained a better classification accuracy.

Tab. 30 shows the characteristics of the first decision tree and the ϵ used to build such decision tree. For those data sets where was not possible to build a type A progressive decision tree, $P_{T_1} = 1$ is shown meaning that a complete decision tree was built.

set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$	$\Delta\hat{L}$
CO	7.6571	14.136	0.24204	0.00347	0.16524 −	0.07680 +	−0.89 % ↑
GL	0	0	0.35797	0.02940	0.28451 =	0.07346 =	0.0 %
IO	1.5445	1.99174	0.10762	0.00532	0.08718 −	0.02044 +	−0.19 % ↑
IR	0	0	0.04220	0.00675	0.02353 =	0.01867 =	0.0 %
LE	6.33666	8.832	0.16664	0.00145	0.07677 −	0.08987 −	−0.23 % ↑
LI	2.21752	3.43902	0.33412	0.01339	0.26609 −	0.06803 +	−1.66 % ↑
LR	1.23867	1.56667	0.16705	0.00884	0.13220 −	0.03485 +	−1.07 % ↓
OP	4.67565	6.992	0.11988	0.00360	0.04024 −	0.07964 +	−1.23 % ↑
PA	0	0	0.03263	0.00118	0.02675 =	0.00588 =	0.0 %
PE	1.80618	2.42742	0.05284	0.00092	0.01774 −	0.03510 +	−0.66 % ↑
PI	2.15572	3.45161	0.23866	0.00830	0.21563 −	0.02303 +	−0.35 % ↑
SA	5.25638	9.136	0.14230	0.00276	0.09842 −	0.04388 +	−1.39 % ↓
SE	2.61396	3.94393	0.04985	0.00556	0.02623 −	0.02362 +	−0.06 % ↑
SO	1.34008	1.73148	0.26690	0.02831	0.17681 −	0.09009 +	−0.17 % ↑
TH	0	0	0.08811	0.00403	0.06667 =	0.02144 =	0.0 %
VE	3.90313	6.66129	0.29623	0.00884	0.23901 −	0.05722 +	−0.12 % ↑
VO	1.67586	1.87037	0.27605	0.00780	0.09455 +	0.18150 −	−0.28 % ↑
WA	3.9257	6.60484	0.22673	0.00426	0.15644 −	0.07029 +	−1.79 % ↑
WD	0	0	0.06947	0.01441	0.05052 =	0.01895 =	0.0 %
WI	0	0	0.06827	0.01014	0.02667 =	0.04160 =	0.0 %

Table 29: Bias-variance decomposition for a type A cascading ensemble of two decision trees using the entropy splitting criterion and the best $\epsilon \in (0, 1/2)$ for the first decision tree. Signs +, =, − and Δ are referred to Tab. 19, and signs ↑ and ↓ are referred to Tab. 21.

This experiment shows the difficulty of finding an appropriate value for ϵ for a given data set. It seems completely unrelated to $\hat{L}$ and it probably strongly depends on data set intrinsic characteristics. This is mainly caused by the too simplistic new labelling rule, which uses the same ϵ across all the leaves of the decision tree without taking into account *a priori* class probabilities or any other criteria related to the intrinsic characteristics of the data set. Furthermore, P_{T_1} is also very high for some data sets, which partially explains why type A progressive decision trees do not perform as well as expected: it is very difficult to build a good first decision tree using such a simple labelling rule. The number of classes and the uneven class distribution should be taken into account, adapting the labelling rule to ensure all classes are present in the final decision tree. This is directly related to the definition of margin, which is only properly defined for $K = 2$, and not for $K > 2$.

Regarding the cost of building and using such classifiers, there is only a remarkable improvement in those data sets with P_{T_1} small enough. As an estimate of both training and exploitation costs, we can compute the expected number of questions needed to classify a sample. For a classical

set	ϵ	R_1	$\overline{d}_1$	P_{T_1}	$\hat{L}_1$
CO	0.10	9.48118	19.704	0.72800	0.19447 $\downarrow$
GL	0.50	3.96833	5.952	1.00000	0.35797 –
IO	0.01	1.81538	2.728	0.21149	0.38515 $\uparrow$
IR	0.50	1.80073	2.584	1.00000	0.04220 –
LE	0.23	9.90987	17.904	0.98092	0.15625 $\downarrow$
LI	0.04	2.64859	4.704	0.27826	0.55620 $\uparrow$
LR	0.14	2.81544	5.288	0.89785	0.13874 $\downarrow$
OP	0.02	6.60534	10.336	0.82205	0.09927 $\downarrow$
PA	0.50	4.78741	7.84	1.00000	0.03263 –
PE	0.07	7.30509	12.408	0.99187	0.05051 $\downarrow$
PI	0.17	2.05722	4.008	0.42269	0.35097 $\uparrow$
SA	0.12	5.52563	11.224	0.83640	0.10164 $\downarrow$
SE	0.03	4.34968	10.144	0.93188	0.04092 $\downarrow$
SO	0.02	1.8758	2.784	0.40209	0.45965 $\uparrow$
TH	0.50	2.35748	3.008	1.00000	0.08811 –
VE	0.02	4.73396	9.6	0.38599	0.28234 $\downarrow$
VO	0.20	6.67227	10.4	0.97845	0.27021 $\downarrow$
WA	0.10	5.09988	9.84	0.70140	0.18871 $\downarrow$
WD	0.50	2.10501	3.568	1.00000	0.06947 –
WI	0.50	2.25573	2.984	1.00000	0.06827 –

Table 30: Percentage of classified vectors and classification accuracy for the first decision tree in Tab. 29. Signs $\uparrow$ and $\downarrow$ are referred to Tab. 19.

decision tree, such number is just R_1. For a type B progressive decision tree built using two decision trees, it is $R_1 + R_2$, and for a type A progressive decision tree using also two decision trees it is $R'_1 + (1 - P_{T_1})R'_2$, although in this case R'_1 is not the same as in the previous ensembles as the first decision tree has been pruned back by using the new labelling rule and R'_2 is also different. Tab. 31 shows the classification cost estimates for the three types of ensembles.

Notice that type A progressive decision trees do not increase classification cost too much, and they are much less computationally expensive than type B progressive decision trees. Although not shown in Tab. 31, type B progressive decision trees use $K + 2$ additional classification features, which obviously also increases training cost and tree internal representation size.

4.5.2.2 Limited depth decision trees

In this experiment we study the effect of limiting the maximum depth of the first decision tree, trying to force such decision tree to be smaller and, therefore, to classify a smaller percentage of

set	single	type B	type A
CO	9.56141	16.57122	11.56391
GL	3.96833	7.47859	—
IO	2.47415	4.68891	3.03323
IR	1.80073	3.58743	—
LE	9.91032	18.92981	10.03077
LI	3.38671	5.93295	4.24906
LR	2.84827	5.39462	2.94197
OP	6.66539	12.54571	7.43737
PA	4.78741	8.48855	—
PE	7.30555	13.09	7.31977
PI	2.44464	4.19881	3.30174
SA	5.60507	10.42098	6.38557
SE	4.37432	8.39405	4.52774
SO	2.24472	4.14837	2.67705
TH	2.35748	4.37022	—
VE	5.2307	9.21647	7.13052
VO	6.67239	12.32228	6.70838
WA	5.25705	10.58855	6.27209
WD	2.10501	3.70648	—
WI	2.25573	4.24564	—

Table 31: Estimated classification cost for a classical decision tree, a type B progressive decision tree (two decision trees) and a type A progressive decision tree (two decision trees).

input samples. Two parameters need to be fixed for this experiment: the maximum depth of the first tree $\overline{d}_1$, and ϵ, which determines the new labeling rule. Using the knowledge acquired from the previous experiments, we could use the best d_1 as found in Tab. 23 and the best $\epsilon \in (0, 1/2)$ could be chosen, so we could compare it to Tab. 29, in order to see whether it is possible to improve classification accuracy by building a smaller first decision tree. Nevertheless, as we want our first decision tree to be a good approximation of a classical decision tree, we will force $\overline{d}_1$ to be large enough so the resulting decision tree has enough leaves to represent all classes in the training set. Tab. 32 shows the results obtained for this experiment using the setup described above.

Notice that it is very difficult to establish an appropriate pair of $\overline{d}$ and ϵ for each data set. Furthermore, varying $\overline{d}$ has an effect on ϵ, which must be sometimes larger and sometimes smaller. Only in 7 out of 20 data sets limiting the maximum depth of the first decision tree in the type A ensemble produces an increase in classification accuracy. Only in three cases (*iris*, *thyroid* and *wdbc*) is not possible to build a type A progressive decision tree.

Tab. 33 shows the characteristics for the first decision tree in the type A progressive ensemble.

set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$	$\Delta\hat{L}$
CO	8.87792	17.352	0.24221	0.00419	0.16905 −	0.07316 +	−0.82 % ↑
GL	1.57049	1.96471	0.36608	0.03181	0.29577 +	0.07031 −	+2.27 % ↑
IO	1.59696	2.07258	0.10674	0.00505	0.08718 −	0.01956 +	−1.01 % ↓
IR	0	0	0.04345	0.00697	0.02353 =	0.01992 +	+2.96 % ↑
LE	10.045	18.048	0.16606	0.00147	0.07365 −	0.09241 −	−0.57 % ↓
LI	2.44208	3.8629	0.33350	0.01461	0.26609 −	0.06741 +	−1.84 % ↑
LR	1.54715	2	0.16800	0.00929	0.13785 −	0.03015 −	−0.51 % ↑
OP	6.64992	10.6	0.11873	0.00428	0.03938 −	0.07935 +	−2.18 % ↓
PA	3.02766	4.848	0.03257	0.00096	0.02599 −	0.00658 +	−0.18 % ↓
PE	5.51657	8.464	0.05325	0.00045	0.01790 −	0.03535 +	+0.11 % ↑
PI	2.29326	3.808	0.23816	0.00980	0.21406 −	0.02410 +	−0.56 % ↓
SA	5.70043	9.848	0.14243	0.00260	0.09879 −	0.04364 +	−1.30 % ↑
SE	2.65171	3.90654	0.04987	0.00544	0.02831 −	0.02156 +	−0.02 % ↑
SO	1.33696	1.72477	0.26690	0.02831	0.17681 −	0.09009 +	−0.17 % −
TH	0	0	0.08856	0.00327	0.06667 =	0.02189 +	+0.51 % ↑
VE	4.89176	8.736	0.29617	0.00738	0.24326 −	0.05291 +	−0.14 % ↓
VO	1.68975	1.88889	0.27615	0.00775	0.09454 =	0.18161 −	−0.24 % ↑
WA	3.95514	6.712	0.22649	0.00500	0.15664 −	0.06984 +	−1.90 % ↓
WD	0	0	0.07048	0.01424	0.05158 +	0.01890 −	+1.45 % ↑
WI	1.68617	2.288	0.07200	0.00695	0.02667 =	0.04533 +	+5.46 % ↑

Table 32: Bias-variance decomposition for a type A cascading ensemble of two decision trees using the entropy splitting criterion, a limited maximum depth and the best $\epsilon \in (0, 1/2)$ for the first decision tree. Signs +, =, − and Δ are referred to Tab. 19, and signs ↑ and ↓ to Tab. 29.

The maximum depth $\overline{d}$ for each data set is also shown.

In this case, the first decision tree is smaller than those computed in Tab. 30, due to the imposed constrain on $\overline{d}$. To sum up, type A progressive decision trees may be used as accurate partial classification systems, reducing also classification cost when the maximum depth is limited. Both P_{T_1} and $\hat{L}_1$ are also reduced (but for a few data sets), as smaller decision trees can capture only a few boundaries between classes. In fact, when ϵ is too small, as for the *letter* data set, the resulting first decision tree does only remove elements from a few classes. As mentioned before, this is also related to the poor definition of margin when $K > 2$.

4.5.3 Additional info, mixed class

Finally, in this section we combine the two basic types of cascading ensembles, A and B, into a new type C, trying to combine the better characteristics of each type: bias reduction by using new

set	$\overline{d}$	ϵ	R_1	$\overline{d}_1$	P_{T_1}	$\hat{L}_1$
CO	12	0.11	8.79391	12	0.62223	0.17325 ↓
GL	5	0.28	3.73354	4.856	0.92890	0.35697 ↓
IO	6	0.01	1.73368	2.544	0.17559	0.38508 ↑
IR	3	0.50	1.76266	2.32	1.00000	0.04345 ↑
LE	5	0.01	1.90843	4.96	0.03671	0.01643 ↓
LI	8	0.04	2.56428	4.088	0.25447	0.54999 ↑
LR	5	0.15	2.66264	4.488	0.86884	0.13173 ↓
OP	4	0.05	1.82719	3.728	0.13394	0.08373 ↓
PA	7	0.16	4.60144	6.44	0.97645	0.02627 ↓
PE	9	0.01	6.90807	9	0.82517	0.04369 ↓
PI	6	0.15	1.8287	3.112	0.32775	0.35448 ↑
SA	9	0.15	5.45159	8.864	0.85322	0.10341 ↓
SE	11	0.04	4.3513	9.856	0.94900	0.04118 ↓
SO	6	0.02	1.87563	2.776	0.40186	0.45960 ↑
TH	4	0.50	2.31424	2.904	1.00000	0.08856 ↑
VE	5	0.04	2.95956	4.416	0.21348	0.19752 ↓
VO	12	0.20	6.66624	10.352	0.97816	0.27026 ↓
WA	11	0.11	5.14802	9.264	0.72825	0.18937 ↓
WD	5	0.44	2.04661	3.368	0.99945	0.07044 ↑
WI	2	0.02	1.35803	1.56	0.34240	0.24036 ↑

Table 33: Percentage of classified vectors and classification accuracy for the first decision tree in Tab. 32. Signs ↑ and ↓ are referred to Tab. 19.

classification additional features, and variance reduction by using smaller decision trees due to the use of smaller training sets. This also might reduce both classification and exploitation costs as only small decision trees are used as the first stage and then reduced training sets are used to build the second stage (in a two stage sequential cascading ensemble).

