THE USE OF BOOLEAN CONCEPTS IN GENERAL CLASSIFICATION CONTEXTS

THÈSE N^o 2316 (2000)

PRÉSENTÉE AU DÉPARTEMENT DE MATHÉMATIQUES

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTOR ÈS SCIENCES

PAR

Luis Miguel MOREIRA

Ingénieur de Systèmes Informatiques, Universidade do Minho, Portugal

de nationalité portugaise

acceptée sur proposition du jury:

Prof. Alain Hertz, directeur de thèse
Prof. Robert Dalang, président
Dr. Samy Bengio, rapporteur
Prof. Giovanni Coray, rapporteur
Dr. Eddy Mayoraz, rapporteur

Lausanne, EPFL

2000

Summary

Data classification, in the present context, concerns the use of computers in order to create systems that learn how to automatically decide to which of a predefined set of classes a given object belongs.

Boolean concepts have long been present in data classification, and still are today, at various levels. We consider a kind of Boolean elements, called patterns, which consist of specific conjunctions of Boolean facts. Assuming the existence of two classes of objects, the positive and the negative, patterns can be expressed as conditionals of the type “If A and B and not C, then positive”, where A, B, and C are Boolean values, each associated with a specific attribute describing the objects to be classified. From the classification system point of view, patterns are basic Boolean expressions that can be processed in a purely mathematical manner. However, if the values of the conjunction have an intuitive meaning, then this type of representation provides an intuitive perspective of the classification system, which can be interpreted by humans.

The use of sets of patterns for building classification systems is justified by the fact that certain conjunctions of the type described above are matched by objects of one class but not by those of the other class. In this sense, patterns possess distinguishing properties that can be used to determine whether an object belongs to a class (if it matches certain patterns) or not.

In this thesis, we consider the use of classification systems based on patterns, which we have seen to be inherently Boolean, in the context of classification tasks where nothing is Boolean, that is to say, where the objects are described by multi-valued or continuous attributes and instead of a positive/negative decision, suitable to situations where only two classes are involved, the output is rather the selection of a class among several.

The work presented here extends along different directions. It starts by the study of a suitable way of transforming the attributes describing the objects to be classified into equivalent Boolean attributes. We describe the constraints that must be satisfied by this transformation and present a procedure allowing it to be done in a short amount of time.

Another research direction explores the possibility of generating multi-class decisions with systems whose output is Boolean. This study has the benefit of also being applicable to other types of classification systems adapted to two-class situations, but whose internal language is not necessarily Boolean. This is the case of some current state-of-the-art systems. To realize this adaptation, the common approach consists in decomposing the original problem into several

two-class sub-problems and generating an independent classification system to solve each one of them. The final class decision is obtained from the combination of the partial Boolean decisions. We analyze and compare several existing procedures for this purpose, and in addition propose an interesting new procedure.

Finally, we consider the possibility of sharing patterns among several classes and develop a Boolean-based classification system which is directly adapted to multi-class problems, but maintaining the property of being understandable by humans. We show that the performance of this type of system is not in all cases comparable to the best available systems, but they have the advantage of providing, in certain circumstances, a better understanding of the problem at hand.

Version Abrégée

La classification de données, telle qu'elle est considérée ici, est un domaine qui traite des systèmes informatiques capables d'apprendre à décider par eux-mêmes à quelle classe, parmi un ensemble prédéfini, appartient un objet donné.

On retrouve depuis longtemps des concepts booléens dans des systèmes de classification de données, à plusieurs niveaux. Cette thèse aborde un type d'éléments booléens, désignés «patterns», qui sont des conjonctions de faits booléens. Si on considère la présence de deux classes d'objets, les positifs et les négatifs, un pattern peut être décrit comme une condition du type «Si A et B et non C, alors positif», où A, B et C sont des valeurs booléennes, chacune associé à un des attributs décrivant les objets à classer. Bien que les patterns soient des expressions booléennes traitées par le système de classification de façon strictement mathématique, dans le cas où les valeurs dans la conjonction ont un sens intuitif ce type de représentation permet une perception intelligible du système par un humain.

L'utilisation d'ensembles de patterns dans des systèmes de classification se justifie par le fait que certaines conjonctions du type décrit ci-dessus s'appliquent à certains objets d'une classe mais à aucun des objets de l'autre classe. Dans ce sens, les patterns possèdent des capacités discriminatoires qui peuvent être utilisées pour déterminer si un objet donné appartient à une classe (s'il est accepté par certains patterns) ou pas.

Cette thèse aborde l'utilisation de systèmes de classification de caractère booléen, basés sur des patterns, dans des contextes de classification plus généraux où rien n'est booléen. C'est le cas où les objets sont décrits par des attributs à plusieurs valeurs ou même des attributs continus, et au lieu d'une décision du type positif/négatif, adapté au cas à deux classes, la réponse désirée est plutôt une classe parmi plusieurs.

Le travail est développé selon trois directions. D'abord, une façon appropriée de transformer les attributs qui décrivent les objets à classer en attributs équivalents au format booléen est étudiée. Après en avoir décrit les contraintes, une méthode performante de transformation est élaborée.

Une autre direction de recherche explore la possibilité de générer des décisions du type multi-classes à partir de systèmes dont la sortie est booléenne. Ceci a l'avantage de pouvoir être utilisé par d'autres types de systèmes de classification adaptés aux cas à deux classes, mais dont le langage de représentation interne n'est pas forcément booléen. C'est le cas de quelques uns des systèmes actuels les plus performants. Afin de permettre cette adaptation, l'approche usuelle consiste à décomposer le problème initial en plusieurs sous-problèmes à deux classes. Ensuite, chacun de ces sous-problèmes est résolu par son propre système de classification. La décision finale est obtenue par une combinaison des décisions partielles. Dans ce travail, plusieurs méthodes existantes sont analysées et comparées entre elles, et une nouvelle méthode offrant des propriétés intéressantes est élaborée.

Finalement, la possibilité de partager des patterns entre plusieurs classes est abordée, et un système de classification booléen à la base mais capable de traiter directement des problèmes du type multi-classes est développée. Ce système conserve la propriété d'être compréhensible par un humain. Bien que ce type de système ne soit pas parmi les plus performants dans tous les cas, il présente, sous certaines conditions, l'avantage d'offrir une meilleure compréhension du problème traité.

Remerciements

Une thèse de doctorat est rarement le fruit de l'effort d'une seule personne. Je tiens à remercier quelques uns de ceux qui ont apporté leur participation à la bonne réussite de celle-ci.

Je remercie d'abord Eddy Mayoraz, qui a jeté les bases de ce projet et qui lui a donné son impulsion fondamentale; Alain Hertz pour son orientation, ses idées et son enthousiasme; Samy Bengio pour sa rigueur et sa disponibilité. Je remercie aussi tous les membres du jury de thèse pour avoir accepté de donner leur avis critique, ce qui a permis de valider la qualité de ce travail.

Tout au long de ce projet certains de mes collègues m'ont parfois rendu le parcours moins ardu. Je tiens donc à remercier spécialement Frédéric Gobry, Johnny Mariéthoz, Perry Moerland et Gianni Pante, mais aussi bien d'autres à l'Idiap.

Un mot spécial pour Astrid qui pendant toute cette période a donné de soi-même certainement beaucoup plus que ce que j'ai aperçu. Merci aussi à ma famille, à Vitor et à Rui qui, de l'autre coté, ont toujours souhaité le bon aboutissement de cette aventure.

Enfin, ma reconnaissance au Fonds National Suisse de la Recherche Scientifique qui a fourni le soutient financier à ce projet de recherche, avec les subsides 21-46974.96 et 2000-053902.98/1.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms and Notation

AI	Artificial Intelligence
CC	Correcting Classifiers
ECOC	Error-Correcting Output Codes
GA	Genetic Algorithms
IDEAL	Iterative Discriminant Elimination Algorithm
IDIAP	Institut Dalle Molle d'Intelligence Artificelle Perceptive
KDD	Knowledge Discovery in Databases
LAD	Logical Analysis of Data
MC-LAD	Multi-Class LAD
ML	Machine Learning
OPC	One-Per-Class
PD	Pertinent Dichotomies
PWC	Pairwise-Coupling
PWC-CC	Pairwise-Coupling with Correcting Classifiers
PWC-DAG	Pairwise-Coupling with Directed Acyclic Graphs
SA	Simulated Annealing
SCP	Set Covering Problem
SG	Simple-Greedy
Std-LAD	Standard LAD
TS	Tabu Search

Σ	output space
σ	sigmoid activation function
Ω	input space
ω	coverage of a term
$\underline{\omega}$	coverage rate of a term
A	number of attributes
a	attribute
B	number of binary attributes
b	binary attribute
$\mathcal{B}$	set of terms
c	class

$\mathcal{C}$	class of functions
$\boldsymbol{D}$	decomposition matrix
D	number of discriminants
d	discriminant
d_{Hamm}	Hamming distance
$\mathcal{D}$	set of discriminants
e	class entropy
F	target (unknown) classification function
$\hat{F}$	classification model
f	ideal (unknown) function of a dichotomy
$\hat{f}$	dichotomy
$\mathcal{F}$	set of false (negative) Boolean data
G	degree of a term
$\hat{g}$	correcting dichotomy
$\mathcal{G}$	matrix (set) of used pairs of classes (germs)
H	neighborhood
h	partially defined Boolean function
K	number of classes
$\mathcal{K}$	cluster
$\mathcal{L}$	set of clusters
$\boldsymbol{M}$	reconstruction matrix
M	consistency level
m	Boolean mapping
m	move (in the framework of Tabu search)
N	number of instances
O	objective function
P	number of patterns
p	pattern
$\mathcal{P}$	set of patterns
Q	number of dichotomies
R	matrix of differentiability rates
r	differentiability rate
$\mathcal{R}$	set of segments
S	set of solutions
s	solution
$\mathcal{S}$	segment
T	tabu list
t	term
$\mathcal{T}$	set of true (positive) Boolean data
u	class frequency
w	weight
$\boldsymbol{x}$	instance
$\mathcal{X}$	set of data
$\mathcal{Y}$	set of Boolean data
z	linear mapping function

CHAPTER 1

Introduction

1.1 Principles of automated learning

Automated learning is related to computer systems that use a set of training material in order to develop some kind of internal representation allowing them to produce conclusions that extend beyond the learning source. What can computer systems learn? For example, they can be trained to predict whether a patient suffers from a certain disease by having observed previous patients, or to decide whether a credit applicant is trustworthy based on previous loan records. They can also be trained to recognize handwritten or printed characters, to detect the existence of faces or objects in images, and even to recognize a person from the image of its face. Under certain conditions, some systems can even learn how to understand human speech. These are illustrations of domains where computer systems are able to learn from examples, and to generalize the acquired capabilities to new cases that have not been used during the training phase. Automated learning systems are gaining ever more acceptability and applicability, and besides their capability for giving approximately correct answers to the treated problems, they can be useful tools for a better human understanding of the problem itself. For example, if medical staff may be reluctant to trust a learned system for emitting important diagnostics, they can take great advantage of knowing the corresponding decision process. That is, if the system is able to explain its decisions, it can give the doctor new valuable insights into his/her own field. All these techniques are usually treated in the domain of *Machine Learning*.

With the advent of information technologies and their ever growing ability for handling and storing large amounts of data, comes the need for tools that can extract meaningful information from possibly largely irrelevant or noisy data, and allow useful conclusions to be derived from the most representative of them. Machine Learning tools have an important role to play also in this process. The field of Knowledge Discovery in Databases (KDD) (Fayyad et al., 1996) is involved in the large-breadth process of gathering, preparing, and processing data for knowledge extraction which integrates Machine Learning techniques at its core.

1.1.1 Supervised and unsupervised learning

Automated learning approaches can be divided in at least two types: *supervised* and *unsupervised*. In the former, the system receives training cases, each being represented by a set of input values together with the correct response for it. The goal is to acquire the ability to correctly respond to new cases. *Classification* is a supervised learning technique where the output to predict is contained in one of a set of predefined *classes*. This is the case for all the above-mentioned examples of automated learning systems. The set of classes can be a set of words or phonemes (in automatic speech recognition), a set of characters and digits (in character recognition), but it can also contain only two classes, like "healthy/sick patient", or "qualified/unqualified credit applicant". This dissertation is concerned with classification problems. Another kind of supervised learning is *regression*, also known as *curve-fitting*, where the output is real-valued.

In unsupervised learning, there are no outputs provided to the learner, only the input values, so it must try to discover meaningful information by itself. Examples of unsupervised learning techniques include clustering, modeling the probability distribution of the data, and finding other kinds of structures in them (Jain and Dubes, 1988; Gersho and Gray, 1992; Kohonen, 1995; Cheeseman and Stutz, 1996). Clustering techniques, for example, try to find a natural distribution of the data into clusters, without any prior class membership information, which can be seen as attempting to determine the natural classes of the data. The name "unsupervised" does not imply that the learning process carried out with these techniques is totally autonomous. In reality, it is guided by some measure of optimality, as is the case with supervised techniques.

1.1.2 The approach to learning a classification task

The data

We have seen that the output in classification tasks is a discrete set of classes. As for the input, it consists of a vector of values that describe the cases which are to be classified. These values are called *attributes*, or *features*, and they should allow the learner to create distinguishing structures between cases belonging to different classes. For example, we can reasonably accept that the natural eye- or hair-color of a patient does not help in diagnosing him or her as healthy or as suffering from some sort of heart disease. These two features would certainly not help deciding if someone is qualified for obtaining a credit either. In contrast, the age and the blood pressure for the heart disease problem, or the annual income for the credit assignment problem, are likely to be considered important distinguishing attributes.

In any practical classification task we suppose that there is an underlying, unknown, stochastic mapping between the input and the output that is represented by the training data. Suppose that this mapping could be learned exactly, if the learner had access to the exhaustive set of input data, that is, if it could learn the correct response for every possible input. It can be easily deduced that no learning system would be useful in such situation, because the answers for every possible classification query would have been known. In addition, no practical situation offers the possibility of accessing the whole set of input data, which would correspond to considering every possible combination of the input attribute values. This would be overwhelming even for an input space of reasonably small dimension and, more than that, impossible in case any of the attributes is real-valued.

So far we have been referring to "cases" to denominate the data units that are used for training automated learning systems, and also those that must be classified posteriorly. Throughout this dissertation these entities will be called *instances* or *examples*, and they correspond to patients, credit applicants, images, or split-seconds of speech audio signals, to match the examples mentioned before. In the current framework, we assume that for a given classification task all the instances of a set of data are described by the same set of attributes, which allows them to be stored in a data table where each row contains one instance and each column corresponds to one attribute. In addition, the examples that are used for learning contain an additional column with the *target* attribute, which is the output value of the instance. This special attribute is also called the *class label*.

The learner

To build a classification system, we employ a *learning algorithm*, or *learner*, whose goal is to create a model of the relation between the input and the output of the given classification task. This hidden relation is called the *target concept* (Mitchell, 1997), the function to be learned. The model created by the learning algorithm is called the *classification model*. Moreover, we consider that a *classifier* is a system that employs the generated model to provide classification decisions. For building the classification model, the learning algorithm is given a finite set of data representative of the mapping to be learned. The finalized classifier is expected to correctly classify cases that have not been learned, but that are also representative of the target concept. The *generalization ability* of a classifier is measured by the extent to which it can generalize what it has learned to new situations, i.e. its ability to correctly classify examples of the same population as the training data but which have not been presented before.

The generalization ability is an important measure of the quality of a classifier. In nearly all practical situations, it must be measured using a finite amount of data before the system is put into practice. The general approach consists in splitting the data that are available for creating the classifier into two different sets: the *training set* and the *testing set*, the former being used to train the classifier (the actual learning phase) and the latter to test its performance. The training set is provided to the learning algorithm with the class labels included, while the testing set is submitted to the classifier without the labels. The *accuracy* of the classifier is given in terms of the proportion of testing examples that it correctly classifies. An equivalent measure, from a different point of view, is the *error rate*, the proportion of misclassified testing examples. As will be discussed later in Sect. 1.3, the training set can in some cases still be partitioned into two subsets, the actual training set and a *validation set*.

Practical difficulties

Usually, there are certain difficulties involved in learning a classification task. Depending on the particular problem treated, and on the quantity and quality of data available, it is seldom the case that the accuracy of a classifier approaches 100%, in fact it is common to have it considerably below this level if the learning conditions are especially hard. There are several factors that account for this: the amount of training data is finite and sometimes scarce; the choice of the input features may be sub-optimal; the modeling capabilities of the learning algorithms themselves have limitations; or the data may be corrupted. Data corruption can happen at least in two different forms: (i) they can contain *noise*, which means that certain class labels are incorrectly assigned or that some attribute values have been incorrectly measured; and (ii) some of the instances may have *missing attribute values*. These are normal situations that must be faced by any learning algorithm.

1.2 Formalism of classification

We now establish the core of the notation for classification tasks that will be used throughout this dissertation.

The learning algorithm learns an estimate $\hat{F}$ of the unknown target concept F, which is defined from an *input space* $\Omega \equiv \mathbb{R}^A$, where A is the number of attributes, into an *output space* $\Sigma \equiv \{c_1, ..., c_K\}$ of discrete classes, which may be ordered or not, with K being the number of classes. Note that the input Ω is defined in the real valued space, which means that the input features can take any form as long as their values can be represented numerically[1]. The learning phase is carried out using a set $\mathcal{X}$ of training data of the form $\langle \boldsymbol{x}_n, F(\boldsymbol{x}_n)\rangle, n \in \{1, \ldots, N\}$, where $\boldsymbol{x}_n$ is an instance, $F(\boldsymbol{x}_n)$ is its class label, and N is the number of instances in $\mathcal{X}$.

We can say that the function $F : \Omega \to \{c_1, ..., c_K\}$ defines a K-partition of the input space into subsets $F^{-1}(c_k)$, each containing the data of class c_k. Hence, the training data set $\mathcal{X}$ is also partitioned into class subsets ($\mathcal{X} \equiv \bigcup_{k=1}^{K} \mathcal{X}_k$). The function $\hat{F} : \Omega \to \{c_1, ..., c_K\}$ is a *decision rule* and it assumes different forms internally, depending on the particular type of classification model.

The set $\mathcal{X}$ of training data contains a subset of the whole feature space ($\mathcal{X} \subseteq \Omega$, not considering the class labels), and the created model $\hat{F}$ should generalize to instances of the feature space not contained in $\mathcal{X}$. The accuracy of a classifier, that is, the quality of the obtained approximation $\hat{F}$, can be measured empirically by using a testing set $\mathcal{X}_{test}$ of examples agreeing with F, but being mutually exclusive with $\mathcal{X}$ ($\mathcal{X}_{test} \cap \mathcal{X} \equiv \varnothing$).

1.3 Basic elements of learning theory

The approximation $\hat{F}$ of the target concept F can be implemented by a large variety of different models. The next section briefly presents some of the major types of learning algorithms and the models they create. Every type of classification model is implemented by a different representational language, which can range from very simple to highly complex. This language determines how the learned information is stored and used for inducing new classification decisions.

Certain types of representational languages are very powerful, capable of modeling highly complex information structures, while others are more limited. There is a relation between the representational power of classification models and the ability of their respective algorithms to

[1] A detailed discussion of basic attribute types is presented in Sect. 3.1.1 on page 34.

correctly learn classification tasks.

We can consider that each learning algorithm has access to a space of functions, also called the space of *hypotheses*. This space is defined by the representational language of the particular model type considered. In other words, each type of model implements functions of a certain class or family. This can be the class of all neural networks, the class of all decisions trees, and so on (cf. next section). The goal of the learning algorithm is to find, in the space of hypotheses (class of functions), the most appropriate one for a particular learning task.

According to the discussion of the previous sections, the choice of the best hypothesis relies on a finite set of training data. There are two main implications of this fact: (i) if the hypothesis space is too rich, it will enable the algorithm to choose a hypothesis that fits the training data too perfectly, which may potentially cause the resulting model to poorly generalize; and (ii) the more training examples are available, the higher the probability that the chosen hypothesis will correctly approximate the actual target concept F.

The representational power of a class $\mathcal{C}$ of functions is determined by the *capacity* of $\mathcal{C}$. Vapnik (1995) has developed important groundwork on statistical learning theory, in which he formally defines the capacity of a class $\mathcal{C}$ of functions as the maximal number N of training examples that can be correctly classified by a function in $\mathcal{C}$, independently of the class labels assigned to the N examples. More informally, we apply the term "capacity" also to a specific function or learning algorithm, to denote the class of functions they represent.

Any successful learning algorithm must have some control on the capacity, for the sake of generalization ability. One possibility for limiting the capacity of a model consists in considering a class of functions with limited representational power, which prevents the model to overly adapt to the training data. Another, usually better, alternative is to integrate mechanisms in the learning algorithm allowing it to avoid too powerful hypotheses. This can be done by using either validation techniques or statistical measures of confidence. The former consists in setting part of the training data aside in order to measure the generalization of different hypotheses during the execution of the algorithm. These data are usually called the *validation set*. On the other hand, statistical techniques can be used to calculate estimates of the performance of each hypothesis on future examples, based only on the behavior of the training data with relation to the model. The reason why these two kinds of approaches are better than using limited-capacity hypotheses is that they can adapt to the characteristics of each particular learning problem.