For the first experiment of this section, we use the simplest criteria: we set $\overline{d} = \lceil \overline{d} \rceil$ from Tab. 19 and $\epsilon \approx \hat{L}$. The basic idea is to build a classical decision tree and then use the computed decision tree results as the criteria for determining $\overline{d}$ and ϵ. Tab. 34 shows the results obtained for this experiment.

Once again, the gain in classification accuracy is caused by a reduction of the bias component, while variance is always increased but for two data sets (*iris* and *wavelet*). Without any parameter fine tuning (for each data set we used fixed values for both $\overline{d}$ and ϵ estimated from a classical decision tree), type C progressive decision trees perform slightly better than type A progressive decision trees, as expected.

Tab. 35 shows the characteristics for the first decision tree in the type C progressive ensemble.

set	R	$\overline{d}$	$\hat{L}$	$\sigma_{\hat{L}}$	$\hat{B}$	$\hat{V}$	$\Delta\hat{L}$
CO	6.64514	15.424	0.24309	0.00394	0.16738 −	0.07571 +	−0.46 %
GL	2.89453	4.16129	0.36631	0.02747	0.26197 −	0.10434 +	+2.33 %
IO	1.74812	2.28	0.10585	0.00597	0.08718 −	0.01867 +	−1.84 %
IR	1.17505	1.70796	0.04737	0.00617	0.03137 +	0.01600 −	+12.3 %
LE	6.21132	15.256	0.16681	0.00167	0.07544 −	0.09137 +	−0.13 %
LI	2.74372	4.552	0.33586	0.01430	0.26087 −	0.07499 +	−1.15 %
LR	1.99909	3.52	0.17162	0.00690	0.13446 −	0.03716 +	+1.63 %
OP	4.42625	7.944	0.11956	0.00315	0.04034 −	0.07922 +	−1.49 %
PA	3.10464	6.504	0.03354	0.00131	0.02566 −	0.00788 +	+2.79 %
PE	2.94396	5.872	0.05257	0.00073	0.01730 −	0.03527 +	−1.17 %
PI	2.2542	4.272	0.23891	0.01018	0.21485 −	0.02406 +	−0.25 %
SA	4.66852	10.224	0.14368	0.00248	0.09898 −	0.04470 +	−0.44 %
SE	2.84235	4.96639	0.05008	0.00554	0.02520 −	0.02488 +	+0.40 %
SO	1.42814	1.9	0.26910	0.02879	0.18551 −	0.08359 +	+0.65 %
TH	1.31032	1.71429	0.09000	0.00457	0.06667 =	0.02333 +	+2.15 %
VE	3.59676	6.696	0.29470	0.00653	0.24114 −	0.05356 +	−0.64 %
VO	2.6545	3.74167	0.27598	0.00835	0.09333 −	0.18265 +	−0.30 %
WA	4.12144	8.056	0.22901	0.00449	0.16024 −	0.06877 −	−0.81 %
WD	1.3261	2.03306	0.07107	0.01433	0.04631 −	0.02476 +	+2.30 %
WI	1.37422	1.79	0.07040	0.00857	0.02000 −	0.05040 +	+3.12 %

Table 34: Bias-variance decomposition for a type C cascading ensemble of two decision trees using the entropy splitting criterion. Signs +, =, − and Δ are referred to Tab. 19.

The maximum depth $\overline{d}$ and the confidence threshold ϵ used for each data set are also shown.

Notice that some data sets show a very bizarre behavior, as the *pima* data set, for instance. The value $R_1 < 1$ shows that degenerated decision trees (a single leaf labelled as mixed) are being built, probably caused by a bad choice of ϵ. Only in five data sets a small first decision tree (in comparison with Tab. 19) is built, showing that $\overline{d}$ and ϵ must be fine tuned in order to build a good first decision tree. Furthermore, building a small decision tree with $\hat{L}_1 < \epsilon$ does not ensure that the second decision tree, even using additional classification features, will perform better.

To sum up, type C progressive decision trees seem to be a compromise between classification cost and classification accuracy, splitting the classification problem in a sequence of simpler problems. Nevertheless, parameters ($\overline{d}$ and ϵ) become critical as they completely determine the structure of the final progressive decision tree, and it is difficult to determine *a priori* good values for such parameters. Furthermore, the amount of additional information which is carried from one stage to the next is also an important subject of study, as it should be reduced accordingly to the percentage of samples classified by the first decision tree in order to minimize the curse of dimensionality problem.

set	$\overline{d}$	ϵ	R_1	$\overline{d}_1$	P_{T_1}	$\hat{L}_1$
CO	20	0.25	9.32212	18.704	0.25881	0.28407 ↑
GL	6	0.36	3.6932	5.384	0.51144	0.34424 ↓
IO	4	0.11	1.48462	2.048	0.11022	0.40820 ↑
IR	3	0.05	1.58836	2.04	0.64502	0.01987 ↓
LE	18	0.17	9.90019	17.56	0.81441	0.16602 ↓
LI	6	0.35	1.9791	2.952	0.13969	0.59606 ↑
LR	6	0.18	2.22569	4.328	0.21356	0.38343 ↑
OP	11	0.12	6.51256	10.248	0.54053	0.13457 ↑
PA	8	0.04	4.40748	6.696	0.08375	0.14134 ↑
PE	13	0.05	7.26876	12.168	0.73206	0.05819 ↑
PI	5	0.24	0.90831	1.544	0.02744	0.70596 ↑
SA	12	0.15	5.16731	10.568	0.18450	0.15175 ↑
SE	11	0.05	4.3056	9.856	0.77392	0.03691 ↓
SO	4	0.28	1.7467	2.424	0.32754	0.46167 ↑
TH	4	0.09	2.20772	2.8	0.60633	0.10918 ↑
VE	11	0.30	4.63027	8.552	0.33163	0.30592 ↑
VO	11	0.28	6.65075	10.184	0.88924	0.26718 ↓
WA	11	0.23	4.61037	8.96	0.14553	0.32017 ↑
WD	4	0.07	1.86826	3.032	0.36080	0.20037 ↑
WI	3	0.07	1.97342	2.544	0.60627	0.14176 ↑

Table 35: Percentage of classified vectors and classification accuracy for the first decision tree in Tab. 34. Signs ↑ and ↓ are referred to Tab. 19.

4.6 Summary

We summarize in this section the results obtained throughout this chapter which give support to our ensemble. The most important experiments describing the generalization of the cascading paradigm may be found in (Minguillón and Pujol, submitted). The main conclusions that may be drawn from all the empirical evaluation carried out in this chapter are:

1. No splitting criterion is better than the rest for all data sets, although both entropy and R-norm ($R = 1/2$) perform slightly better. On the contrary, there is a clear loser, the Bayes error criterion. Furthermore, both entropy and R-norm ($R = 1/2$) seem to produce decision trees closer to those a human expert would create, which may be useful for internal data structure inspection. The twoing criterion did not perform as well as expected, either, even when the number of classes is large.

2. For data sets with a large baseline error, the k-NN classifier always outperforms a classifier

based on decision trees. Furthermore, in all of these data sets, the contribution of variance to the estimated error is larger than the contribution of bias, which is not the common behavior for decision trees, where bias is usually much larger than variance. This shows the different behavior from a bias-variance point of view of the k-NN classifier and decision trees, which make them suitable for creating a cascading ensemble or even a simple voting scheme for improving classification performance.

3. The bias-variance decomposition is a powerful tool to compare different kinds of classifiers, and it gives useful information about their respective behaviors. As classifier combination schemes rely on combining different classifiers, this difference may be measured using the bias-variance decomposition.

4. Type B progressive decision trees are an effective tool for increasing classification accuracy, mainly by fighting against bias, due to the fact that more complex boundaries are built when the new additional classification features are used by the next decision tree in the ensemble. Type A progressive decision trees allow us to build accurate partial classification systems, by reducing the percentage of classified samples, and breaking the classification problem in two (or more) parts: an easy problem which is solved, and a more difficult problem which needs more accurate classifiers to be solved. Finally, type C progressive decision trees try to combine partial classification with building more complex classifiers, so classification accuracy is increased with respect to type A progressive decision trees but with a lower cost than type B progressive decision trees. Nevertheless, the use of orthogonal hyperplanes is the main reason for improving bias, by building more complex boundaries with the additional classification features.

5. Maybe the parameters fine-tuning process is the weakest part of our analysis. It is very difficult to infer any good *a priori* values for $\overline{d}$ and ϵ, and only t seems to be easy to bound. Two or three decision trees seem to be enough to build progressive decision trees. The larger the data set, the larger t can be, although the need to obtain different classifiers under a bias-variance decomposition approach makes t to be bounded, so it is unlikely to use more than five or six decision trees, even for large data sets. This subject is still an elusive goal and more exhaustive testing is needed in order to find a relationship between $\overline{d}$ and ϵ, and some unknown intrinsic characteristics of the training set.

6. Experiments also show that the new labelling rule is too simplistic, as it does not take into account any criteria related to the number of classes or class distribution. The definition of margin, related to two-class classification problems, it is too poor to represent all the possible situations at each leaf when the number of classes is larger than two. This is obviously a further research subject, too.

7. Regarding the cost issue, type B progressive decision trees become very expensive for data sets with a large number of classes, as $K + 2$ new additional classification features are added at each stage. Increasing the number of decision trees in the ensemble reduces misclassification error but both training and exploitation costs increase dramatically. On the other hand, type A progressive decision trees are also more computationally expensive than classical decision trees, but only in a small amount. Only for really large data sets, type A progressive decision trees may be used to build complex classification systems with a reduced cost. Type C progressive decision trees are a compromise between accuracy and classification cost, as they take advantage of the characteristics of each basic type.

Chapter 5

A theoretical framework

Mathématiques: Dessèchent le coeur
Gustave Flaubert, *Le Dictionnaire des Idées Reçues*

In the previous chapters we have discussed several methods to build and to ensemble progressive classification trees using empirical criteria obtained from experimentation. All these methods lack of a formal framework that could explain why progressive classification trees (may) perform better than classical classification trees under some (unknown) conditions. Furthermore, such algorithms need to define several parameters (number of trees, mixed label threshold, tree maximum depth, and so) which make them difficult to use in an automated learning environment. Although decision tree design is a very interactive task involving data mining expertise, it would be interesting to have some criteria to know in advance when to use progressive decision trees instead of classical decision trees, and a basic set of guidelines for determining the parameters involved in the training process.

In this chapter we try to understand these problems by studying the generalization error of classical decision trees and studying how such work could be extended to include the progressive decision tree approach. Our intuition is that progressive decision trees are good because only the best splits are used (avoiding overfitting) and only small decision trees are built and then combined under a cascading paradigm.

Surprisingly, the number of papers related to other types of ensembles such as boosting, bagging or stacking is really high when compared to cascading. Although it cannot be considered complete, there are no references directly related to the cascading paradigm in the most important compilation of multiple classifier systems (Kittler and Roli, 2000; Kittler and Roli, 2001; Kittler and Roli, 2002), and only a few papers describe hierarchical or multi-stage systems.

All of these types of ensembles seem to contradict Occam's razor, as more and more complex classification systems are created to explain data, increasing the final classification system complexity without incurring in overfitting (Murphy and Pazzani, 1994; Webb, 1996). A probable reason is

that simple decision trees are not complex enough to extract all information present in input data, specially when only orthogonal splits are used as decision functions. Our experiments in Sect. 4.5 also show such behavior.

5.1 Problem formulation

In this section we define the basic behavior of a decision tree as a function g which maps an input vector x to the space of possible labels Y. We are interested in studying the error generalization performance for a given decision tree, and then to extend such study to include also progressive decision trees.

Suppose we are in a two-class classification problem, so $K = 2$, $Y = \{-1, 1\}$. Nevertheless, we use a mixed class to denote a set is not pure enough, so $Y = \{-1, 0, 1\}$ for notation purposes when dealing with progressive decision trees. This does not mean $K = 3$, as the zero value is only used to denote that a decision tree does not generate an output for an input vector. Then, a decision tree T may be seen as a function g which maps an input vector x to a label y, as follows:

$$T(x) = \text{sgn}(g(x)) \tag{5.1}$$

where $\text{sgn}(x)$ is -1 for $x < 0$, 0 for $x = 0$ and $+1$ for $x > 0$, and $g(x)$ is defined as

$$g(x) = \sum_{i=1}^{|\tilde{T}|} p_i \sigma_i h_i(x) \tag{5.2}$$

where $\tilde{T}$ is the set of leaves of T, p_i is the probability of leaf i, σ_i is the label assigned to leaf i and $h_i(x)$ is a function which is 1 if x falls in leaf i and 0 otherwise. Actually, the set of functions $\{h_i\} \in \mathcal{H}$ defines completely the decision tree behavior, as $\mathcal{H}$ is a set of $\{-1, 1\}$-valued functions defined in the input space X. Notice that Eq. (5.2) is also valid if we allow $\sigma_i = 0$ as a possible label, so $T(x)$ will be also 0.