Figure 1.1 on the next page, which has been borrowed from Bishop (1995, Sect. 1.5), shows a simple example of a classification task where the training data are represented in a two-dimensional space (by two attributes, a_1 and a_2) and divided into two classes. Each plot shows a different function implementing the decision boundary. It can be observed that the function of (a) has low capacity. It produces several training errors, and it will probably produce several errors also on unseen data. In (b), the function seems to have the required level of capacity. Despite a few errors on the training data, it is likely to generalize correctly. Finally, the function plotted in (c) has high capacity, and it can model the training data very accurately. However, the results on unseen data are likely to be disappointing. In this case, we say that the model *overfits* the data. Note that these conclusions rely on the supposition that the training data may be noisy, which explains the statement that the function of (b) is the most appropriate one for the task.

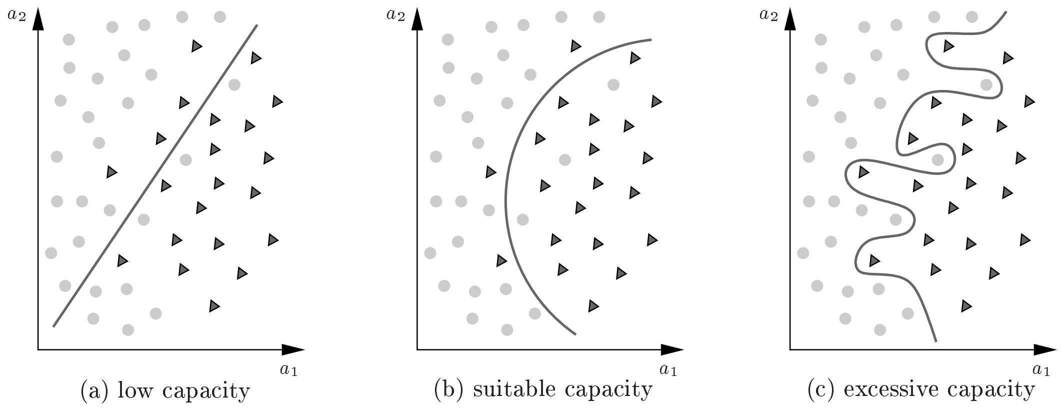


Figure 1.1: A decision boundary implemented by three functions, each with a different level of capacity, for the same classification task.

Hence, due to the usual limitation on the amount of training data, there is a tradeoff between the error obtained on the training set, and the expected error on unseen data. Figure 1.2 gives

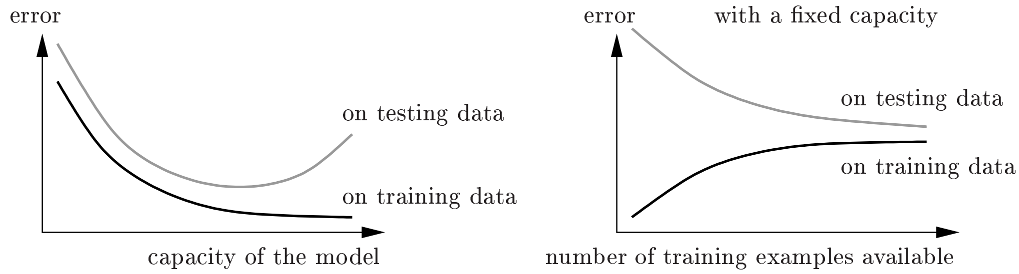


Figure 1.2: Combined effect of the capacity of the classification model and of the number of available examples, and the resulting error.

an idea of two important relations that are usually observed in automated learning contexts, and that are supported by the theory of statistical learning (Vapnik, 1995). The plot on the left represents the phenomenon observed before on Fig. 1.1, i.e. raising the capacity can favor the generalization error up to a certain level, after which it degrades, in spite of the continuing improvement on the training data. The plot on the right of Fig. 1.2 shows the relation between the classification error obtained on the training and on the testing data, for a fixed capacity level. In practice, the training error is irrelevant in classification contexts, it is the generalization error (calculated on the testing data) that provides the quality measure of a classifier. The fewer training examples are available, the easier it is to generate a model that classifies them correctly. However, this causes a high testing error. On the other hand, using many training examples allows the approximation $\hat{F}$ of the target concept F to be better, perhaps at the expense of a higher training error, which is irrelevant.

1.4 LAD and alternative learning algorithms

This section describes some of the major types of learning algorithms that are applied to classification tasks. It begins by presenting a method which has served as the main motivation for all the work presented in this dissertation.

1.4.1 What is LAD

Logical Analysis of Data (LAD) (Boros et al., 2000) is a methodology for automated learning that builds classification models based on *patterns*. Patterns are basically conjunctions of certain attribute values, and they are used in LAD to describe target concepts involving two classes only. The use of patterns assumes that it is possible to find combinations of attribute values which are typical of examples from one of the classes and do not occur in the other. The LAD classification model is built by a process that explores a search space in order to find appropriate sets of patterns explaining the generality of training examples and distinguishing the data of the two classes. Like for any other learning method, this is done during a training phase in order to construct a model that should generalize its conclusions to previously unseen examples of the same domain.

The method is inherently logical, as it has been developed in the framework of Boolean function theory. Its logical nature makes it especially adapted to learn concepts involving two classes and where the data are represented by Boolean features. LAD employs techniques that allow the transfer of data of numerical or symbolic format into Boolean features.

Besides providing classification decisions, the patterns of LAD are a powerful and interpretable mechanism for allowing human understanding of the problem domain. A deeper exposition of the method is made in Sect. 2.3.1 on page 17 and following.

1.4.2 Other approaches for classification

There are several kinds of approaches for solving classification tasks. In this section, we briefly describe the most important ones. Certain approaches have a close resemblance to LAD and are described in more detail in Sect. 2.4. Each type of classification model has its own internal knowledge representation, with varying levels of complexity and interpretability. Some of them use a very intuitive representation, while others consist of a set of numerical parameters with a certain mathematical meaning but completely opaque to human understanding.

We distinguish two groups of methods: one originating in the domain of Artificial Intelligence, composed of methods that are essentially symbolic, i.e. that consider input attributes taking symbolic values, and build models based on relations between those possible values. This is the case of rule induction methods (similar to LAD), decision trees, and other more complex logic-based methods using first-order Horn clauses. Throughout the years, most of the algorithms included in this category have evolved in order to be able to appropriately handle attributes of numerical type. The other group of methods includes those that consider the input space as inherently numerical. They originated mostly in the field of Statistical Pattern Recognition (Duda and Hart, 1973; Bishop, 1995; Ripley, 1996). Their representational language is fundamentally mathematical but has usually an equivalent geometrical interpretation, such as separating hyperplanes or polynomials, and regions defined in the input space of features, or some derived hyperspace.

For each category described below, we provide a set of references containing historical, basic, and comprehensive material.

Classification rules

Rule induction methods are based on logic, like LAD, although the precise representational language varies according to the specific model. In general, the model consists of a set of rules of the type *IF condition THEN class label*, where the condition involves a particular combination of attribute values. New examples are classified by assigning them the class of the matching rule or, when they are matched by several rules simultaneously, by basing the decision on an ordering or a weighted combination of the rules. Most of the systems of this type use conditions which are conjunctions of attribute values (like the patterns in LAD), but some of them use alternative representations, like disjunctions or ranges of values. Different strategies can be used to construct the set of rules. Rule induction learning algorithms are discussed in more detail in Sect. 2.4, due to their proximity to LAD. (Michalski, 1969; Plotkin, 1971; Vere, 1975; Mitchell, 1978; Michalski et al., 1986; Rivest, 1987; Clark and Niblett, 1989).

A special type of rules that can be derived from standard propositional rules are first-order Horn clauses, which contain variables in the condition part, allowing to treat data with more elaborate representations, not circumscribed to fixed-length vectors of attribute values. Models based on first-order rules are constructed with PROLOG programs. (Quinlan, 1990; Lavrač and Džeroski, 1994).

Decision trees

Decision trees are another symbolic approach, whose models have a tree structure. Each node of the tree contains a test on the value of an attribute, where different outcomes of the test lead to different sub-nodes. The nodes in the lowest level of the tree are the *leaf* nodes and contain a class label. The classification model is built recursively from the first node (the *root*) to the leaves. For each added node, the input space is partitioned into sub-spaces, which are in turn also partitioned by their respective sub-nodes. Hence, each leaf node is associated with a certain portion of the input space. New instances are classified by following a path from the root node through the tree to one of the leaves. The path is decided by the tests done at each node according to the value of the instance for the corresponding attribute. Decision tree models are also described in more detail in Sect. 2.4.3. (Hunt et al., 1966; Quinlan, 1979, 1983; Breiman et al., 1984; Kononenko et al., 1984; Cestnik et al., 1987).

Instance-based learners

Also known as *k-nearest-neighbors classifiers*, these models are composed of the set of training data itself. New examples are classified by assigning them the majority class of the k nearest training examples in the input space. This requires a distance measure, which can be a simple

Euclidean distance or other more complex functions. Classifiers of this type are sometimes referred to as *lazy-learners*, due to the absence of an actual learning phase. However, the model representation lacks conciseness, as it is based on the whole set of training data. (Cover and Hart, 1967; Duda and Hart, 1973; Aha et al., 1991).

Discriminant functions and Artificial Neural Networks

Linear discriminant functions are simple models that can be applied when the data of the different classes are linearly separable in the input space. The main idea is to define a function which is a linear combination of the inputs and which discriminates the data of two classes. The linear discriminant function can be interpreted as a weighted sum of the input signals (the values of the attributes) where the set of weights defines the normal vector of an hyperplane separating the data in the input space. In addition, a nonlinearity (usually sigmoidal) can be applied to the result of the sum in order to force the output to be either 0 or 1. The model is usually trained based on an error measure that is a function of the weights. Techniques such as gradient descent of the error function are used to progressively adjust the weights towards an optimal configuration, which is equivalent to finding the best orientation of the separating hyperplane. In the presence of problems of small size, direct optimization techniques, that use matrix inversion operations, can also be applied.

Artificial neural networks are a generalization of the above type of model, where several discriminating units are connected in order to create more complex (nonlinear) discriminant functions. They are inspired by biological neuronal systems where each *neuron* is a unit similar to the simple discriminant function described above, coupled with the respective nonlinear transformation. Neurons are organized in layers in a neural network and each neuron of a layer is connected with every neuron of the following layer. The most common network topologies contain three layers of neurons: a layer of simple inputs, connected to a layer of hidden nonlinear units which is in turn connected to a layer of output units. The signals are propagated from the input units to the output layer, which contains a neuron for each class.

The hidden layer of neurons creates internal representations able to define discriminant functions of high complexity. The number of hidden units determines the learning capacity of the model. Following the discussion about capacity held earlier, using few neurons renders the task hard to learn, while using many of them creates the risk of over-learning the training data and generalizing poorly.