Following the notation from (Golea et al., 1998), define the convex hull $co(\mathcal{H})$ of $\mathcal{H}$ as the set of $[-1, 1]$-valued functions of the form $\sum_i a_i h_i$ where $a_i \geq 0$, $\sum_i a_i = 1$ and $h_i \in \mathcal{H}$. Taking $a_i = p_i \sigma_i$, it is easy to see that $g(x)$ defined in Eq. (5.2) is in $co(\mathcal{H})$, so $T(x)$ may be studied under this framework. The only condition is that there is at least one leaf with $\sigma_i \neq 0$.

In the rest of this chapter we will study this framework and several works which extend it in order to include progressive decision trees.

5.1.1 Generalization error

In Sect. 2.6 we described the BFOS pruning algorithm which was used to reduce tree size using a second data set (the corpus set), with the aim of reducing overfitting. As stated in (Scheffer, 2000), decision tree algorithms have to solve two distinct but related problems: to identify the size of the

optimal decision tree (which leads to optimal generalization error) and to minimize the empirical error rate $\hat{L}_n$. For the training set D_n it is possible to build a tree T such $\hat{L}_n = 0$, and then use pruning to find the best subtree which minimizes $\hat{L}_{n,m}$, the resubstitution estimate for the corpus set C_m. Using Eqs. (2.5) and (2.6) we can compute a confidence interval about our generalization error estimate.

But suppose we do not have a second data set (because we do not have enough data available, for example) and what we want is to estimate the generalization error for a given tree T and a given training data set D_n using a more theoretical approach, using the whole training set for building a classifier. Intuitively, our misclassification error estimation should take into account any measurable concept about T and D_n. Actually, such approach may include any algorithm which minimizes a regularization function $\Upsilon(\hat{L}_n, c(T))$, where $c(T)$ is a complexity measure of T, and $\hat{L}_n$ is obviously a measure of T over D_n. This approach has been taken by several authors although following different ways, see (Lugosi and Zeger, 1996) for example. As we are interested in the study of generalization error for decision trees, we will follow the approach described in (Golea et al., 1998; Mason et al., to appear), which is based on (Schapire et al., 1998). Another remarkable approach is described in (Scheffer, 2000). These kind of bounds on generalization error that express a tradeoff between the empirical error rate and the complexity of the classifier are often called *structural risk minimization* (SRM) bounds (Devroye et al., 1996; Vapnik, 1998). A theoretical framework for structural risk minimization can be found in (Shawe-Taylor, Bartlett, Williamson and Anthony, 1996), and a more detailed study on generalization bounds for decision trees can be found in (Mansour and McAllester, 2000).

Basically, all of these approaches use Υ as a bound for the true generalization error, that is,

$$L^* \leq \Upsilon(\hat{L}_n, c(T)),$$

and usually Υ is decomposed in two parts: the training error resubstitution estimate for a given threshold θ and a complexity term c which also depends on θ. This threshold $0 \leq \theta \leq 1$ is called the *margin*, and it is defined to be the confidence of the predictions made by the classification system. Therefore,

$$L^* \leq c\hat{L}_n^\theta + c(T, \theta) \tag{5.3}$$

where c is a constant[1] and $\hat{L}_n^\theta$ is defined as

$$\hat{L}_n^\theta \doteq \mathbf{P}_{D_n}\{yg(x) \leq \theta\} = \mathbf{P}_{D_n}\{yg(x) < 0\} + \mathbf{P}_{D_n}\{0 \leq yg(x) \leq \theta\}, \tag{5.4}$$

that is, the proportion of samples in the training set that are wrongly classified plus the proportion of samples in the training set which are correctly classified but with a margin lower than θ.

The main idea under this framework is that generalization error is a compromise between doing a good job in the training set and paying a penalty related to the intrinsic characteristics of the

[1] We are abusing of notation, here c represents both a constant and the complexity term.

classifier. As we will see in the following sections, the penalty term $c(T, \theta)$ grows inversely with θ, so we want θ as large as possible but keeping $\hat{L}_n^\theta$ as small as possible. As shown in (Mason, Bartlett and Baxter, 2000), it is possible to improve generalization error through the explicit optimization of the margin θ.

As pointed out in (Shawe-Taylor, 1997), there is indeed a relationship between the margin and the corresponding estimation of error probability, though it is not simply the margin that enters into the equation. Therefore, we must be careful when studying $\hat{L}_n^\theta$ defined as in Eq. (5.4), as we will see in the following sections, where the lack of a specific framework for decision trees yields to poor generalization error bounds.

The importance of generalization bounds is clear: in order to compare two algorithms for building classification systems, there are usually two relevant arguments: cost and performance. Cost involves both training and exploitation costs, but it is performance often the criterion used to choose among several classification systems. Although experimentation is usually the best method for choosing a classification system for a given data set, it would be useful to have a tool (the error generalization bound) for predicting the performance and then discarding those classification systems with a priori worse behavior, reducing training costs. Furthermore, a good generalization bound could even replace the pruning stage, see (Helmbold and Schapire, 1997) for example, or to be combined into it, as described in (Kearns and Mansour, 1998).

5.1.2 Vapnik-Chervonenkis dimension

Both the margin θ and the penalty cost $c(T, \theta)$ are related to the statistical learning theory developed by Vapnik and Chervonenkis (1971), which is fully described and enriched with practical examples in (Vapnik, 1998).

We will follow the definitions in (Devroye et al., 1996). First, we will briefly define the concepts of *shatter coefficient* and *Vapnik-Chervonenkis (VC) dimension*. Then, we will use the VC dimension as the basic tool to analyze the complexity of decision trees (that is, the form of the penalty term c) and, therefore, the generalization error bound.

Definition 5.1 *Let $\mathcal{A}$ be a collection of measurable sets. For $(z_1, \ldots, z_n) \in \{\mathbb{R}^d\}^n$, let $N_{\mathcal{A}}(z_1, \ldots, z_n)$ be the number of different sets in*

$$\{\{z_1, \ldots, z_n\} \cap A;\ A \in \mathcal{A}\}.$$

The n-th shatter coefficient of $\mathcal{A}$ is

$$s(\mathcal{A}, n) = \max N_{\mathcal{A}}(z_1, \ldots, z_n).$$

That is, the shatter coefficient is the maximal number of different subsets of n points that can be picked out by the class of sets $\mathcal{A}$.

Definition 5.2 *Let $\mathcal{A}$ be a collection of sets with $|\mathcal{A}| \geq 2$. The largest integer $k \geq 1$ for which $s(\mathcal{A}, k) = 2^k$ is denoted by $VC_{\mathcal{A}}$, and it is called the Vapnik-Chervonenkis dimension (or VC dimension) of the class $\mathcal{A}$. If $s(\mathcal{A}, n) = 2^n$ for all n, then by definition, $VC_{\mathcal{A}} = \infty$.*

Now we can define the generalization error bound using Theorem 2 in (Schapire et al., 1998), which is extended in (Golea et al., 1998) to include functions of several classes. We will use the latter because is better for our purposes:

Theorem 5.1 *Let $\mathcal{D}$ be a distribution on $X \times \{-1, 1\}$, $\mathcal{H}_1, \ldots, \mathcal{H}_k$ hypothesis classes with finite*[2] *$VC_{\mathcal{H}_i} = d_i$, and $\delta > 0$. With probability at least $1 - \delta$ over a training set D_n of n examples chosen according to $\mathcal{D}$, every function $g \in co\left(\cup_{i=1}^{k} \mathcal{H}_i\right)$ and every $\theta > 0$ satisfy both*

$$\mathbf{P}_{\mathcal{D}}\{yg(x) \leq 0\} \leq \mathbf{P}_{D_n}\{yg(x) \leq \theta\} + \mathcal{O}\left(\frac{1}{\sqrt{n}}\left(\frac{1}{\theta^2}(d \log n + \log k)\log(n\theta^2/d) + \log(1/\delta)\right)^{1/2}\right) \tag{5.5}$$

and

$$\mathbf{P}_{\mathcal{D}}\{yg(x) \leq 0\} \leq 2\mathbf{P}_{D_n}\{yg(x) \leq \theta\} + \mathcal{O}\left(\frac{1}{n}\left(\frac{1}{\theta^2}(d \log n + \log k)\log(n\theta^2/d) + \log(1/\delta)\right)\right) \tag{5.6}$$

where $d = \sum_i a_i d_{j_i}$ and the a_i and j_i are defined by $g = \sum_i a_i h_i$ and $h_i \in \mathcal{H}_{j_i}$, for $j_i \in \{1, \ldots, k\}$.

Both Eqs. (5.5) and (5.6) follow Eq. (5.3). Notice that $c = 1$ for Eq. (5.5) and $c = 2$ for Eq. (5.6), while the complexity term is better for the latter. For lower values of $\mathbf{P}_{D_n}\{yg(x) \leq \theta\}$, Eq. (5.6) is tighter. Nevertheless, it is accepted (Bottou, Cortes and Vapnik, 1994) that the traditional VC bound widely overestimates the generalization error. In the following sections we will describe several approaches to the generalization error for decision trees which try to overcome this problem.

5.2 Decision trees

Theorem 5.1 is general enough to include any function g satisfying the theorem conditions. When g is a decision tree, a bound which does not use the VC dimension but tree complexity measures may be derived from such theorem, as stated in (Golea et al., 1998). There are two theoretical results, depending on whether the training set is perfectly classified or not.

Let $\rho(P, U) = \sum_i^{|\tilde{T}|}(p_i - 1/|\tilde{T}|)^2$ be the quadratic distance between the probability vector $P = \{p_1, \ldots, p_{|\tilde{T}|}\}$ and the uniform probability vector $U = \{1/|\tilde{T}|, \ldots, 1/|\tilde{T}|\}$. Define the *effective number of leaves* of T as

$$N_e(T) \doteq |\tilde{T}|(1 - \rho(P, U)).$$

[2] missing in the original paper (Golea et al., 1998).

The parameter $N_e(T)$ is a complexity measure of T. We will denote it by N_e when no confusion is possible. Notice that $\rho(P,U)$ is close to zero when all leaves in tree define regions with similar probability. Therefore, N_e is almost $|\tilde{T}|$. On the other hand, if T has very small leaves or very large leaves, $\rho(P,U)$ is close to one (for large $|\tilde{T}|$) and N_e is close to zero.

5.2.1 Consistent decision trees

Suppose we use the growing algorithm with the whole data set and a perfect decision tree is created. Then, the following theorem applies:

Theorem 5.2 *For a fixed $\delta > 0$, there is a constant c that satisfies the following. Let $\mathcal{D}$ be a distribution on $X \times \{-1,1\}$. Consider the class of decision trees of depth up to $\overline{d}$, with decision functions in $\mathcal{U}$. With probability at least $1-\delta$ over the training set D_n (of size n), every decision tree T that is consistent[3] with D_n has*

$$\mathbf{P}_{\mathcal{D}}\{T(x) \neq y\} \leq c\left(\frac{N_e\, VC_{\mathcal{U}} \log^2 n \log \overline{d}}{n}\right)^{1/2} \tag{5.7}$$

where N_e is the effective number of leaves of T.

Therefore, tree performance depends on N_e, the VC dimension of the decision functions in $\mathcal{U}$, and its maximum depth $\overline{d}$. Suppose the training set D_n is fixed and n is large enough. Suppose also that both $VC_{\mathcal{U}}$ and $\overline{d}$ are known and small (we will discuss this later in this section). Then Eq. (5.7) is determined by N_e which can be much smaller than $2^{\overline{d}}$, depending on internal tree structure. This is the weakest part of Theorem 5.2, as it is difficult to accept that a single parameter (N_e) determines the whole generalization error estimation model. Furthermore, the fact that T must be consistent with D_n, which is not the usual case in a real scenario where pruning is used to avoid overfitting, makes this bound completely useless. Nevertheless, the idea behind Theorem 5.2 is very intuitive: unbalanced decision trees capture the unknown data set internal structure better than balanced trees, and small decision trees (with small N_e and $\overline{d}$) are preferable.

For orthogonal decision trees, $VC_{\mathcal{U}}$ is just 1 (see (Devroye et al., 1996, pp. 220), for example). $\mathcal{U}$ is used in the definition of $\mathcal{H}$ (the class of leaf functions for leaves up to depth $\overline{d}$) as

$$\mathcal{H}_{\overline{d}} = \{h : h = u_1 \wedge u_2 \wedge \ldots \wedge u_r | r \leq \overline{d}, u_i \in \mathcal{U}\}$$

then, using some simple results about shatter coefficients (see (Devroye et al., 1996, pp. 219)), it is easy to show that $VC_{\mathcal{H}_{\overline{d}}} \leq 2\overline{d} VC_{\mathcal{U}} ln(2e\overline{d})$. Using general hyperplanes as decision functions may reduce $\overline{d}$ (and N_e), but $VC_{\mathcal{U}} = d+1$, where d is data dimensionality. Therefore, the use of orthogonal hyperplanes is not a bad assumption under a theoretical point of view.

[3] Here consistency means $\hat{L}_n = 0$, that is, D_n is perfectly classified.

Nevertheless, this bound is completely useless for a practical use. It is only valid for $K = 2$, and T must be consistent with D_n, which is not the usual setting. The UCI data sets which fulfill the condition $K = 2$ used in this thesis are not large enough to keep the term $(\log^2 n/n)^{1/2}$ small.

5.2.2 General case

If T is not consistent with D_n (that is, $\hat{L}_n > 0$), then Theorem 5.2 is not valid. Define $Q_i = \mathbf{P}_{D_n}\{y\sigma_i = -1 | h_i(x) = 1\}$ (that is, the probability of making a mistake in leaf i) and $p'_i = p_i(1 - Q_i)/(1 - \mathbf{P}_{D_n}\{T(x) \neq y\})$. The effective number of leaves of T is now computed as $N'_e(T) \doteq |\tilde{T}|(1 - \rho(p', U))$. Then, the following theorem is valid:

Theorem 5.3 *For a fixed $\delta > 0$, there is a constant c that satisfies the following. Let $\mathcal{D}$ be a distribution on $X \times \{-1, 1\}$. Consider the class of decision trees of depth up to $\overline{d}$, with decision functions in $\mathcal{U}$. With probability at least $1 - \delta$ over the training set D_n (of size n), every decision tree T has*

$$\mathbf{P}_{\mathcal{D}}\{T(x) \neq y\} \leq \mathbf{P}_{D_n}\{T(x) \neq y\} + c\left(\frac{N'_e\, VC_{\mathcal{U}} \log^2 n \log \overline{d}}{n}\right)^{1/3} \tag{5.8}$$

where N'_e is the effective number of leaves of T.

Notice that both Eq. (5.7) and Eq. (5.8) coincide in the second term but in the $1/3$ exponent, which is worse than $1/2$. N'_e is a generalization of N_e, as N_e is a special case of N'_e when T is consistent with D_n. Each p'_i is a corrected version of the original p_i, weighted accordingly to leaf performance, and all p'_i are normalized by tree performance in order that they sum one. In order to keep N'_e low, p'_i must be far from uniform, so it would be interesting that large leaves (with large p_i) perform better than average, while small leaves (with low p_i) perform worse than average. This is consistent with the basic idea of progressive decision trees of type A (see Fig. 15 defined in Sect. 3.7): large regions which are easily classified must be kept as a single region, while small or too impure regions must be joined again trying to find new splits.

For progressive decision trees of type B (see Fig. 16 defined in Sect. 3.7), this reasoning is not valid, as decision trees are not sequentially ensembled creating a single decision tree. In this case, new classification features are added and a completely different decision tree is built. Experiments show that trees become smaller when the additional classification features are selected for splitting, thus reducing both N'_e and $\overline{d}$, but the term $VC_{\mathcal{U}}$ in Eq. (5.8) is larger. This is equivalent to using more complex internal node decision functions, such as general hyperplanes or hyperrectangles.

Fig. 33 shows the behavior of both error generalization bound terms defined by Eq. (5.8), ignoring the c constant factor, for the *pima* data set, the largest data set with $K = 2$ in the UCI collection. Notice that the first term decreases faster than the second one, showing that the smallest decision tree yielding a reasonable classification accuracy must be used, instead of building a large decision tree which classifies the training set better, but with a higher complexity (and computational

cost). Experiments also show that for normal data sets, N_e is not much smaller than $|\tilde{T}|$, specially when the entropy splitting criterion is used, because at the first stages leaves are split creating new leaves which are reasonably balanced. Other splitting criteria such as misclassification error, for example, produce more unbalanced leaves, but as shown previously, they yield to poorer decision trees, because $\overline{d}$ is usually too large.

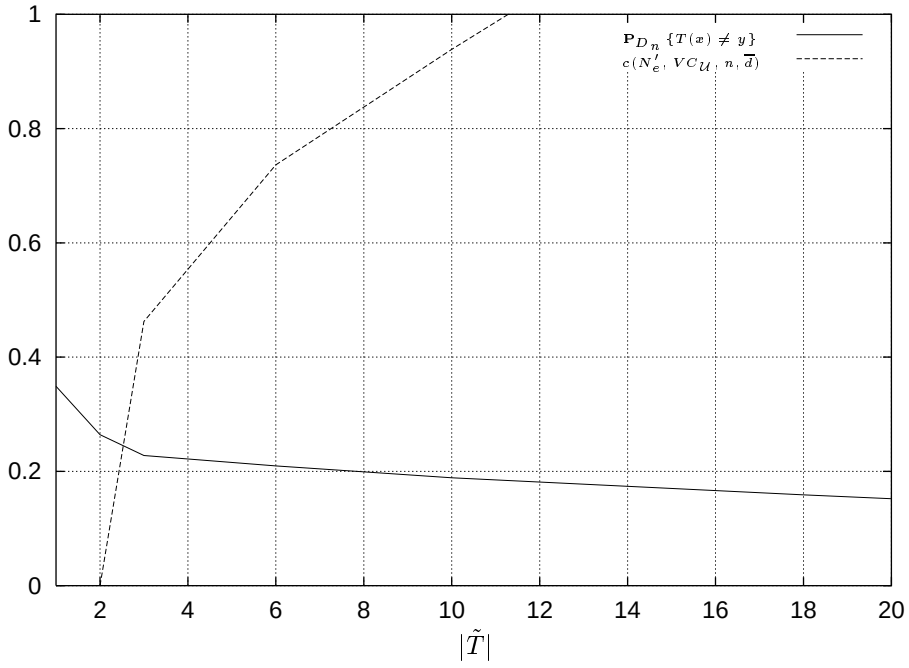

Figure 33: Error generalization bound for the *pima* data set.

Therefore, the main conclusion that may be drawn from this experiment is that, unless n is really large, the bound is completely useless for practical purposes, although it establishes that the decision tree should be as small as possible and also that it should have a skewed probability leaf distribution. Nevertheless, Theorem 5.3 does not help us to determine the parameters involved in the type A cascading ensemble (the number of trees t and the maximum depth $\overline{d}$ and the threshold ϵ for each tree).

5.2.3 Progressive decision trees

Using the sequential ensemble described in Sect. 3.6.1 (decision trees of type A), a cascading ensemble of small decision trees may be seed as a single tree, where each mixed leaf of the first tree has been replaced by a complete decision tree (the second tree in the cascading ensemble), and so on. This procedure clearly produces an unbalanced tree, as several leaves of the first decision tree

have a maximum depth limited by $\overline{d}_1$, and some of these leaves are replaced by a decision tree, causing the resulting tree to have several leaves with a maximum depth of $\overline{d}_1 + \overline{d}_2$. This process is repeated several times (t, to be more precise), so a final tree with leaves at different depths is produced. Experiments show that the total number of leaves of the resulting tree is usually smaller than the number of leaves of a classical decision tree which has been grown up to a maximum depth $\overline{d} = \overline{d}_1 + \ldots + \overline{d}_t$, as only small decision trees are combined, and only a few leaves are labeled as mixed and therefore replaced by a decision tree. Furthermore, the log term in Eq. (5.8) makes this parameter not so relevant for small values of $\overline{d}$, which is the typical case.

Therefore, when comparing the classification performance of a classical decision tree with maximum depth $\overline{d}$ and a cascading ensemble of t small decision trees summing up to the same maximum depth, Eq. (5.8) may be used as an objective criterion to determine which one of the two approaches is better. Experiments show that using an appropriate value for ϵ, it is possible to obtain a progressive decision tree with a classification performance similar to a classical decision tree, so N'_e is the only parameter that determines which approach may perform better. Usually, the combination of small decision trees produces a single tree smaller than the classical approach, because leaves that are pure enough (for a given ϵ) are never split, thus reducing N'_e.

Another possibility is to study the resulting decision tree as an acyclic graph, as described in Sect. 3.2, instead of as a decision tree. In this case N'_e is smaller, but the term $VC_{\mathcal{U}}$ is no longer valid in itself as the complexity of the decision functions in an acyclic graph is higher than in the tree case. The class $\mathcal{H}$ should take into account that it is possible to follow more than one path from the root to a leaf, and that n is no longer a constant across the whole acyclic graph but it varies depending on leaf index. Such study is far from the scope of this thesis, but it is an interesting research subject.

5.2.4 Using hyperrectangles

As stated in Sect. 3.4.1, progressive decision trees may be seen as a suboptimal way to build hyperrectangles, one side at each step. The resulting decision tree is much simpler, as decision functions in internal nodes become more complex. Thus, both N'_e and $\overline{d}$ may be reduced, but $VC_{\mathcal{U}}$ also increases as well, so no improvement is achieved at all.

As stated in (Devroye et al., 1996), when decision trees are built using hyperrectangles whose sides are jointly optimized, it is possible to show that they are consistent, although no experimental results are available. The basic idea described in (Devroye et al., 1996) is that consistency is achieved when *elementary* splits are used: a good splitting method is one that includes many (but not too many) small sets. For example, $d + 1$ general hyperplanes or $2d$ orthogonal hyperplanes jointly optimized form an elementary split. Type A progressive decision trees may be seen as a naive approach to create a $2d$ hyperrectangle.

5.3 Combined decision trees

In (Mason et al., to appear), an upper bound on the generalization error of any voted combination of binary decision trees in terms of the margin and the average complexity of the decision trees is described. As we will see in the following section, both type A and type B progressive decision trees may be also described as voted combination of decision trees, so the framework defined in (Mason et al., to appear) may be applied.

Given a decision tree T_i with $|\tilde{T}_i|$ leaves, let $\sigma_{ij} \in \{-1, +1\}$ denote the label associated and $h_{ij}(x)$ be the $\{0,1\}$-valued function producing 1 if and only if x reaches leaf j in T_i. Then, for any sequence of a_{ij} such that $a_{ij} > 0$ and $\sum_{j=1}^{|\tilde{T}_i|} a_{ij} = 1$ the output of the tree for an instance x can be written as

$$T_i(x) = \operatorname{sgn}\left(\sum_{j=1}^{|\tilde{T}_i|} a_{ij}\sigma_{ij}h_{ij}(x)\right).$$

The following theorem can be applied for any ensemble of decision trees which are combined as a thresholded convex combination, as follows

$$T(x) = \sum_{i=1}^{t} \alpha_i \operatorname{sgn}\left(f(x)\right).$$

Theorem 5.4 *There exists a constant c, such that with probability at least $1-\delta$ over random choice of the sample set D_n, every convex combination g of decision trees, $0 < \theta_0 \leq 1$ and $0 < \theta_i \leq 1$ satisfies*

$$\begin{aligned}\mathbf{P}_{\mathcal{D}}\{yg(x) \leq 0\} \leq 2\mathbf{P}_{D_n}\{yg(x) \leq \theta_0\} + \\ c\frac{1}{n}\left[\frac{\ln n}{\theta_0^2}\left(\sum_{i=1}^{t}\alpha_i \min(C_1, C_2) + C_3\right) + \ln(1/\delta)\right],\end{aligned} \tag{5.9}$$

where

$$\begin{aligned}C_1 &= \frac{\ln(n/\theta_0)}{\theta_i^2}\left(R_i V C_{\mathcal{U}} \ln n + ln(\max_l \overline{d}_l)\right) + \frac{n\theta_0}{\ln n}\sum_{j=1}^{|\tilde{T}_i|} P_{ij} 1_{\{P_{ij} < \theta_i\}},\\ C_2 &= |\tilde{T}_i|\left(V C_{\mathcal{U}} \ln n + ln(\max_l \overline{d}_l)\right),\\ C_3 &= \ln\left(\frac{t\ln(n/\theta_0)}{\min_i \theta_i^2}\right),\end{aligned}$$

and R_i is the average depth of T_i, $P_i = \{P_{ij}\}$ is a probability distribution over leaves via $P_{ij} = \mathbf{P}_{D_n}\{h_{ij}(x) = 1\}$, and $\mathcal{U}$ denotes the class of node decision functions.

The most remarkable fact that can be extracted from Theorem 5.4 is that θ_0 must be large but keeping the term $\mathbf{P}_{D_n}\{yg(x) \leq \theta_0\}$ as small as possible, and this depends on both classifier performance and the attainable margin, which depends on the weights α_i assigned to each decision

tree. If we split $\mathbf{P}_{D_n}\{yg(x) \leq \theta_0\}$ in $\mathbf{P}_{D_n}\{yg(x) < 0\} + \mathbf{P}_{D_n}\{0 \leq yg(x) \leq \theta_0\}$, and we suppose the first term is small, then we need the α_i to be skewed, in order to find a large value for θ_0.

Both C_1 and C_2 depend on each T_i internal tree structure. When trees are unbalanced and show a skewed P_i probability distribution, it is possible to choose θ_i in a manner that $C_1 < C_2$, keeping the second term of C_1 relatively small. For balanced trees, $C_1 > C_2$, and the latter represents the average complexity of each decision tree, that is, it is proportional to the sum of the complexities of the internal node decision functions. Regarding C_3, the log term makes it not so relevant, although it may become large if $\min_i\{\theta_i\}$ is really small, which is possible when t (the number of trees) is large.

5.3.1 Progressive decision trees

Each partial progressive decision tree may be seen as a classification rule T_i, and the final progressive decision tree T is therefore a combination of all the partial progressive decision trees. The weight assigned to each T_i determines its importance in the final ensemble.