The most common training method for artificial neural networks is based on error back-propagation, that is used to adapt the weights of the connections through gradient descent of the error function. (Rosenblatt, 1962; Duda and Hart, 1973; Minsky and Papert, 1988; Rumelhart et al., 1986; Hertz et al., 1991; Haykin, 1994; Bishop, 1995).

Bayesian classifiers

Bayesian learning assumes that the data are distributed in the input space according to some probability distribution, and that it can be approximated by a parametric statistical model, using the training data in order to determine the respective parameters. The most commonly used models are based on Gaussian distributions.

Bayesian classifiers are directly based on Bayes Rule, which provides a framework for determining the most probable class given the data, and which minimizes the error of misclassification. It allows classification decisions to be made according to the class c_k with the highest posterior probability $\mathrm{P}(c_k|\boldsymbol{x})$, where $\boldsymbol{x}$ is the example to be classified[2]:

$$\mathrm{P}(c_k|\boldsymbol{x}) = \frac{\mathrm{p}(\boldsymbol{x}|c_k)\mathrm{P}(c_k)}{\mathrm{p}(\boldsymbol{x})} \ . \tag{1.1}$$

The function $\mathrm{p}(\boldsymbol{x}|c_k)$ is the class conditional probability density (also called the *likelihood* of c_k), and it describes the distribution of the input feature values for observations of class c_k. This is the major distribution that must be calculated during the training of the model. The *prior probability* $\mathrm{P}(c_k)$ is the probability of occurrence of class c_k. Finally, $\mathrm{p}(\boldsymbol{x})$ is the *unconditional probability* density function, representing the distribution of the data in the input space independently of the class; it has a normalization role in Eq. (1.1), to guarantee that the sum of the posterior probabilities for all classes equals 1. (Duda and Hart, 1973).

Support Vector Machines

Support Vector Machines (SVM) have recently received considerable attention from the Machine Learning and Statistics communities, because of their strong theoretical background, together with proven good empirical performances. These models are derived from concepts of statistics and computational learning theory. The main principle consists in finding optimal separating hyperplanes that discriminate the data of two classes. However, this is not done in the original input space, but rather in a high-dimensional space to which the original data are transfered in order to gain linear separability. This space transformation is done through the use of *kernels*, which are mapping functions whose properties may allow the data to become linearly separable, under certain conditions, when mapped to a high-dimensional space.

The learning algorithm consists basically of an optimization procedure that finds the optimal separating hyperplane, which is the one that maximizes the *margin*, the minimal distance between any training example and the hyperplane. The maximization of the margin is assumed to provide the best discriminating properties, and consequently the best generalization ability. The name of the method originates from the type of classification model that it generates, which consists simply of a set of key training examples, the *support vectors*, and associated coefficients. These are the examples with minimal margin which, combined linearly, define the separating hyperplane in the high-dimensional space. (Vapnik, 1995; Burges, 1998; Schölkopf et al., 1999).

1.4.3 Discussion

Many studies have shown that there is no such thing as the best classification algorithm, which depends on the treated problem and on the quality criterion considered (e.g. classification accuracy, complexity of the model, interpretability, training time). Some algorithms are better than others in some tasks but are less efficient in other tasks. Nevertheless, there are some

[2] In Eq. (1.1), $\mathrm{P}(\cdot)$ represents a probability value, while $\mathrm{p}(\cdot)$ represents a probability density function.

prominent methods, at least with regard to classification accuracy, like those based on Decision Trees (one such algorithm is used in several experiments in this dissertation), Artificial Neural Networks and, more recently, Support Vector Machines, which are all known to be very accurate in a large variety of tasks.

Among all the types of algorithms that have been mentioned in this section, only LAD, Classification Rules, and, in a sense, Decision Trees and Instance-based learners, generate models that are human-readable and that can be used for a purpose other than high classification accuracy. Concerning the standing of LAD in particular, its classification performance has been compared favorably with other methods on some well-known classification tasks (Boros et al., 1996) and it has shown interesting results also in some pilot studies (in the same publication). More conclusive results are presented later in Chap. 5.

1.5 The dissertation

1.5.1 Goals

The initial proposal for this doctoral project has been largely motivated by the LAD approach, which has a marked presence throughout this dissertation. At the beginning of the project, the overall goal consisted in generalizing the applicability of the method to a wider range of problems than those that were feasible at that time. The main identified obstacles were the following:

- the method was applicable exclusively to two-class classification problems;
- the main inductive engine had exponential time complexity on the number of attributes, despite the use of certain heuristics aimed at reducing the actual execution time;
- the existing procedure for finding a mapping between the input data of arbitrary numeric format into Boolean format had quadratic time and storage space complexity on the number of training examples, which made it unmanageable with medium to large-sized classification problems.

Hence, our work started with the attempt to develop solutions to each one of these problems. This document shows that these goals have been attained. In addition, the evolution of the research activity led to certain additional developments, not specific to LAD, which are also reported here. In order to achieve the proposed goals, we have been faced with several different types of problems. Hence, the subject of this thesis lies somewhere at the confluence of various subjects, of which the most important are: automated learning, knowledge discovery, heuristics for optimization, and Boolean logic.

The core of LAD is inherently Boolean: it receives Boolean inputs, provides a Boolean output, and the internal reasoning is based on Boolean concepts. Ideally, this core must be enveloped by an external layer allowing the method to interface with more general conditions, those found in real-life situations. Figure 1.3 on the following page describes this principle graphically. It shows the scheme of a classifier receiving raw data for classification and gener-

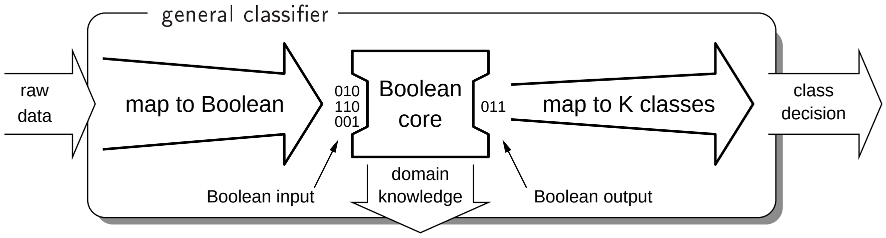


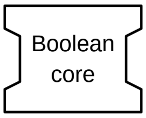
Figure 1.3: General scheme of a classification system composed of a Boolean inductive engine.

ating class decisions. As the inductive engine is Boolean, the classifier must contain internal mechanisms allowing the input data to be transformed before they are fed to the core, and allowing a class decision to be made even when that decision is not Boolean, i.e. when it involves more than two classes. The scheme also shows that the type of classifier considered here has the possibility of generating another type of information, which consists in human-interpretable conclusions about the treated domain, denoted "domain knowledge" in the figure. Using this scheme, we briefly describe below the layout of the dissertation and how the developed work will be presented.

1.5.2 Guide to the dissertation

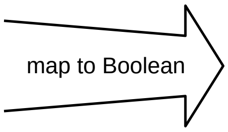
This document contains seven chapters, including the current one, plus two appendices. Chapters 1 and 2 contain introductory material, while Chapters 3 to 6 present actual research developments, and finally Chap. 7 draws the main conclusions.

Logical Analysis of Data



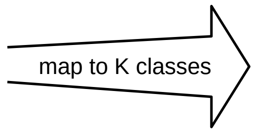
Chapter 2 describes the main principles involved in the use of Boolean concepts for classification tasks. This is done through a description of the LAD approach, but it also refers to alternative, similar logic-based methods proposed in the literature.

Boolean mapping



Chapter 3 treats the problem of transforming the input data, which is presented to the classifier in general numeric format, into Boolean format. As the size of the Boolean representation is normally much larger than the original size of the data, we discuss a way of reducing as much as possible the Boolean images created by the mapping, while obeying certain restrictions.

Decomposition of classification tasks



Chapter 4 discusses several methods for allowing classifiers which generate binary decisions (suitable to two-class problems) to be applied to multi-class classification tasks. The studied methods work by decomposing the original task into simpler binary sub-tasks, and then recombining the solutions to the sub-tasks in order to create the final class decision. This discussion applies to Boolean-based classifiers as well as to any other type of classifier generating two-class decisions.

LAD in multi-class contexts

Chapters 5 and 6 are closely related. They extend some of the ideas presented in Chap. 4 in order to create a version of the LAD method that is able to generate multi-class decisions directly from the Boolean core. This is highlighted in Fig. 1.4, where the scheme presented in Fig. 1.3 on the preceding page is slightly modified in order to represent the new way of considering the core of the method. Chapter 5 describes the implementation of this new type

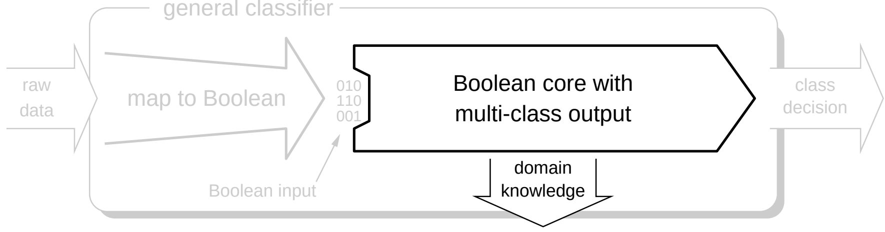


Figure 1.4: The scheme of Fig. 1.3 with a modified inductive engine directly treating multi-class outputs.

of core and discusses the modifications that it introduces with respect to the interpretability of the generated domain knowledge. Chapter 6 describes in detail one of the basic components of the new inductive engine, more precisely the pattern generation process.

Experimental setup

A series of experiments is described throughout the dissertation, providing empirical support to the formulated arguments. These experiments were made using a group of data sets derived from real-life applications, which are freely available to the scientific community. These data sets are described in Appendix A. Also related to the empirical tests, Appendix B provides details about the way in which the experiments were made, as well as about the statistical tests used to measure the significance of the obtained results.

1.5.3 Main contributions

Here, we summarize the three main contributions provided by this work of research.

Boolean mapping

We propose a new heuristic-based procedure for creating a mapping that allows the input data to be transformed into Boolean format. Its worst-case complexity is quadratic on the number of examples, but the actual execution time of the procedure has been shown to be very short in all cases. In practice, for the larger tested data sets, the problem has been solved in a few minutes. This is treated in Chap. 3.

Study of decomposition and reconstruction procedures

We make a comprehensive comparison between several methods allowing two-class classifiers to be applied to multi-class problems. As far as we know, no such comparison has ever been made before, despite the great interest of this type of approach. The major observation that this comparison provides is the fact that there is usually a tradeoff between the attained level of generalization and the complexity of the procedure. For large problems, it may be infeasible to apply the methods providing better classification accuracy, due to the computational cost involved. In addition to the mentioned comparison, we also propose a new method which provides a good compromise between the obtained accuracy and the required complexity. This is treated in Chap. 4.