Unlike the voting methods, which rank each base-level classifier according to its performance or uniformly, in this case we try to weight each decision tree according to its importance. In a cascading ensemble, importance is related to the order of appearance of each tree: for type A progressive decision trees, the first decision tree in the cascading ensemble is the most important one, the second decision tree is the second most important one, and so. The idea is that the outcome of the combined classification system is that from the first decision tree giving an outcome different than zero. On the other hand, type B progressive decision trees do exactly the opposite: the first decision tree gives an opinion that it may be used or not by the second decision tree, which also gives an opinion, which may be used or not by the third decision tree, and so. The last tree in the cascading ensemble generates the final output, so it is the most important one.

Suppose $K = 2$, but $Y \in \{-1, 0, +1\}$ and the mixed class is represented by 0, so $T_i(x) = 0$ if tree T_i classifies x as mixed. Then $T(x)$ may be written as

$$T(x) = \sum_{i=1}^{t} q_i T_i(x) \tag{5.10}$$

where q_i are set up accordingly to tree T_i importance. As only $T_i(x)$ is different from zero if x is classified by the i-th tree, one could set $q_i = \frac{1}{t}$ or study any other setting based of each decision tree performance (as in (Golea et al., 1998), for example) in order to maximize the margin. The problem is that this is not a true convex combination, as several decision trees (all T_j with $j > i$) are forced to produce 0 as their outcome for type A progressive decision trees.

In order to study progressive decision trees, the work of Schapire et. al (Schapire et al., 1998) and the further work of Golea et al. (Golea et al., 1998) should be extended to include a special class of functions $\mathcal{H}'$ which determines not only which leaf of the tree classifies the input vector (the $h_i(x)$

function), but also the fact that different training sets (and therefore, different trees with different values of n, N_e and $\overline{d}$) are being involved.

5.3.1.1 Type A progressive decision trees

Therefore, in order to use Theorem 5.4 as is, and to reflect the fact that the first decision tree is the most important and so (for type A progressive decision trees), the q_i must satisfy the following conditions:

$$\begin{aligned} q_i &> 0 \\ \sum_{i=1}^{t} q_i &= 1 \\ \sum_{i=j+1}^{t} q_i &< q_j \quad \forall j < t. \end{aligned} \tag{5.11}$$

For example, $q_i = \frac{2^{t-i}}{2^t-1}$. For $t = 2$, $q_1 = 2/3$ and $q_2 = 1/3$. For $t = 3$, $q_1 = 4/7$, $q_2 = 2/7$ and $q_3 = 1/7$. The main problem is that for large t, last trees have a relative importance too small, so $T(x)$ is also too small for those regions, which is not good under a margin analysis point of view, specially if the last decision tree is the most important one. This can be generalized to $q_i = (a-1)\frac{a^{t-i}}{a^t-1}$, $a \geq 2$. In this case, the larger a, the more skewed the $\{q_i\}$ are.

Another possibility is to define $q'_0 = 1$ and $q'_{2i+1} = \frac{1}{2}q'_{2i} + \delta$ and $q'_{2i+2} = \frac{1}{2}q'_{2i} - \delta$, $\delta > 0$, and then set $q_i = q'_{2i-1} \forall i < t$ and $q_t = q'_{2t-2}$. For example, if $t = 2$ this setup yields to $q_1 = \frac{1}{2} + \delta$ and $q_2 = \frac{1}{2} - \delta$. It is easy to see that with this setup, the q_i satisfy all the conditions described above. This procedure produces a more balanced $\{q_i\}$ distribution, controlled by the parameter δ.

The main problem of the convex combination framework is that all decision trees contribute to the final decision, which is not the reality in a cascading ensemble of progressive decision trees of type A. Even if we force the decision to be taken using only the most important decision tree (name it t_j) through a correct weight distribution, the margin may be low because all trees with index $i > j$ produce a random $T_j(x) \in \{-1, 0, +1\}$ which is also taken into account by Eq. (5.10) (trees with lower index have generated a 0 as outcome).

Therefore, it seems reasonable to have a skewed weight distribution in order to minimize the random effect of the decision trees which also contribute to the final decision, although they do not determine it. But this causes the last trees to have very small weights, specially when the number of trees is high, and that is not good from a margin analysis point of view. Golea et al. (Golea et al., 1998) used the probability of being at each leaf as its weight, which is very intuitive. If we want to use a similar criterion for progressive decision trees, then we should force the first tree to classify at least q_1 (proportionally to one) of the training set, that is, the first tree must do a lot of work, and do it very well. Depending on the training set, this is not possible unless T_1 is a large tree which almost classifies the whole training set, so it is like using $t = 1$ and not any ensemble is

used. If it is possible to build a small tree which classifies a large fraction of the training set with a limited misclassification error, then it is possible to use the remaining training samples (when n is large enough) to repeat this process, so $t > 1$. Unless n is really large, t cannot be very large, so we have here a criterion for t: it depends somehow on n, t must be small ($t = 2$ or $t = 3$) for small n. For large n, even if each T_i classifies a large fraction of data, we have data enough for the next stage, so t may be larger.

Regarding ϵ, it determines the first term in Eq. (5.4) as at each stage, $\mathbf{P}_{D_n}\{yg(x) \leq 0\} \leq \epsilon$, so the first term in Theorem 5.4 is partially bounded. The experiments described in Sect. 4.5.2 show that ϵ is a critical parameter which depends a lot on the intrinsic characteristics of the training set, and it is not suitable to be estimated but by a fine-tuning process.

5.3.1.2 Type B progressive decision trees

In this case, the third condition in Eq. (5.11) must be changed to reflect the opposite behavior, that is, the last tree is the most important one:

$$\sum_{i=1}^{j-1} q_i < q_j \quad \forall j < t.$$

The weight schemes described previously are also valid, but assigning them in reverse order, so the first decision tree will have the smallest weight, and the last decision tree the largest one.

In this case the original idea of margin in (Schapire et al., 1998) is preserved, as all decision trees contribute to the final decision, although only the last one will determine the outcome of the combined classifier. Suppose that all the decision trees in the cascading ensemble generate the same outcome. Then $T(x)$ will be -1 or $+1$, which is good under a margin analysis point of view.

As stated previously, it is interesting to have a large value for θ_0 but keeping $\mathbf{P}_{D_n}\{yg(x) \leq \theta_0\}$ low, so the q_i should be skewed, giving more importance to the final decision trees. Thus, trees should become more and more accurate at each stage, which is exactly the intuitive idea behind progressive decision trees.

5.4 Conclusions

In this chapter we have studied several theoretical frameworks for decision trees, trying to see how progressive decision trees may also be explained under such frameworks. Our goal was to give a plausible explanation to the question "why do progressive decision trees perform well, and when", at least partially. We also want to give some answers to the questions defined in Sect. 3.4, or at least some useful directions to determine the parameters involved in the cascading ensemble.

Type A progressive decision trees can be studied as a convex combination of decision trees and also as a single decision tree, while type B progressive decision trees can be though as a final decision

tree with more complex internal decision functions, but also as a convex combination reversing the order of importance with respect to type A progressive decision trees.

If we study type A progressive decision trees as a single decision tree, Theorem 5.3 shows that it is better to reduce effective tree size by building skewed and unbalanced trees. Experiments show that joining leaves which are too impure and creating a new decision tree, and then replacing each impure leaf by such decision tree, causes that the resulting decision tree is usually smaller and with a more skewed distribution, so both empirical results and theory support our ensemble of small decision trees.

On the other hand, if we study type A progressive decision trees as convex combinations of decision trees, Theorem 5.4 states that decision trees should perform accordingly to its importance, that is, its order of appearance. Therefore, we need a good first decision tree. Experiments described in Sect. 4.5.2 also show this behavior, it is impossible to build a good type A progressive decision tree unless the first decision tree in the ensemble is good enough. In fact, experiments show that sometimes it is not even possible to build a type A progressive decision tree better than a classical decision tree.

Regarding type B progressive decision trees, Theorem 5.3 states that performance may be improved as both the effective number of leaves and maximum depth become smaller, although the complexity term associated to the VC dimension of the internal decision functions increases, as new classification features (which are in fact more complex boundaries) are added to the training set. Studying them as a convex combination of decision trees, Theorem 5.4 establishes that decision trees should be more and more accurate at each stage, which is the basic idea of progressive decision trees.

The basic idea of this chapter is that progressive decision trees are a special case of cascading and that they can be studied as convex combinations of decision trees. Nevertheless, the lack of a specific theoretical framework for cascading makes all these approaches and generalization error bounds very poor. A new theory which focuses on cascading is therefore needed to explain the sequential ensembles of small decision trees or any other classifier. Such theoretical framework is far from the scope of this thesis, but it is an obvious research subject.

Chapter 6

Conclusions

Diplôme: Signe de science. Ne prove rien
Gustave Flaubert, *Le Dictionnaire des Idées Reçues*

In this chapter, the last of this thesis, we summarize all the results obtained from the study of decision trees and the possible ways to combine them into more complex classification systems under the cascading paradigm.

Different aspects related to decision trees have been covered: we have studied the parameters which determine the process of constructing decision trees, the importance of such parameters in classification accuracy, and the dependence of the learning algorithm on such parameters. We have also studied the problem of overfitting from a complexity point of view: instead of building a large (and expensive) decision tree, we have tried to split such process in a sequential ensemble of small decision trees. If one word was used to summarize this thesis, it should be cascading: the resulting classification system may be seen as a weighted cascading of decision trees, as shown in Fig. 34. Depending on the training set used at each stage, and the final weights assigned to the outcomes of the decision trees, we have one of the three types of cascading ensembles described in Sect. 3.7. Actually, this scheme is so general that may be used for any voting scheme (boosting and bagging).

The basic idea studied in this thesis is also described in Fig. 34: if the prediction of a classifier is accurate enough for a given input sample, use it. If not, pass such input sample and all the information computed during the classification process to the next classifier, and so on. The weighted scheme decides how much importance has each prediction in the final decision.

6.1 Thesis contributions

In this thesis, we have studied the combination of small decision trees for constructing ensembles which yield to more complex classification systems, using the cascading paradigm as a basis. The

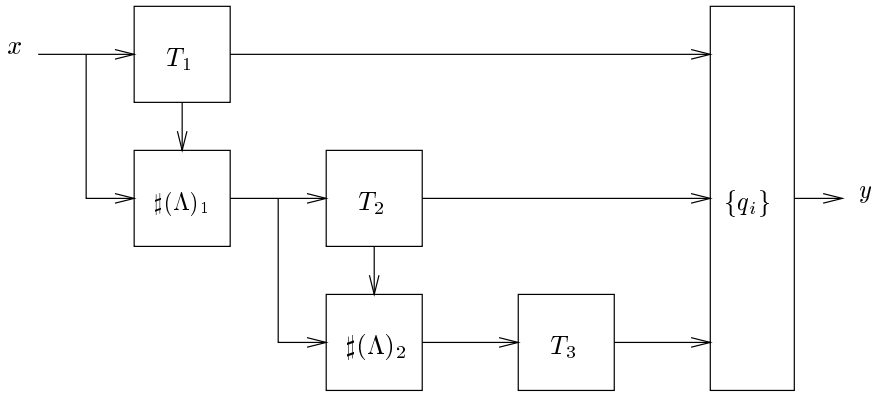

Figure 34: A progressive decision tree as a cascading ensemble of three decision trees.

main contributions of this thesis are:

1. We have extended the cascading paradigm using the concepts of partial classification and rejection: the training set is not only modified by adding new classification features computed during a classification stage, but also by removing those samples which have already been correctly classified for a given accuracy threshold or margin. Depending on how the training sets are built from one stage to the next, the three kinds of cascading ensembles (described in Sect. 3.7) may be easily described.

2. We have studied the parameters that determine the process of constructing decision trees, basically the splitting criterion, and the maximum depth the decision tree is allowed to achieve during the training stage, which is related to the stopping criterion. We have investigated the dependence of classification accuracy on these two parameters, using the bias-variance decomposition as the basic methodology for explaining the behavior of the learning algorithm. Our experiments show that it is possible to construct small decision trees with a reasonable classification accuracy, reducing both training and exploitation costs. This practical knowledge can also be used to construct classical decision trees: the entropy criterion and the family determined by the R-norm splitting criterion provide the best performance in average. The relationship between bias and variance may be used as a simple tool for determining early stopping criteria.

3. We have outlined a basic set of guidelines for constructing progressive decision trees: how many decision trees are needed, and which are the particular parameters of each decision tree in the ensemble. The experiments using the standard UCI data sets have shown that it is enough to use two or three decision trees to improve classification accuracy through cascading, and that, depending on the kind of cascading ensemble used, it is better to give more importance

through the weighted voting scheme to the first or the last decision tree. In general, cascading small decision trees improves classification accuracy fighting against both bias (because more complex boundaries can be constructed) and variance (because small decision trees are more robust), although depending on the kind of cascading ensemble, the classification cost is also increased. Nevertheless, it is possible to build progressive decision trees that perform similarly or even better than classical decision trees while reducing the classification cost, specially when the training set is large enough.