Extension of LAD to multi-class problems

The goal of generalizing the LAD approach to the multi-class case and reducing its complexity, which allows it to be applied to problems of larger size, has effectively been attained through a newly proposed method. Its classification performance is not better than alternative state-of-the-art methods, but we show the benefit of the Boolean approach in terms of the interpretability of the model, also in the new multi-class case. This is treated in Chapters 5 and 6.

Chapter 2

Logical Analysis of Data

Motivation

This chapter describes Logical Analysis of Data as a method for solving classification tasks and providing insights of the problem domain. It provides the historical context (Sect. 2.1), the basic notions and concepts (Sect. 2.2), and the state of the art of the methodology at the moment when the work presented in the current dissertation was initiated (Sect. 2.3). The goal is not only to describe the method, but also to provide the motivation for the work that has been developed. Section 2.4 contains a basic description of comparable classification approaches proposed in the literature.

2.1 Brief historical background

The roots of LAD can be found in the 1980's, when the team of Prof. Hammer at the Rutgers University Center for Operations Research (RUTCOR) in New Jersey encountered the possibility of employing special kinds of Boolean functions for the extraction of knowledge from data. The first public presentation of those seminal ideas was made in a conference in Passau, Germany (Hammer, 1986) whose contents were published later (Crama et al., 1988).

The research team at RUTCOR continued working in this topic regularly mainly at the theoretical level. Meanwhile, the methodology evolved on the practical side: a procedure for transforming raw data into Boolean format was created, to allow LAD to handle arbitrary data, instead of Boolean inputs only, and techniques for handling noise and missing data were added. From 1994 the creation of a software implementation of LAD allowed the first empirical experiments, reported in (Boros et al., 1994). Those results gave practical support to encourage the pursuit of further research in this domain. A technical report of 1996 (Boros et al., 1996) provided a detailed and comprehensive description of the implementation of LAD as a concrete method for data analysis and classification, along with results of empirical experiments on known data sets, as well as some pilot studies. This material has only recently been published in an international journal (Boros et al., 2000).

After 1995, LAD became also one of the research subjects of the Machine Learning Group at IDIAP, and this document assembles most of the developments made in this framework. To support the research activity, the existing software package has been continuously updated at IDIAP to allow the experimentation of all new ideas.

LAD was created in a Boolean function context, and that is the perspective used here to describe its main features. However, the use of logical concepts for classification tasks is not exclusive of LAD, and can be found in other approaches. A parallel with these alternatives is presented in Sect. 2.4.

Before describing LAD, it is useful to define suitable notation and terminology. Below we provide basic Boolean concepts that are used throughout this dissertation. Complete sources on this subject include the books of Mendelson (1970) and Brown (1990).

2.2 Notation and terminology

A *Boolean variable* b is a variable that can take two values, 0 or 1, and these two values form the set $\mathbb{B} \equiv \{0, 1\}$. The *complement* of a Boolean variable $\overline{b}$ is its negation ($\overline{b} = 1 - b$) and the symbols b and $\overline{b}$ are called *literals*.

If $\boldsymbol{x}$ is a vector of B Boolean values, then the function $f : \mathbb{B}^B \to \mathbb{B}$, such that the output of $f(\boldsymbol{x})$ is either 0 or 1, is a *Boolean function* of B variables. Considering that f has an output in $\mathbb{B}$ for every possible input vector $\boldsymbol{x}$, then it defines a partition of the set $\mathbb{B}^B$ into two disjoint subsets $\mathcal{T}$ and $\mathcal{F}$, respectively the subset of true (positive) vectors, and the subset of false (negative) vectors in $\mathbb{B}^B$.

We define a *set of Boolean data* as a set of vectors of Boolean variables, all of the same length B, which is the number of attributes describing the data. Given a set $\mathcal{Y}$ of Boolean data ($\mathcal{Y} \subseteq \mathbb{B}^B$) partitioned into two disjoint subsets of positive and negative data, respectively $\mathcal{Y}^+$ and $\mathcal{Y}^-$, the function defined as:

$$h : \mathcal{Y} \to \mathbb{B} \qquad\qquad h(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad \boldsymbol{x} \in \mathcal{Y}^+ \\ 0 & \text{if} \quad \boldsymbol{x} \in \mathcal{Y}^- \end{cases}$$

is a *partially defined Boolean function* on $\mathcal{Y}$. Given two Boolean functions h and h', h' is said to agree with h if:

$$\begin{aligned} h(\boldsymbol{x}) = 1 \quad &\Rightarrow \quad h'(\boldsymbol{x}) = 1 \\ h(\boldsymbol{x}) = 0 \quad &\Rightarrow \quad h'(\boldsymbol{x}) = 0 \; , \end{aligned}$$

that is, all the true (resp. false) vectors of h are also true (resp. false) vectors of h'. Every *completely defined Boolean function* h' (i.e. defined on the entire set $\mathbb{B}^B$) that agrees with a partially defined Boolean function h is an *extension* of h.

A *term* is a conjunction of literals, and the *degree* G_t of a term t is the number of literals that it contains. We say that a term t *covers* a Boolean vector $\boldsymbol{x}$ if every literal in t is satisfied by the corresponding value in $\boldsymbol{x}$, and this is denoted by: $t(\boldsymbol{x}) = 1$. The *characteristic term* of a Boolean vector $\boldsymbol{x}$ is the unique term of degree $G = |\boldsymbol{x}|$ covering $\boldsymbol{x}$. For example, $\overline{b}_1 b_2 b_3 \overline{b}_4$ is the characteristic term of the vector $(0, 1, 1, 0)$. The *coverage* of a term corresponds to the number of vectors that it covers. It follows naturally that the coverage of a characteristic term of some vector $\boldsymbol{x}$ is equal to 1, because $\boldsymbol{x}$ is the only vector that it covers. In fact, all terms of degree equal to the number of Boolean variables can have coverage of at most 1. As the degree of a term is reduced, it becomes more general and the possibility that it covers more vectors increases.

2.3 LAD in a Boolean function context

2.3.1 The Boolean core of LAD

For the current description, we will make the simplifying assumption that we are in the presence of a classification task where every attribute describing the data is of Boolean type, and where the task consists in distinguishing among two classes. This fits directly into the Boolean framework and allows the core of LAD to be described in a natural way. However, even if many real-life classification tasks have only two classes, it is seldom the case that the attributes are all Boolean. Chapter 3 is dedicated to the problem of transforming data in symbolic or numeric format into Boolean format.

Describing the core of LAD can be done naturally by combining the formalisms presented in Sections 1.2 and 2.2. Considering that the data has B attributes, then the target concept F consists of a Boolean function mapping the input space $\Omega \equiv \mathbb{B}^B$ into the output space $\Sigma \equiv \mathbb{B}$. As explained before, the concept F partitions the input space into the subsets of positive and negative examples, $\mathcal{T}$ and $\mathcal{F}$. The examples are vectors of Boolean values of length equal to the number of attributes B.

For training the classifier we are given a set of Boolean training data $\mathcal{Y}$ representative of the concept, which means that $\mathcal{Y}$ contains data classified as positive and negative according to the concept F. Hence, all the following conditions are satisfied:

$$\mathcal{Y} \equiv \mathcal{Y}^+ \cup \mathcal{Y}^-, \qquad \mathcal{Y}^+ \cap \mathcal{Y}^- \equiv \varnothing, \qquad \mathcal{Y}^+ \subseteq \mathcal{T}, \qquad \mathcal{Y}^- \subseteq \mathcal{F} .$$

The set $\mathcal{Y}$ defines a partially defined Boolean function h, and the goal is to find an approximation $\hat{F}$ of the concept F which is an extension of h. Among the large variety of extensions that possibly exist for the function h, the learner has to choose the one which, given the existing conditions, it will consider to be the most appropriate. This preference is sometimes called *inductive bias* (Mitchell, 1991). Figure 2.1 on the following page shows an abstract representation of the target concept, associated positive and negative regions, and their relation with the training data. The question arises now of how to create the extension $\hat{F}$ that approximates the unknown concept F and that will constitute the classification model.

Every term t that can be formed with B Boolean variables has discriminating properties on the set $\mathbb{B}^B$, because some Boolean vectors in $\mathbb{B}^B$ are covered by t and others are not (except for the empty term, that covers everything in $\mathbb{B}^B$). More precisely, for any subset $\mathcal{Y}$ of $\mathbb{B}^B$ as defined above, there are some terms with interesting discriminating properties, that cover positive examples but no negative ones. Such terms are called *patterns*. As will be seen, patterns are the fundamental concept of the LAD model. We call *positive coverage* and *negative coverage* of a pattern (or a term in general) the number of positive and negative examples, respectively, that it covers. Formally, a pattern p must fulfill the following conditions:

$$\begin{aligned} \exists\, \boldsymbol{x} \in \mathcal{Y}^+ &: \quad p(\boldsymbol{x}) = 1 \quad \text{(the positive coverage of } p \text{ is non-null);} \\ \forall\, \boldsymbol{x} \in \mathcal{Y}^- &: \quad p(\boldsymbol{x}) = 0 \quad \text{(the negative coverage of } p \text{ is null).} \end{aligned}$$

Note that the definition of pattern is necessarily associated with a set of Boolean data partitioned into a positive and a negative subset, as is the case of $\mathcal{Y}$.

As the goal is to build a function discriminating positive from negative data, using patterns is the natural way to accomplish it. A single pattern usually covers only part of the positive instances of a set of data. To achieve complete coverage, one normally needs a set of patterns, denoted by $\mathcal{P}$. If such a pattern set can be found, it implements a classification model, and to classify a new instance we simply need to check if it is covered by any of the patterns in the set. If the answer is positive, then it is a positive instance, otherwise it is a negative one.

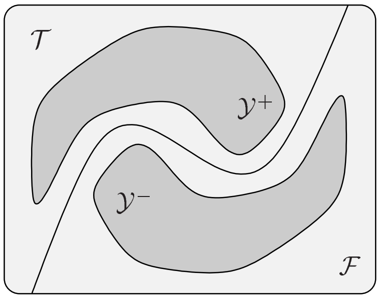


Figure 2.1: Boolean target concept and associated training data.

This is indeed the basic principle of an LAD classification model. In addition, instead of a single pattern set covering the ensemble of the positive instances, two sets are normally employed, one covering the positive instances (the positive pattern set), and the other the negative instances (the negative pattern set). Figure 2.1 tries to give an intuitive explanation for this. The training data usually cover the input space only partially. If the model is based exclusively on one of the classes it will probably be biased. If, on the contrary, it considers both classes, creates a sub-model for each, and finally finds a discriminating function based on these sub-models to generate the final decision, it will be able to attain a better balance[1].

Generating the two pattern sets is done in a similar manner, but considering each of the sets $\mathcal{Y}^+$ and $\mathcal{Y}^-$ alternately. We denote the two pattern sets as $\mathcal{P}^+$ and $\mathcal{P}^-$ respectively. Section 2.3.3 explains how the actual function $\hat{F}$ is defined in LAD, and how the pattern sets are usually weighted, meaning that different patterns receive different weights, according to some measure of importance.