4. We have also studied our ensemble under a theoretical framework which not was specifically designed for cascading. We have used the reduced amount of bibliography related to generalization error bounds for decision trees and ensembles of decision trees to study the parameters of our ensemble. Results show that the basic intuitive ideas behind progressive decision trees (first, small decision trees perform reasonably accurately, and second, to break up the classification process in a sequence of small partial classification problems) are consistent with the current research in this subject. In theory, unbalanced decision trees have a better generalization error performance, and we have seen that progressive decision trees have usually such characteristic.

5. Last, but not least, a complete software package for building decision trees and different ensembles of decision trees (voting, stacking and cascading) has been developed. We expect to release this software to the public domain once a short documentation is written and a more developed user-friendly interface is available.

The main results of this thesis can be found in (Minguillón and Pujol, submitted). Some ideas have been also used in several experiments involving progressive decision trees in different projects: document layout recognition (Minguillón et al., 1999), hyperspectral imaging (Minguillón, Pujol, Serra and Ortuño, 2000; Minguillón, Pujol, Serra, Ortuño and Guitart, 2000), brain tumor classification (Tate et al., 2002; Minguillón, Tate, Arús and Griffiths, 2002; Minguillón, Tate, Griffiths, Majos, Pujol and Arús, 2002), and web mining (Mor and Minguillón, submitted).

6.2 Further research

Our future research will mainly focus on the new applications of progressive decision trees and in the search of better generalization error bounds for cascading ensembles of decision trees. Other interesting issues that still need more research are:

1. One of the main problems of classification using unstable classifiers such as decision trees, for example, is that the noise present in both the classification features and in the known labels has a large impact on classification accuracy. Lossy compression techniques using tree structured vector quantizers may be used as a basic tool for noise removal and data preprocessing,

improving classification results by building more robust classifiers. Nowadays we have an ongoing project related to hyperspectral imaging where trees are used for both classification and compression, using a modified distortion function and an adaptive splitting criterion.

2. Also related with the subject mentioned above, our experiments show that the entropy splitting criterion yields the best classification performance in average. Nevertheless, the R-norm splitting criterion may be tuned for a given classification problem, taking advantage of the different leaf behavior when descending in a decision tree: instead of using a fixed value for R, an adaptive approach could be exploited. We are currently working in a new version of the Kearns and Mansour (1999) paper but using the R-norm as a family of splitting functions. Then, the Brodley (1995) paper could be also reformulated using the appropriate value for R at each stage.

3. In this thesis, we have used orthogonal splits for creating the splitting hyperplanes used during the training stage. The use of general or multivariate splits usually improves classification accuracy, but increasing also cost. It is difficult to find the optimal hyperplane, because it involves optimizing a large number of variables (the classification features) at the same time. We are currently working on a greedy method based on variable importance that provides a simple way to compute multivariate hyperplanes. Some experiments which are not described in this thesis, show that it is possible to find suboptimal multivariate hyperplanes which outperform the orthogonal hyperplane which uses the most important variables in splitting importance.

4. Both the labelling rule and the policy of joining mixed regions are fixed and very simple. Currently, all mixed leaves are joined together, but another possibility could be to join regions depending on their class probabilities, trying to create large regions where there is a preponderance of one class. Our labelling rule is a simple way to force a minimum margin at each leaf, but more complex strategies involving other criteria such as leaf impurity, for example, or having a different threshold for each class, could be used.

5. Extracting information from data using a meta-theory is also a hot topic in the machine learning community. Knowing in advance when and how to build an ensemble method of weak learners instead of building a strong learner, is also an interesting research subject.

6. Finally, the generalization error bounds described in Chapter 5 are clearly related to the voting ensemble using classifiers where the margin has a real meaning. There is a real need of similar bounds for the cascading paradigm using small decision trees as base level classifiers.

In addition to the experiments mentioned above, we are currently starting to work on new fields where the use of very large data sets make the classical learning algorithms unfeasible, such as web mining, for example. In such project, millions of events produced by users accessing a web site

during a long period of time are analyzed trying to obtain patterns which can be used to identify user profile. Classification features are binary, as they indicate whether an user has produced a specific event or not. We are studying the use of progressive decision trees for building fast and simple classifiers to cluster users depending on the way they use the web site. The first experiments in this direction can be found in (Mor and Minguillón, submitted).

6.3 A final disquisition about learning

Finally, a few words about learning but from a more philosophical point of view. The question is: *what is learning?* A good definition for learning could be the process of putting knowledge in a framework which can be easily queried and updated. This definition seems to be valid for both human learning and machine learning. But, do machines really *learn*? Well, this is not a scientific problem, but a philosophical one. We are not able to give an answer to such question, because we use a restricted version for the definition of *learning*. For our purposes, Fig. 1 should be relabelled using "extract classification features from a given data set" instead of "perception", "compute a splitting hyperplane" instead of "reasoning", and "split the given data set in two subsets using the computed hyperplane" instead of "action". In this case, artificial intelligence becomes "computing decision trees".

In this thesis, we were not interested in the real meaning of the updating aspect of the learning process, although it is maybe even more important than the other aspects related to the definition of learning for real scenario classification systems. The basic idea of this thesis is that the easiest concepts must be learned first using simple classifiers, delaying the classification of the hardest concepts to further more complex classifiers. This sequence of classifiers may be seen as a progressive classification system where a first opinion is obtained at a low cost, and if such opinion does not satisfy a confidence criterion, a second opinion from a more complex expert is required, and so. This process partially reproduces the one humans follow in real life for making decisions. Therefore, we try to reproduce an intelligent behavior by partially mimicking humans behavior.

Regarding learning, the main difference between humans and algorithms is that humans put into context (or understand) what they are learning at a level higher than the level used for knowledge representation (neural connections or whatever). In machine learning, algorithms are completely connected to the structure used for knowledge representation, as the structure may be the learning algorithm itself (as in decision trees, for example). Unless the learning algorithm has the possibility of incorporating new tools for building new classification features from input data, it is not possible learn concepts such as the sum operation, for example. This is a consequence of the *No Free Lunch Theorem* (Theorem 1.1), as the universal classifier should use all possible decision functions available and all possible learning algorithms to be, in fact, universal.

Nevertheless, although we cannot say our classifiers learn in a complete sense of such word, they

show an intelligent behavior, as they solve a problem for which they were constructed. Therefore, it is very important to tune the learning algorithm for the given classification problem, as the more knowledge we have about the problem, the better our classification system will probably be, and the more complex our learning algorithm is, the better performance it will probably achieve. But we must always bear in mind the following quotation from H. H. Williams, "*furious activity is no substitute for understanding*".

Appendix A

UCI data sets for simulations

In this thesis we have used several data sets from the UCI Machine Learning repository (Blake and Merz, 1998), which form a more or less standard database in classification (although it would be more general to say in machine learning). These data sets may be found at:

- `http://www.ics.uci.edu/~mlearn/MLRepository.html`
- `http://ccd.uab.es/~julia/thesis/datasets` (mirror).

Some of the later have been preprocessed in order to homogenize their internal structure and simplify their use.

These data sets were selected among the whole list of available data sets because of two fundamental reasons: first, no missing values are present, and second, they only have binary and numerical features, but no categorical ones. Data set characteristics are listed in Tab. 7, while a brief description of the origin and main goal of these data sets and their peculiarities follows (the complete descriptions may be found in the links mentioned above):

1. *covtype:* this large data set (over half a million vectors) is part of a forest cover-type study. The overall objectives of this research were to compare and evaluate classification accuracy when predicting forest cover types in undisturbed forests.

2. *glass:* a study of classification of types of glass motivated by criminological investigation done in the USA Forensic Science Service, using physical properties and chemical compounds of pieces of glass investigated.

3. *ionosphere:* this radar data set was created collecting data from a phased array of 16 high-frequency antennas. The targets were free electrons in the ionosphere. This data set has been preprocessed using autocorrelation functions.

4. *iris:* it is probably the best known data set to be found in the pattern recognition literature, based on the work of R.A. Fisher, as stated in Duda et al. (Duda and Hart, 1973). There are three classes of iris plants, one of them is linearly separable from the other two, and the latter are not linearly separable from each other.

5. *letter:* the objective is to identify each of a large number of black and white rectangular pixel displays as one of the 26 capital letters in the English alphabet. This large data set has been extensively used by the machine learning community due to its intrinsic characteristics: a large size and a high number of classes.

6. *liver:* a study of the sensitivity to liver disorders that might arise from excessive alcohol consumption, using measures obtained through blood analysis and other information related to the patient.

7. *lrs:* this data set is part of the IRAS (Infra-Red Astronomy Satellite) Low Resolution Spectrometer Database. This database was originally obtained for use in development and testing a system called AutoClass for Bayesian classification.

8. *optdigits:* the main goal of this data set is to construct a system capable of optical recognition of handwritten digits. Each digit is a bitmap of 32×32 pixels which is decomposed into non-overlapping blocks of 4×4, and then the number of pixels are counted in each block, so all 64 elements of each input vector are integer numbers ranging from 0 to 16.

9. *page:* this data set is used to build a classifier for all the blocks that have been detected by the segmentation process of a document layout recognition system. Some outliers (mislabeled samples in the training set) were detected in this data set, although that may be caused by an insufficient set of classification features or by class definition ambiguity.

10. *pendigits:* in this case, the main goal is also to recognize handwritten digits but using a pen-based system, instead of using normalized bitmaps. Such a system measures pen coordinates and pressure, although pressure values are not present in the data set.

11. *pima:* this data set contains information about patients of Pima Indian heritage, trying to obtain a diagnostic which is whether a patient shows signs of diabetes or not. All patients were females at least 21 years old.

12. *sat:* this data set is a sub-area of a Landsat MSS scene, containing spectral information which is used to discriminate among several terrain classes. Surprisingly, there is a class called "mixture" which is not present because the authors doubt about its validity.

13. *segmentation:* this data set has been created taking random instances from a database of outdoor images, which were hand-segmented in order to create a classification for each pixel.

Each instance is then processed computing several continuous attributes which are used for classification purposes.

14. *sonar:* this is a data set used in a study of the classification of sonar signals using a neural network, trying to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock.

15. *thyroid:* in this study five laboratory tests are used to try to predict whether the thyroid of a patient is euthyroidism, hypothyroidism or hyperthyroidism. The diagnosis was based on a complete medical record including anamnesis, scan information and so on.

16. *vehicle:* this data set comes from the Turing Institute, Glasgow, Scotland, and its main goal is to classify a given silhouette as one of four types of vehicle using a set of features extracted from the silhouette, when viewed from one of many different angles.

17. *vowel:* this data set is part of a context sensitive learning work, and its main goal is to recognize one of the eleven possible classes of vowel sound in English using then classification features and contextual information on the gender of the speaker and his or her identity. We removed speaker identity information in order to avoid a highly overtrained classifier.

18. *waveform:* constructed by Breiman et al. (Breiman et al., 1984, pp. 33-44), it is a three class problem, linearly separable, although a small amount of noise is added in order to make it more difficult. According to (Breiman et al., 1984), the Bayes risk of misclassification error is 0.14, although it is difficult to achieve such accuracy using only hyperplanes (TSVQ achieves such accuracy, for example).

19. *wdbc:* based on a study named Wisconsin Diagnostic Breast Cancer, this data set is linearly separable using all input classification features, which describe characteristics of the cell nuclei present in a digitized image of a fine needle aspirate of a breast mass.

20. *wine:* this data set is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. Classes are non-overlapping, so a 100% of correct classification percentage could be achieved using a more complex version of linear discriminant analysis.

Appendix B

Notation

Symbol	Definition	First apparition
$1_{\{A\}}$	indicator function of event A	22
$\sharp$	transpose concatenation operator	50
α	pruning algorithm complexity cost measure	27
Δ_K	K-ary probability distribution	16
$\Delta_T(s,t)$	tree T impurity decrease for split s at node t	19
ϵ	mixed class threshold	32
Λ^i	additional information operator associated to T_i	50
θ	margin	119
$\phi_{i,j}$	error correlation between two classifiers g_i and g_j	53
$\overline{\phi}_{i,j}$	mixed class correlation between two classifiers g_i and g_j	54
Φ	impurity function	17
$\Psi_{\mathcal{P}_T}$	random variable induced by $\mathcal{P}_T$	14
π_j	*a priori* probability of class j	18
Υ	regularization function	119
$\mathcal{A}$	collection of measurable sets	120
$\hat{B}$	average bias decomposition	56
c	each internal node has exactly c children	11
$c(T,\theta)$	penalty cost	120
$C(j,k)$	misclassification cost matrix	14
C_m	corpus set containing m vectors	24
d	input space dimension, number of classification features	13
$\overline{d}$, $\overline{d}(T)$	tree maximum depth	22
D_n	training set containing n vectors	2

$D_{n\overline{i+1}}^{(\overline{i+1})}$	vectors classified as mixed at stage $i+1$	50
$\mathbf{E}\{A\}$	expectation of event A	14
$\mathbf{E}_{\mathcal{D}}\{A\}$	expectation of event A under distribution $\mathcal{D}$	14
$\mathbf{E}_{D_n}\{A\}$	expectation of event A measured in a training set D_n	14
$\mathcal{F}$	class of functions	2
$f(x)$	unknown true class function for an input vector x	2
$g_n^i(f\|D_n)$	learning algorithm which uses D_n and $f \in \mathcal{F}$	2
$\mathcal{H}$	set of $\{-1,1\}$-valued functions	118
$i(t)$	node t impurity	18
$I(T)$	tree T impurity	18
K	number of classes	13
L^*	optimal misclassification error, Bayes error	17
L_n	true misclassification error	22
$\hat{L}_n$	training set resubstitution estimate	22
$\hat{L}_n^\theta$	training set margin	119
$\hat{L}_{n,m}$	corpus set resubstitution estimate	24
$l(t)$	label for leaf t	14
$l'(t)$	label for leaf t using the mixed class labeling rule	32
M	label for mixed class	32
$N_e(T)$, N_e	Effective number of leaves of tree T	121
$\mathbf{P}\{A\}$	probability of event A	14
$\mathbf{P}_{\mathcal{D}}\{A\}$	probability of event A under distribution $\mathcal{D}$	121
$\mathbf{P}_{D_n}\{A\}$	probability of event A measured in a training set D_n	119
$p(t)$	probability of being at node t	14
p_i	probability of being at leaf i	118
$p(j\|t)$	probability of class j given a node t	18
p_j	probability of class j	16
P_T	percentage of input vectors classified by tree T	35
$\mathcal{P}_T$	partition induced by tree T	14
R	R-norm splitting criterion parameter	17
R, $R(T)$	tree average rate	22
$r(t)$	node t misclassification cost	14
$S(T)$	number of pruned subtrees of T	26
S_k	matrix covariance for class k	55
S_{k_1,k_2}	Pooled matrix covariance for classes k_1 and k_2	55
T	classification or decision tree	11
$\tilde{T}$	set of leaves of T	12

$\lvert\tilde{T}\rvert$	number of leaves of T	12
t	number of trees in a cascading ensemble	39
$VC_{\mathcal{A}}$	VC dimension of $\mathcal{A}$	121
$\hat{V}$	average variance decomposition	56
x	input vector	3
$\overline{x}_k$	class centroid for class k	55
(x, y)	pair of vector, label	3
(X, Y)	joint distribution of pairs (vector, label)	3

Bibliography

Ali, K. and Pazzani, M. (1996). Error reduction through learning multiple descriptions, *Machine Learning* **24**(1).