2.3.2 An illustrative example

From the Cleveland heart disease problem, which is one of the data sets used in this project for experimental purposes (the description can be found in Appendix A), we have derived a small and overly simplified example to illustrate the principles of a classification model based on patterns. Note that the conclusions shown in the example are totally artificial and do not correspond in any way to realistic conditions.

[1] A similar principle is used in the Bayesian probabilistic approach for classification, which derives a *decision boundary* from an estimation of the distribution of each one of the two modeled classes (cf. Duda and Hart (1973, Chap. 2)).

The complete data set is described by thirteen attributes of different types (measurable and discrete) and the data consist of observations made on patients (the instances) which are classified into two categories: the positive, meaning that the patient suffers from the specific heart disease being considered, and the negative, corresponding to the opposite conclusion. Naturally, like the vast majority of attributes appearing in real-life classification tasks, these attributes must undergo a transformation in order to produce Boolean variables. From the thirteen original attributes we extracted a subset of four and created one Boolean variable from each, as described below:

Boolean variable	meaning
b_1	age > 54
b_2	chest pain = typical angina
b_3	fasting blood sugar > 120 mg/dl
b_4	serum cholesterol > 262 mg/dl

We also created an artificial training set of data described by these four variables. Figure 2.2 shows the training data in the table on the left. In the same figure, the table on the right shows a set of patterns covering the totality of the examples in the data. There are two positive patterns p_1 and p_2 covering four positive examples each, and two negative patterns p_3 and p_4 covering three and two negative examples respectively. It can be observed that two of the positive examples are covered by both positive patterns, and that the Boolean variable b_2 is not used in the model.

b_1	b_2	b_3	b_4	class	patterns
0	1	0	1	1	p_1
0	0	0	1	1	p_1
1	1	1	1	1	p_1, p_2
1	0	1	1	1	p_1, p_2
1	1	1	0	1	p_2
1	0	1	0	1	p_2
0	1	1	0	0	p_3
1	1	0	0	0	p_4
1	0	0	0	0	p_4
0	0	0	0	0	p_3
0	0	1	0	0	p_3

pattern	b_1	b_2	b_3	b_4	class	coverage
p_1	–	–	–	1	1	4
p_2	1	–	1	–	1	4
p_3	0	–	–	0	0	3
p_4	1	–	0	0	0	2

Positive patterns: $p_1 = b_4$
$p_2 = b_1 b_3$

Negative patterns: $p_3 = \overline{b}_1 \overline{b}_4$
$p_4 = b_1 \overline{b}_3 \overline{b}_4$

Figure 2.2: Artificial example with the Boolean data set of a classification task (left-hand side) and associated pattern sets (right-hand side).

The model has a very intuitive interpretation, and can be stated textually as:

IF the patient has serum cholesterol above 262 mg/dl,
OR IF he/she is older than 54 AND has fasting blood sugar level above 120 mg/dl,
THEN his/her diagnostic is positive.

But, IF he/she is younger than 54 AND his/her serum cholesterol does not exceed 262 mg/dl,
OR IF he/she is older than 54 AND his/her fasting blood sugar level does not exceed 120 mg/dl AND his/her serum cholesterol does not exceed 262 mg/dl,
THEN his/her diagnostic is negative.

Note again that this is an artificial example without real meaning.

2.3.3 The classification theory

Looking at the table representing the patterns in Fig. 2.2, the latter appear as a sort of compression of the data. This is indeed the representational language used for describing a classification model in LAD. A new example can be classified according to the patterns that cover it. However, as patterns are created according to the training data, it is possible that new examples are simultaneously covered by a positive and a negative pattern. For this reason, an additional mechanism is used to remove this kind of ambiguity and generate a definite output for every input. This is done by assigning weights to the patterns so that the positive patterns have positive weights and the negative patterns negative ones. There are different strategies for assigning weights to patterns (Boros et al., 1996):

- based on the positive or negative coverage of the pattern, in order to give preference to those covering more positive or negative examples, depending on the nature of the pattern. This is based on the intuition that patterns with large coverage are strong indicators of the studied phenomenon;
- based on the degree, in order to give preference to short patterns, which are usually better understood by human experts than those containing many literals.

Each one of these base factors can be used in different ways, e.g. the weight can be a linear, quadratic, or exponential function of the coverage. Note that the situation in which an example is not covered by any of the patterns is also possible. In fact, in large problems, it is not probable that the generated pattern sets cover the whole input space. We can think of different alternatives to face such situations. One possibility is to assign the example to a default class, reasonably the most frequent one. Another alternative is to calculate some kind of distance to determine the class that is closest to that example in the input space. Yet another possibility is simply to consider that the example cannot be classified. The best choice will depend on the particular situation.

With the above elements, the definitive form of the LAD classification model can be defined as follows:

$$\hat{F}(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad \Delta(\boldsymbol{x}) > \tau^+ \\ 0 & \text{if} \quad \Delta(\boldsymbol{x}) < \tau^- \end{cases} , \qquad \Delta(\boldsymbol{x}) = \sum_{p_i \in \mathcal{P}^+} w_i \cdot p_i(\boldsymbol{x}) + \sum_{p_j \in \mathcal{P}^-} w_j \cdot p_j(\boldsymbol{x}) \tag{2.1}$$

$$\tau^+ \geq \tau^- \ , \quad w_i \geq 0 \ \ \forall i, \quad w_j \leq 0 \ \ \forall j \ .$$

In the LAD terminology, this is called a *theory*. In general, the thresholds τ^+ and τ^- are both equal to zero. However, they can be set such that $\tau^+ > \tau^-$, in order to create a neutral uncertainty region between the thresholds, in which the value of $\Delta(\boldsymbol{x})$ does not provide enough evidence for a classification decision to be made. Likewise, the center of the decision threshold may be moved away from zero ($\frac{\tau^+ + \tau^-}{2} \neq 0$), in order to account for situations in which the classes are unbalanced (they have different numbers of examples), or if one of the classes is more important than the other. The latter applies in some medical domains, for example, or in person identification systems, in which it is more relevant to strongly model the most important class. This mechanism is similar to the use of class prior probabilities in Bayesian approaches (cf Sect. 1.4.2).

Although the example of Fig 2.2 does not show such situation, it is possible that the number of positive and negative patterns is very unbalanced. To avoid the LAD theory to favor the class with the highest number of patterns, the weights may have to be normalized so that the following equality holds:

$$\sum_{p_i \in \mathcal{P}^+} w_i \ - \ \sum_{p_j \in \mathcal{P}^-} w_j \ = \ \frac{\tau^+ + \tau^-}{2} \ .$$

2.3.4 The search space of patterns

For constructing the LAD model, it is necessary to discover in the training data an appropriate set of patterns. To help describing the pattern finding procedure, we first provide a suitable organization of the search space containing all the possible terms, and then explain how this space is browsed through in order to find the best patterns.

Given the set $\mathcal{B}$ of all possible terms that can be defined with B Boolean variables, it is possible to define the *partial-order relation* $\leq$ on $\mathcal{B}$, meaning *more-specific-than*. For every two terms $t, t' \in \mathcal{B}$, t is more specific than t' if the set of instances covered by t is a subset of the set of instances covered by t' (Mitchell, 1982). With the set $\mathcal{B}$ and the partial order relation $\leq$, we can define a *lattice*, that is, the ordered pair $\langle \mathcal{B}, \leq \rangle$ satisfying the condition that for every two elements $t, t' \in \mathcal{B}$, the set $\{t, t'\}$ has both a *least upper bound* (lub) and a *greatest lower bound* (glb) (Mendelson, 1970). Normal lattices have both a global lub and a global glb. The latter corresponds in fact to the empty term $\varnothing$, since $t \leq \varnothing$ for every non-empty term $t \in \mathcal{B}$. The lub has no interesting meaning in our case, and consequently is left out of consideration.

Graphically, the lattice $\langle \mathcal{B}, \leq \rangle$ can be represented by a *Hasse diagram*, a directed acyclic graph where every edge connecting t' with t means that $t \leq t'$, that is, t is more specific than t' (Brown, 1990). Figure 2.3 on the next page shows a Hasse diagram for the set of Boolean variables $\{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$. As can be seen in the figure, the most natural way of presenting

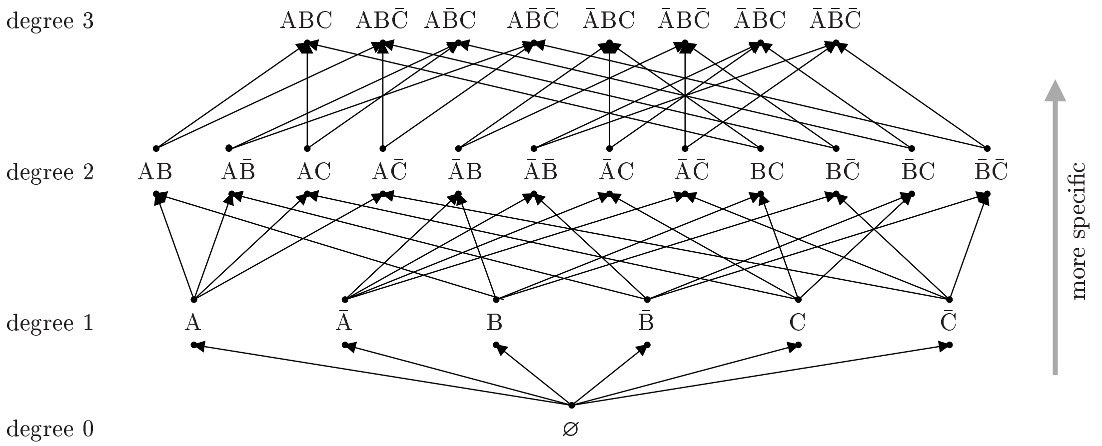


Figure 2.3: Search space containing all terms formed from the set of Boolean variables $\{A, B, C\}$, connected by the partial-order relation $\leq$ (more-specific-than).

the diagram is by structuring it in levels, each level corresponding to a degree.

The nodes in level 1 correspond to the available literals, while those in the upper level are the characteristic terms of the possible instances in $\mathbb{B}^B$. A term t covers an instance $\boldsymbol{x}$ if there is a path in the graph connecting it to the characteristic term of $\boldsymbol{x}$. Adding literals to terms corresponds to specializing them, by appending additional conditions. Likewise, removing literals from terms make them more general.

2.3.5 Pattern finding

Considering again Fig. 2.3, each one of the instances that are represented by the terms in the higher level can be either negative or positive. The goal is to find terms in the space that cover many positive instances and no negative ones for the positive pattern set, and vice-versa for the negative pattern set.

Browsing the whole structure in order to discover the optimal patterns has a prohibitive computational cost in normal situations. The number of terms of degree G that can be formed based on B binary attributes is equal to $2^G\binom{B}{G}$, which is a very rapidly growing function of G and B even when they are small. As Fig. 2.4 on the facing page shows, with degree five this space is already extremely large for problems with a moderate number of binary attributes.

Below we describe a heuristic procedure that does a selective search, which has been proposed in the framework of LAD (Boros et al., 1996), although there are many other possible approaches to this problem, as will be explained later in Sect. 2.4. The procedure described below can be used for finding either positive or negative patterns. It is only a matter of complementing the positive and negative character of the class labels of the training instances. It combines a bottom-up, breadth-first search of terms up to a user-defined degree G_{max} with a subsequent top-down search. The first phase consists of a selective exhaustive search that finds all interesting patterns of degree smaller or equal to G_{max} without visiting all the terms of these same degrees. It is a general-to-specific approach that generates terms of small degree and usu-

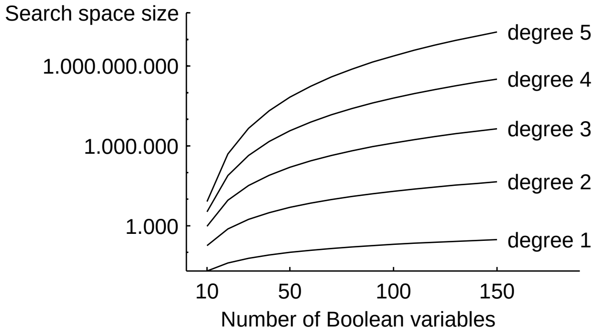


Figure 2.4: Number of possible terms of degrees $G = 1, \ldots, 5$ depending on the number B of Boolean variables.

ally high coverage. The top-down search, on the other hand, is intended to cover instances not covered by the general patterns, and does a specific-to-general search. The two search phases are described in the following paragraphs.

Bottom-up breadth-first search

The bottom-up, breadth-first search is the main search phase, and it starts by generating all the patterns of degree 1, followed by all those of degree 2, and so forth until degree G_{max}. At every stage (a stage being a level in the graph of Fig. 2.3), terms are sorted out according to their type. With regard to the search process, every term of degree G can be of one of the following three types:

(i) a pattern, if it covers some positive instances and no negative one;
(ii) a candidate to become a pattern of higher degree by specialization (addition of literals), if it covers both positive and negative instances;
(iii) an uninteresting term covering no positive instances.

The terms of type (i) are added to the pattern set, those of type (ii) are used to continue the search for patterns of higher degree, and the others (iii) are discarded. After the search is completed in one level, the breadth-first procedure continues to the following level. The terms that are analyzed in level G consist of all the candidate terms of degree $G-1$ with an additional literal. In order to avoid enumerating several times the same term of degree G from different candidates of degree $G-1$, the search makes use of a lexicographical order between literals. For example, a possibility is to use the order:

$$b_1 \prec \overline{b}_1 \prec b_2 \prec \overline{b}_2 \prec b_3 \prec \ldots .$$

So, given a candidate term t, only literals which are higher than any literal in t can be added to form terms of higher degree. This can be illustrated by an example. Figure 2.5 shows the same search space as Fig. 2.3, where the terms are ordered lexicographically and the arrows mean specialization of terms to generate terms of higher degree. The lexicographical restriction allows each term of degree G to be derived from a single term of degree $G-1$. For example, if the terms AB and A$\bar{\text{C}}$ were both candidates of degree 2, they could both give rise to the term AB$\bar{\text{C}}$, by adding the literal $\bar{\text{C}}$ or B, respectively. But if the lexicographical restriction is applied, AB$\bar{\text{C}}$ can only be generated from AB and not from A$\bar{\text{C}}$, since B $\prec$ $\bar{\text{C}}$. Another strategy

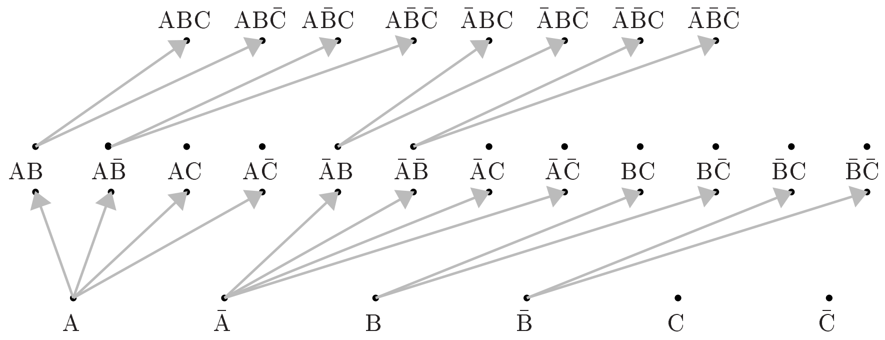


Figure 2.5: Term derivation by specialization. Every term of degree G is derived from an ancestor term of degree $G-1$ by adding a literal which does not precede lexicographically any of the literals contained in the ancestor term.

for reducing the computational effort of the search consists in checking, for every term t of degree G, if all its ancestors[2] of degree $G-1$ were candidate terms. Indeed, if any direct ancestor of t was not a candidate, then it can be discarded immediately without the need for checking its coverage on the data.

This bottom-up, breadth-first search procedure reduces the size of the effective search space by restricting the search to feasible terms. Indeed, every time a term of the types (i) or (iii) is found, the whole subgraph accessible from that term upwards is discarded from the search. This is justified by the fact that going up to higher levels of the graph by adding literals to terms of the current level corresponds to specializing these terms, and thus to limiting their coverage. Considering a term t of degree G :

- if t is of type (i), then it is a pattern, and no pattern of degree higher than G containing t will ever cover more positive examples than t ;
- likewise, if t is of type (iii), then no term of degree higher than G containing t will ever cover a positive example.

The exact computational saving obtained with this selective search process is difficult to quantify, as it strongly depends on the characteristics of the problem. In any case, for problems of large size, it may not be sufficient to bring the computational effort down to an acceptable level, as will be shown empirically later in this dissertation, in Sect. 5.3.1.

Top-down by generalization of terms

The set of patterns generated by the bottom-up procedure does not always cover the whole set $\mathcal{Y}^+$ of positive examples. There may be examples with certain specificities that are not

[2] A term t is an ancestor of another term t' if t is of lower degree than t' and all its literals are also literals of t'.

covered by any of the more general patterns of small degree. To cover these, LAD uses a second procedure that generates patterns by generalizing very specific terms.

The top-down search is executed for every uncovered instance. It starts from the characteristic term of the instance, and removes literals while it still covers only positive instances. This initial term is clearly a pattern, that is generalized as much as possible while keeping its negative coverage null. Figure 2.3 on page 22 can be used to illustrate this process. It starts from the characteristic term t of the instance to be covered, at the top level of the graph. This term covers only that instance. The search process then descends one level, choosing the ancestor of t with the highest positive coverage. The process continues till the moment when all the ancestors of the current term cover one or more negative instances.

This complementary pattern search phase has the problem of being exposed to noise in the data. It can generate very specific terms that only cover a very few instances, and those instances may in reality either be incorrectly labeled (i.e. they belong to the opposite class) or have inappropriate attribute values. The patterns generated under such conditions are akin to produce incorrect classification decisions and reduce the generalization ability of the model.

Pattern filtering

The two phases of pattern generation (bottom-up and top-down) create an extensive set of patterns that usually contains many redundancies. As described above, the bottom-up approach generates all the existing patterns up to a given degree. For this reason, a pattern reduction phase is required to compress the pattern set and create a compact classification model.

Two alternative approaches are considered, a simpler one that reduces patterns that are *subsumed* by other more general ones, and a more sophisticated procedure that minimizes the number of patterns while ensuring the coverage on the whole set of positive data.

A pattern p is *subsumed* by another pattern p' if it covers a subset of the instances covered by p'. In such cases, only the latter is interesting, and so p can be discarded. In principle, the selective breadth-first search procedure described before does not generate any pattern descendent of a previously generated pattern, and so there should not exist subsumed patterns in the set. But this is a condition that is satisfied only by considering the whole space of possible instances $\mathbb{B}^B$. With regard to the set of positive training data $\mathcal{Y}^+$, there can still be patterns covering a subset of other patterns in the set. Consider $\mathcal{Y}^+_p$ and $\mathcal{Y}^+_{p'}$ as the sets of instances in $\mathcal{Y}^+$ covered by the patterns p and p' respectively. Consider also $\mathcal{T}_p$ and $\mathcal{T}_{p'}$ as the sets of positive instances in $\mathbb{B}^B$ covered by p and p', respectively. The breadth-first search procedure will never generate two patterns p and p' such that $\mathcal{T}_p \subseteq \mathcal{T}_{p'}$, but it may generate two patterns satisfying $\mathcal{Y}^+_p \subset \mathcal{Y}^+_{p'}$, in which case p should be eliminated.

Alternatively, patterns can be filtered by a set covering approach, where a minimal set of patterns is kept that guarantees coverage of all the positive examples, as described by Boros et al. (1996). We reproduce its formulation below, for the case of positive patterns and instances. The same procedure must be repeated for their negative equivalents. Consider $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ as the set of positive instances and $p_1, p_2, \ldots, p_P$ as the set of all generated positive patterns. A Boolean matrix $\boldsymbol{Z}$ of size $N \times P$ is used to represent the fact that a positive instance $\boldsymbol{x}_n$ is covered by p_i (in which case $Z_{ni}=1$) or not ($Z_{ni}=0$). Furthermore, the solution for the filtering problem is stored in a Boolean vector $\boldsymbol{y}$ of size P that indicates whether pattern p_i is included in the solution (in which case $y_i = 1$) or not ($y_i = 0$). The following condition guarantees that an instance $\boldsymbol{x}_n$ is covered by at least one pattern:

$$\sum_{i=1}^{P} Z_{ni}\, y_i \geq 1 \ .$$

The goal is to find the minimal set of positive patterns such that the above condition is satisfied for all the positive instances. This can be formulated as a *set covering problem* (SCP) as follows:

$$\begin{aligned} \min \quad & \sum_{i=1}^{P} y_i \\ \text{s.t.} \quad & \sum_{i=1}^{P} Z_{ni}\, y_i \geq 1, \quad \text{for all positive instance } \boldsymbol{x}_n \\ & y_i \in \{0,1\}, \quad i = 1, \ldots, P \end{aligned}$$

Finding the optimal solution for this problem is known to be NP-hard (Garey and Johnson, 1979). However, there are heuristic approaches to solve it in acceptable computational time, eventually at the cost of some sub-optimality. We will come back to this subject again in Sect. 3.2.1, where we describe the usual approach used for solving a set covering problem.