Allwein, E. L., Schapire, R. E. and Singer, Y. (2000). Reducing multiclass to binary: A unifying approach for margin classifiers, *Proceedings of the Seventeenth International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, pp. 9–16.

Alpaydin, E. and Kaynak, C. (1998). Cascading classifiers, *Kybernetika* **34**(4): 369–374.

Andersen, E. B. (1997). *Introduction to the statistical analysis of categorical data*, Springer.

Antos, A., Devroye, L. and Györfi, L. (1999). Lower bounds for Bayes error estimation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **21**(7): 643–645.

Auer, P., Holte, R. C. and Maass, W. (1995). Theory and applications of agnostic PAC-learning with small decision trees, *Proceedings of the 12th International Conference on Machine Learning*, Morgan Kaufmann, pp. 21–29.

Auer, P., Long, P. M. and Srinivasan, A. (1997). Approximating hyper-rectangles: Learning and pseudo-random sets, *Technical Report NC-TR-97-024*, Royal Holloway, University of London.

Avilés-Cruz, C., Guérin-Dugué, A., Voz, J. L. and Cappel, D. V. (1995). ELENA: Enhanced learning for evolutive neural architecture,
Available at `ftp://satie.dice.ucl.ac.be`.

Baram, Y. (1998a). Partial classification can be beneficial even for ideal separation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**(10): 1117.

Baram, Y. (1998b). Partial classification: The benefit of deferred decision, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**(8): 769–776.

Baras, J. S. and Dey, S. (1999). Combined compression and classification with learning vector quantization, *IEEE Transactions on Information Theory* **45**(6): 1911–1920.

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting and variants, *Machine Learning* **36**: 105–139.

Baulies, X. and Pons, X. (1995). Approach to forestry inventory and mapping by means of multispectral airbone data, *International Journal of Remote Sensing* **16**(1): 61–80.

Bay, S. D. (1999). The UCI KDD archive,
Available at `http://kdd.ics.uci.edu`.

Bennett, K. P. and Blue, J. A. (1998). A support vector machine approach to decision trees, *Proceedings of the IEEE International Conference on Neural Networks*, Anchorage, Alaska, pp. 2396–2401.

Bishop, C. M. (ed.) (1998). *Neural Networks and Machine Learning*, Vol. 168 of *Series F: Computer and Systems Sciences*, Springer-Verlag.

Blake, C. and Merz, C. (1998). UCI repository of machine learning databases,
Available at `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Blum, A. (1997). Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain, *Machine Learning* **26**: 5–23.

Boekee, D. E. and der Lubbe, J. C. A. V. (1980). The R-norm information measure, *Information and Control* **45**: 136–155.

Bottou, L., Cortes, C. and Vapnik, V. (1994). On the effective VC dimension,
Available at `http://citeseer.nj.nec.com/126486.html`.

Bottou, L. and Vapnik, V. N. (1992). Local learning algorithms, *Neural Computation* **4**(6): 888–900.

Breiman, L. (1993). Hinging hyperplanes for regression, classification, and function approximation, *IEEE Transactions on Information Theory* **IT-39**(3): 999–1013.

Breiman, L. (1996a). Bagging predictors, *Machine Learning* **24**(2): 123–140.

Breiman, L. (1996b). Bias, variance and arcing classifiers, *Technical Report 460*, Statistics Department, University of California, Berkeley, CA, USA.

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984). *Classification and Regression Trees*, Wadsworth International Group.

Brodley, C. E. (1995). Automatic selection of split criterion during tree growing based on node location, *Proceedings of the Twelfth International Machine Learning Conference*, Tahoe City, CA, pp. 73–80.

Brodley, C. E. and Utgoff, P. E. (1992). Multivariate versus univariate decision trees, *Technical Report 92-8*, University of Massachusetts, Amherst, MA 01003.

Brown, D. R. and Pittard, C. L. (1996). Classification trees with optimal multivariate decision nodes, *Pattern Recognition Letters* **17**(7): 699–703.

Buntine, W. and Niblett, T. (1992). A further comparison of splitting rules for decision-tree induction, *Machine Learning* **8**: 75–85.

Casey, R. G. and Nagy, G. (1984). Decision tree design using a probabilistic model, *IEEE Transactions on Information Theory* **30**(1).

Chan, P. K. and Stolfo, S. J. (1995). Learning arbiter and combiner trees from partitioned data for scaling machine learning, *Proceedings of the International Conference on Knowledge Discovery and Data Mining*.

Chou, P. A. (1991). Optimal partitioning for classification and regression trees, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-13**(4): 340–354.

Chou, P. A., Lookabaugh, T. and Gray, R. M. (1989). Optimal pruning with applications to tree-structured source coding and modeling, *IEEE Transactions on Information Theory* **IT-35**(2): 299–315.

Cortes, C. and Vapnik, V. N. (1995). Support-vector networks, *Machine Learning* **20**(3): 273–297.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*, John Wiley & Sons.

Devroye, L., Györfi, L. and Lugosi, G. (1996). *A Probabilistic Theory of Pattern Recognition*, Springer-Verlag.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, Boosting, and randomization, *Machine Learning* **40**(2): 139–157.

Dietterich, T. G. and Kong, E. B. (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms, *Technical report*, Dept. of Computer Science, Oregon State University, Corvall, Oregon, USA.

Domingos, P. (2000). A unified bias-variance decomposition and its applications, *Proceedings of the Seventeenth International Conference on Machine Learning*, Morgan Kaufmann, Stanford, CA, USA, pp. 231–238.

Drucker, H. (1999). Z splitting criterion for growing trees and boosting, *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC.

Drucker, H. (2000). Effect of pruning and early stopping stopping on performance of a boosted ensemble, *Proceedings of the International Meeting on Nonlinear Methods and Data Mining*, Rome, Italy, pp. 26–40.

Drucker, H. and Cortes, C. (1996). Boosting decision trees, *in* D. S. Touretzky, M. C. Mozer and M. E. Hasselmo (eds), *Advances in Neural Information Processing Systems*, Vol. 8, The MIT Press, pp. 479–485.

Drummond, C. and Holte, R. C. (2000). Exploiting the cost (in)sensitivity of decision tree splitting criteria, *Proceedings of the Seventeenth International Conference on Machine Learning*, Morgan Kaufmann, Stanford, CA, USA, pp. 239–246.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*, John Wiley & Sons.

Duda, R. O., Hart, P. E. and Stork, D. G. (2000). *Pattern Classification*, second edn, John Wiley & Sons.

Efron, B. and Tibshirani, R. J. (1993). *An introduction to the bootstrap*, Chapman and Hall, New York, NY, USA.

Frank, E., Wang, Y., Inglis, S., Holmes, G. and Witten, I. (1998). Using model trees for classification, *Machine Learning* **32**: 63–76.

Freund, Y. (1995). Boosting a weak learning algorithm by majority, *Information and Computation* **121**(2): 256–285.

Freund, Y. and Schapire, R. E. (1999). A short introduction to boosting, *Journal of Japanese Society for Artificial Intelligence* **14**(5): 771–780.

Freund, Y., Schapire, R. E., Singer, Y. and Warmuth, M. K. (1997). Using and combining predictors that specialize, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pp. 334–343.

Friedman, J. H. (1997). On bias, variance, 0/1-loss, and the curse of dimensionality, *Data Mining and Knowledge Discovery* **1**: 55–77.

Friedman, J. H., Kohavi, R. and Yun, Y. (1996). Lazy decision trees, *in* H. Shrobe and T. Senator (eds), *Proceedings of the thirteenth national conference on artificial intelligence and the eighth innovative applications of artificial intelligence conference*, AAAI Press, Menlo Park, CA, pp. 717–724.

Gama, J. (1999). Discriminant trees, *Proceedings of the Sixteenth International Conference on Machine Learning*, Morgan Kaufmann, Bled, Slovenia.

Gama, J. and Brazdil, P. (2000). Cascade generalization, *Machine Learning* **41**(3): 315–343.

Gavin, G., Puzenat, D. and Zighed, D. (1999). Classifiers with low decision-making error using linear combination of functions, *Proceedings of the International Conference on Artificial Intelligence*, Las Vegas, NE, USA.

Gelfand, S. B., Ravishankar, C. S. and Delp, E. J. (1991). An iterative growing and pruning algorithm for classification tree design, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-13**(2): 163–174.

Gersho, A. (1982). On the structure of vector quantizers, *IEEE Transactions on Information Theory* **IT-28**(2): 157–166.

Gersho, A. and Gray, R. M. (1992). *Vector Quantization and Signal Compression*, Communications and Information Theory, Kluwer Academic Publishers, Norwell, MA, USA.

Goetz, A. F. H. and Herring, M. (1989). The high resolution imaging spectrometer (HIRIS) for EOS, *IEEE Transactions on Geoscience and Remote Sensing* **27**: 136–144.

Golea, M., Bartlett, P., Lee, W. S. and Mason, L. (1998). Generalization in decision trees and DNF: Does size matter?, *in* M. I. Jordan, M. J. Kearns and S. A. Solla (eds), *Advances in Neural Information Processing Systems*, Vol. 10, The MIT Press.

Hartmann, C. R. P., Varshney, P. K., Mehrotra, K. G. and Gerberich, C. L. (1982). Application of information theory to the construction of efficient decision trees, *IEEE Transactions on Information Theory* **28**(3): 565–577.

Hastie, T., Tibshirani, R. and Friedman, J. H. (2001). *The elements of statistical learning. Data mining, inference and prediction*, Springer series in statistics, Springer.

Heath, D. (1992). *A geometric framewwork for machine learning*, PhD thesis, The Johns Hopkins University, Baltimore, MD, USA.

Helmbold, D. P. and Schapire, R. E. (1997). Predicting nearly as well as the best pruning of a decision tree, *Machine Learning* **27**(1): 51–68.

Ho, T. K. (1998). The random subspace method for constructing decision forests, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**(8): 832–844.

Ho, T. K., Hull, J. J. and Srihari, S. N. (1994). Decision combination in multiple classifier system, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**(1): 66–75.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets, *Machine Learning* **11**: 63–91.

Hyafil, L. and Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete, *Information Processing Letters* **5**(1): 15–17.

Hyvärinen, A., Karhunen, J. and Oja, E. (2001). *Independent Component Analysis*, Wiley series on adaptive and learning systems for signal processing, communications and control, John Wiley & Sons.

Jain, A. K., Duin, R. P. and Mao, J. (2000). Statistical pattern recognition: A review, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(1): 4–37.

Jain, A. K. and Zongker, D. (1997). Feature selection: evaluation, application and small sample performance, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(2): 153–158.

Jobson, J. D. (1992). *Applied multivariate data analysis. Volume II: categorical and multivariate methods*, Springer-Verlag.

Jordan, M. I. (1994). A statistical approach to decision tree modeling, *Proceedings of the 7th ACM Conference on Computational Learning Theory*, Morgan Kaufmann, New Brunswick, NJ, USA, pp. 13–20.

Kaynak, C. and Alpaydin, E. (2000). Multistage cascading of multiple classifiers: One man's noise is another man's data, *Proceedings of the Seventeenth International Conference on Machine Learning*, Morgan Kaufmann, Stanford, CA, USA.

Kearns, M. J. and Schapire, R. E. (1994). Efficient distribution-free learning of probabilistic concepts, *Journal of Computer and System Sciences* **48**(3): 464–497.

Kearns, M. and Mansour, Y. (1998). A fast, bottom-up decision tree pruning algorithm with near-optimal generalization, *Proceedings of the 15th International Conference on Machine Learning*, Morgan Kaufmann, pp. 269–277.

Kearns, M. and Mansour, Y. (1999). On the boosting ability of top-down decision tree learning algorithms, *Journal of Computer and System Sciences* **58**: 109–128.

Kim, B. and Landgrebe, D. A. (1991). Hierarchical classifier design in high-dimensional numerous class cases, *IEEE Transactions on Geoscience and Remote Sensing* **29**(4): 518–528.

Kittler, J., Hatef, M., Duin, R. P. W. and Matas, J. (1998). On combining classifiers, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**(3): 226–239.

Kittler, J. and Roli, F. (eds) (2000). *Multiple Classifier Systems*, Vol. 1857 of *Lecture Notes in Computer Science*, Springer, Cagliari, Italy.

Kittler, J. and Roli, F. (eds) (2001). *Multiple Classifier Systems*, Vol. 2096 of *Lecture Notes in Computer Science*, Springer, Cambridge, UK.

Kittler, J. and Roli, F. (eds) (2002). *Multiple Classifier Systems*, Vol. 2364 of *Lecture Notes in Computer Science*, Springer, Cagliari, Italy.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA, USA, pp. 1137–145.

Kohavi, R. and Wolpert, D. H. (1996). Bias plus variance decomposition for zero-one loss functions, *in* L. Saitta (ed.), *Machine Learning: Proceedings of the thirteenth international conference*, Morgan Kaufmann, pp. 275–283.

Kohonen, T. (1995). *Self-Organizing Maps*, Vol. 30 of *Springer Series in Information Sciences*, Springer, Berlin, Heidelberg.

Kononenko, I. (1993). Inductive and bayesian learning in medical diagnosis, *Applied Artificial Intelligence* **7**(4): 317–337.

Kudo, M. and Sklansky, J. (2000). Comparison of algorithms that select features for pattern classifiers, *Pattern Recognition* **33**: 25–41.

Kulkarni, S. R., Lugosi, G. and Venkatesh, S. S. (1998). Learning pattern classification - a survey, *IEEE Transactions on Information Theory* **44**(6): 2178–2206.

Li, W.-H. (1997). *Molecular Evolution*, Sinauer Associates, Inc.

Linde, Y., Buzo, A. and Gray, R. M. (1980). An algorithm for vector quantizer design, *IEEE Transactions on Communications* **COM-28**(1): 84–95.

Liu, H. and Setiono, R. (1998). Feature transformation and multivariate decision tree induction, *Proceedings of the 1st International Conference on Discovery Science*, Springer-Verlag, Fukuoka, Japan, pp. 279–290.

Loh, W. Y. and Vanichsetakul, N. (1988). Tree-structured classification via generalized discriminant analysis, *Journal of the American Statistical Association* **83**: 715–728.

López de Màntaras, R. (1991). A distance-based attribute selection measure for decision tree induction, *Machine Learning* **6**(1): 81–92.

Lugosi, G. and Zeger, K. (1996). Concept learning using complexity regularization, *IEEE Transactions on Information Theory* **42**(1): 48–54.

Mansour, Y. and McAllester, D. (2000). Generalization bounds for decision trees, *Proceedings of the 13th Annual Conference on Computational Learning Theory*, Morgan Kaufmann, San Francisco, pp. 69–80.

Martin, J. K. (1997). An exact probability metric for decision tree splitting, *Machine Learning* **28**: 257–291.

Mason, L., Bartlett, P. and Baxter, J. (2000). Improved generalization through explicit optimization of margins, *Machine Learning* **38**(3): 243–255.

Mason, L., Bartlett, P. L. and Golea, M. (to appear). Generalization error of combined classifiers, *Journal of Computer and System Sciences* .

McLachlan, G. J. (1992). *Discriminant Analysis and Statistical Pattern Recognition*, Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons.

McLean, G. F. (1993). Vector quantization for texture classification, *IEEE Transaction on Systems, Man, and Cybernetics* **23**(3).

Mingers, J. (1989a). An empirical comparison of pruning methods for decision tree induction, *Machine Learning* **4**: 227–243.

Mingers, J. (1989b). An empirical comparison of selection measures for decision-tree induction, *Machine Learning* **3**: 319–342.

Minguillón, J. and Pujol, J. (submitted). On cascading small decision trees, *Knowledge Discovery and Data Mining* .

Minguillón, J., Pujol, J., Serra, J. and Ortuño, I. (2000). Influence of lossy compression on hyperspectral image classification, *Proceedings of Data Mining'2000*, Cambridge, UK, pp. 545–554.

Minguillón, J., Pujol, J., Serra, J., Ortuño, I. and Guitart, P. (2000). Adaptive lossy compression and classification of hyperspectral images, *Proceedings of Image and Signal Processing for Remote Sensing VI*, Vol. 4170, Barcelona, Spain, pp. 214–225.

Minguillón, J., Pujol, J. and Zeger, K. (1999). Progressive classification scheme for document layout recognition, *SPIE Proceedings, Mathematical Modeling, Bayesian Estimation, and Inverse Problems*, Vol. 3816, Denver, CO, pp. 241–250.

Minguillón, J., Tate, A. R., Arús, C. and Griffiths, J. R. (2002). Classifier combination for *in vivo* magnetic resonance spectra of brain tumours, *in* F. Roli (ed.), *Multiple Classifier Systems*, Vol. 2364 of *Lecture Notes in Computer Science*, Springer, Cagliari, Italy, pp. 282–292.

Minguillón, J., Tate, A. R., Griffiths, J. R., Majos, C., Pujol, J. and Arús, C. (2002). A multistage classifier for *in vivo* magnetic resonance spectra of brain tumors, *MAGMA, to appear*, Cannes, France.

Mitchell, T. M. (1997). *Machine Learning*, McGraw-Hill.

Mor, E. and Minguillón, J. (submitted). An empirical evaluation of classifier combination schemes for predicting user navigational behavior, *Proceedings of the VIII Conferencia Iberoamericana de Inteligencia Artificial*.

Murphy, P. M. and Pazzani, M. J. (1994). Exploring the decision forest: An empirical investigation of Occam's razor in decision tree induction, *Journal of Artificial Intelligence Research* **1**: 257–275.

Murthy, S. K. (1997). *On growing better decision trees from data*, PhD thesis, The Johns Hopkins University, Baltimore, MD, USA.

Nobel, A. B. (1994). Greedy growing of tree-structured classification rules using a composite splitting criterion, *Proceedings of the 1994 Information Theory and Statistics Workshop*, Alexandria, Virginia, USA.

Oelher, K. L. and Gray, R. M. (1995). Combining image compression and classification using vector quantization, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-17**(5): 461–473.

Oliver, J. J. (1993). Decision graphs - an extension of decision trees, *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pp. 343–350. Extended version available as TR 173, Dept. of Computer Science, Monash University, Clayton, Victoria 3168, Australia.

Oliver, J. J. and Hand, D. J. (1995). On pruning and averaging decision trees, *Proceedings of the 12th International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, USA, pp. 430–437.

Ortega, J., Koppel, M. and Argamon, S. (2001). Arbitrating among competing classifiers using learned referees, *Knowledge and Information Systems* **3**(4): 470–490.

Pagallo, G. and Haussler, D. (1990). Boolean feature discovery in empirical learning, *Machine Learning* **5**: 71–99.

Pennebaker, W. B. and Mitchell, J. L. (1993). *JPEG still image data compression standard*, Van Nostrand Reinhold.

Pereira, F. C. and Singer, Y. (1999). An efficient extension to mixture techniques for prediction and decision trees, *Machine Learning* **36**(3): 183–199.

Perlmutter, K. O., Perlmutter, S. M., Gray, R. M., Olshen, R. A. and Oehler, K. L. (1996). Bayer risk weighted vector quantization with posterior estimation for image compression and classification, *IEEE Transactions on Image Processing* **IP-5**(2): 347–360.

Prodromidis, A. L. and Stolfo, S. J. (2001). Cost complexity-based pruning of ensemble classifiers, *Knowledge and Information Systems* **3**(4): 449–469.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games, *in* R. S. Michalski, J. G. Carbonell and T. M. Mitchell (eds), *Machine Learning: an artificial intelligence approach*, Morgan Kaufmann, San Francisco, CA, USA, pp. 463–482.

Quinlan, J. R. (1990). Decision trees and decisionmaking, *IEEE Transaction on Systems, Man, and Cybernetics* **20**(2): 339–346.

Quinlan, J. R. (1992). *C4.5: Programs for Machine Learning*, Morgan Kaufmann.

Quinlan, J. R. (1996a). Bagging, boosting, and C4. 5, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, AAAI Press / MIT Press, Menlo Park, pp. 725–730.

Quinlan, J. R. (1996b). Learning decision tree classifiers, *ACM Computing Surveys* **28**(1): 71–72.

Quinlan, J. R. (1999). Miniboosting decision trees,
Available at `http://www.cse.unsw.edu.au/~quinlan/miniboost.ps`.

Raudys, S. J. (1988). On the accuracy of a bootstrap estimate of the classification error, *Proceedings of the 9th International Conference of Pattern Recognition*, Rome, Italy, pp. 1230–1232.

Raudys, S. J. (1997). On dimensionality, sample size, and classification error of nonparametric linear classification algorithms, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(6): 667–671.

Raudys, S. J. and Jain, A. K. (1991). Small sample size effects in statistical pattern recognition: Recommendations for practitioners, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(3): 252–264.

Safavian, S. R. and Landgrebe, D. (1991). A survey of decision tree classifier methodology, *IEEE Transaction on Systems, Man, and Cybernetics* **21**: 660–674.

Schapire, R. E., Freund, Y., Bartlett, P. and Lee, W. S. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods, *Annals of Statistics* **26**(5): 1651–1686.

Scheffer, T. (2000). Nonparametric regularization of decision trees, *Proceedings of the European Conference on Machine Learning*, Barcelona, Spain.

Schmitz, G. P. J., Aldrich, C. and Gouws, F. S. (1999). ANN-DT: An algorithm for extraction of decision trees from artificial neural networks, *IEEE Transactions on Neural Networks* **10**(6): 1392.

Sethi, I. K. (1995). Neural implementation of tree classifiers, *IEEE Transaction on Systems, Man, and Cybernetics* **25**(8): 1243–1249.

Shah, S. and Sastry, P. S. (1999). New algorithms for learning and pruning oblique decision trees, *IEEE Transactions on Systems, Man, and Cybernetics* **29**(4): 494–505.

Shawe-Taylor, J. (1997). Confidence estimates of classification accuracy of new examples, *Technical Report NC-TR-96-054*, Royal Holloway, University of London.

Shawe-Taylor, J., Bartlett, P. L., Williamson, R. C. and Anthony, M. (1996). A framework for structural risk minimisation, *Computational Learning Theory*, pp. 68–76.

Shih, Y. S. (1999). Families of splitting criteria for classification trees, *Statistics and Computing* **9**: 309–315.

Sloane, N. J. A. (2000). The on-line encyclopedia of integer sequences, Available at `http://www.research.att.com/~njas/sequences`.

Tate, A. R., Ladroue, C. and Minguillón, J. (2002). Developing classifiers for single-voxel ^{1}H brain tumour *in vivo* spectra for the INTERPRET decision support tool, *Technical Report CSRP543*, University of Sussex, School of Cognitive and Computing Sciences.

Ting, K. M. and Witten, I. H. (1997). Stacked generalization: when does it work?, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann.

Todorovski, L. and Džeroski, S. (2000). Combining multiple models with meta decision trees, *Proceedings of the fourth european conference on principles of data mining and knowledge discovery*, Springer, pp. 54–64.

Utgoff, P. E. (1989). Incremental induction of decision trees, *Machine Learning* **4**(2): 161–186.

Valentini, G. and Dietterich, T. G. (2002). Bias-variance analysis and ensembles of SVM, *in* F. Roli (ed.), *Multiple Classifier Systems*, Vol. 2364 of *Lecture Notes in Computer Science*, Springer, Cagliari, Italy, pp. 222–231.

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

Vapnik, V. N. (1998). *Statistical Learning Theory*, Wiley Series in Adaptive and Learning Systems for Signal Processing, Communications, and Control, John Wiley & Sons.

Vapnik, V. N. and Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities, *Theory of Probability and its Applications* **16**: 264–280.

Wallace, G. K. (1991). The JPEG still picture compression standard, *Communications of the ACM* **ACM-34**(4): 30–44.

Webb, G. I. (1996). Further experimental evidence against the utility of Occam's razor, *Journal of Artificial Intelligence Research* **4**: 397–417.

Winston, P. H. (1992). *Artificial Intelligence*, third edn, Addison Wesley.

Witten, I. H. and Frank, E. (2000). *Data Mining: practical machine learning tools and techniques with Java implementations*, Morgan Kaufmann Publishers.

Wolpert, D. H. (1992). Stacked generalization, *Neural Networks* **5**(2): 241–259.

Woods, K., Jr., W. P. K. and Bowyer, K. (1997). Combination of multiple classifiers using local accuracy estimates, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(4): 405–410.

Yockey, H. P. (1992). *Information Theory and Molecular Biology*, Cambridge University Press.

Index

Julià Minguillón i Alfonso
Bellaterra, July 2002