More details on the exact pattern generation procedure and on the data structures used to implement it efficiently can be found in the paper by Boros et al. (1996) mentioned before and in a report by Mayoraz (1995). Various other approaches for finding appropriate patterns in classification contexts have been proposed in the Machine Learning literature along the last two decades. Some of the most prominent are described in Sect. 2.4.

2.3.6 Handling corrupted data

In Sect. 1.1.2, we have mentioned that many classification tasks suffer from the problem of corrupted data. In these cases, the training instances do not provide a totally faithful representation of the concept to be learned. While corruption at the level of missing attribute values is easily detectable, there is no obvious way for a classification algorithm to be aware of the presence of noise, that is, of wrong class labels or inaccurate attribute values. Nevertheless, realistic algorithms must be prepared to handle the noise problem. Here we make a brief reference to the way in which LAD handles corrupted data. A detailed analysis of this problem can be found in other publications (Boros et al., 1996, 1999).

An indicator of classification noise is, for example, the fact that a term covers a large amount of positive instances and at the same time covers some residual negative instances. This might suggest that the negative instances are in fact mislabeled positive instances. A simple way of handling this situation is to relax the definition of pattern in order to accommodate a small negative coverage, namely through the use of a (small) maximal ratio between the positive and the negative coverages. Besides simplicity, this mechanism has the nice property that when a term covers just a few positive examples, the allowed negative coverage is null.

The other problem concerns missing attribute values for certain training (or testing) examples. Whenever an instance has unknown value for a given attribute a, after the data have been transformed into Boolean format, all the Boolean attributes derived from a have also unknown values for this instance. In practice this means that instances can be represented not by strict Boolean vectors, but rather by vectors in $\{0, 1, *\}^B$, where $*$ means "unknown". This has some implications on the notion of *coverage*: for a term t to cover an instance, the latter cannot have an unknown value for any of the Boolean attributes related to literals in t. Unknown values are like *wild cards* that are assumed to always play against the model. So, every instance $\boldsymbol{x}$ with unknown values must be covered by patterns whose literals are all verifiable in $\boldsymbol{x}$. Of course, this option may lead to problematic situations whenever there is a large proportion of missing values, since in this case it will be difficult to cover many instances with a single pattern.

2.4 Related logic-based approaches

In this section we make a resume of classification algorithms that have a close relation to LAD. Most of the techniques that are mentioned below originate in the field of Artificial Intelligence (AI), and most particularly of Machine Learning, while LAD has independently evolved among researchers concerned with Operational Research and with the theory of partially defined Boolean functions in particular.

In the AI domain, a pattern is usually called a *rule*, and it is composed of two parts, the *antecedent* (or condition) and the *consequent* (the class label) as follows:

IF antecedent THEN consequent

In reality, the antecedent part of IF-THEN rules can take various forms. Other than a conjunction of literals, as in the case of LAD and the majority of existing alternative rule-based methods, it can contain disjunctions of literals, or even variables. In the later case they consist of first-order Horn clauses, which form a highly flexible representation language (cf. Sect. 1.4.2). Despite the proximity between the definitions of rule and pattern, it should be mentioned that a rule is in some cases taken in a broader sense, that is, it is allowed to cover negative examples. For example, the algorithm CN2 (Clark and Niblett, 1989; Clark and Boswell, 1991) tries to find classification rules that cover an arbitrary number of examples of any class, and uses a quality measure to select an appropriate subset of such rules. Obviously, the used function forces the rules to cover only positive examples, but no coverage constraints are imposed a priori.

The majority of the rule induction approaches is usually applied to two-class problems only. One of the classes is chosen as the concept to be learned and the goal is to predict whether an example belongs to the concept or not, whereas LAD considers both classes equally and creates a sub-model for each one distinguishing it from the other. Still, some of the methods mentioned below were designed to work with more than two classes, the common approach being to consider each one of the classes in turn as the concept to be learned against the rest of the classes.

Concerning the transformation of numerical or symbolic attributes into logic (Boolean) format, only scarce details can be found in the generality of the literature. In general, continuous attributes are discretized by splitting them into user-defined intervals (e.g. Clark and Niblett (1989)), or considering splits based on the values of the attributes observed in the training data (Cohen, 1995; Venturini, 1993). Chapter 3 of this dissertation is entirely dedicated to this subject.

2.4.1 Exhaustive search

Rymon (1993) applies roughly the same exploration procedure as the bottom-up search of LAD to domains with several classes, and where every rule contains a condition that covers instances of a single class. Instead of a post-filtering phase, rules are discarded during the exploration phase already, which makes it more restrictive than in LAD. A priori rejection criteria may be based on redundancies, where subsumed rules are discarded (as in LAD), and also on inconsistencies, that is, rules which cover examples that are already covered by existing rules but with a different class label are rejected.

Sahami (1995) proposes a method based on the explicit construction of a graph like the one in Fig. 2.3, and where a special type of lattice is considered, with the additional property that every two nodes have a unique common upper bound and a unique common lower bound. Then, instead of creating the entire graph containing the exhaustive set of terms, it constructs the minimal graph supporting the training data. This is done by first inserting the nodes corresponding to the literals (level one), and then inserting in the highest level[3] only the nodes that correspond to instances represented in the training set, and not the whole set of possible terms of degree B. This initial graph does not satisfy the above mentioned property, so intermediate nodes are inserted as necessary until satisfiability is reached. The lattice construction phase allows to label every inserted node with appropriate information (number of instances of each class covered) and the main induction algorithm uses this information to quickly find the appropriate terms, in a *sequential covering* manner, as explained in the subsection below.

In the same line of the bottom-up search applied by LAD, Pijls and Bioch (1999) and Pijls and Potharst (2000) proposed an alternative *depth-first* search procedure that makes use of clever data structures (*tries*) and heuristics to avoid uninteresting search branches and to minimize the computational cost.

2.4.2 Sequential rule induction

The majority of the methods for rule induction found in the literature are based on a *sequential covering* approach (Michalski, 1969), also known as *separate-and-conquer* (Pagallo and Haussler, 1990). Sequential covering methods start with an empty rule set, and generate a first rule to cover as much of the positive examples as possible. The examples covered by that rule are removed from the training set, and another rule is generated to cover the remaining uncovered examples. This process is repeated until all the positive examples are covered by at least one rule or some user-defined minimal error rate is achieved. There is a distinction between models composed of a *rule set* and of a *decision list* (Rivest, 1987). The former may either contain a

[3] In his article, Sahami considers graphs that are inverted with relation to those presented here, where instance nodes are located in the lowest level, while nodes corresponding to conditions with a single-variable are in the highest level.

set of mutually exclusive (non-overlapping) rules, where each instance is covered by at most one rule, or otherwise some weighting or voting method is used to combine multiple-rule matches for a single example. Decision lists, on the other hand, contain overlapping rules that are ordered according to some measure, and new examples are usually classified by the first rule in the list that they match. A default rule (the last in the list) applies when no other rule is activated and it usually has the label of the most frequent class in the data. The order of the rules, in the majority of the sequential rule induction methods, follows the order in which they are generated, or the inverse (Webb, 1994).

Most of the differences found between methods lie in the specific rule induction engine that determines how each one of the rules is obtained. An extensive review of several rule induction algorithms based on sequential covering has been made by Fürnkranz (1999), who classifies the search strategies employed by different methods according to three different parameters: (i) the breadth of the search (greedy unidirectional, using a beam of search paths, or exhaustive search); (ii) the search direction (bottom-up, top-down, or bidirectional); and (iii) the type of quality measure used to evaluate different solutions. We employ a similar classification, although the criterion (iii) is not treated here, but rather in the chapter on pattern generation, more precisely in Sect. 6.4. Fürnkranz makes additional considerations regarding for example rule pruning schemes and representational languages of the different methods, but these are disregarded here, since they are only remotely related to the LAD approach.

Search breadth

Purely greedy methods start with an initial rule and then iteratively add or remove (depending on the adopted search direction) one condition (literal). At each iteration, the condition that is inserted or removed is the one that best improves the quality measure. The PRISM algorithm (Cendrowska, 1987), as well as JoJo (Fensel and Wiese, 1993), and Frog (Fensel and Wiese, 1994) use this approach.

In order to broaden the narrow path followed by standard greedy approaches, some algorithms employ a beam search, that follows several paths simultaneously. In practice, this can be implemented by keeping a certain amount of rules in memory (usually of fixed size), from which new ones are generated in each search step, and only the best rules are kept for the succeeding step. This allows larger portions of the search space to be considered. AQ (Michalski, 1969) and CN2 (Clark and Niblett, 1989; Clark and Boswell, 1991) are two such algorithms.

On the opposite extreme of greedy search are exhaustive search algorithms, that consider all the possibilities, eventually by excluding some parts of the search space that are clearly unattractive. Due to the obvious computational cost of these procedures, they can be employed to search terms (conditions) of limited degree only. This is the case for the bottom-up search employed by LAD and the alternatives referred to in Sect. 2.4.1 on the facing page.

Search direction

Some methods proceed by specialization of terms (bottom-up)[4], others by generalization (top-down), and others using a mixture of both.

In the bottom-up category we find the vast majority of the methods with a uniform search direction. This is the case for CN2 (Clark and Niblett, 1989; Clark and Boswell, 1991), PRISM (Cendrowska, 1987), and the AQ series of algorithms (Michalski, 1969; Michalski et al., 1986; Bloedorn and Michalski, 1991). AQ has the particularity that each rule is searched based on a *seed* positive training example chosen randomly. The rule is constructed in a general-to-specific manner, but every modification must still cover the seed. This is a way of limiting the search space.

Top-down approaches are similar to the complementary pattern search procedure of LAD. They start from a very specific rule matching all the attribute values of a positive example (it is equivalent to a characteristic term) and remove conditions iteratively. This is the case for the DLG method (Webb and Agar, 1992).

Note that top-down approaches will always access only the part of the search space that is reachable from the initial most specific term, while bottom-up search can potentially browse the entire search space, unless additional restricting mechanisms are used, like the one described above for the AQ algorithm.

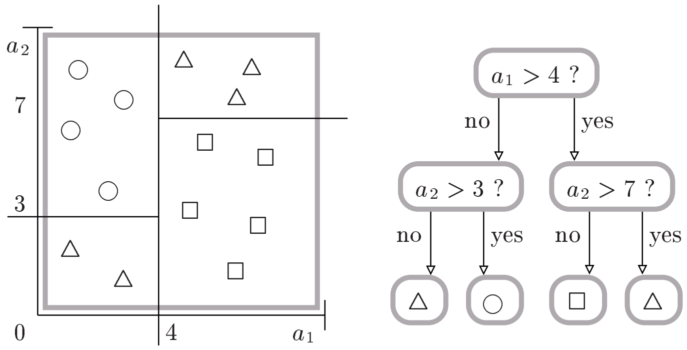
Besides strict bottom-up and top-down directions, methods employing arbitrary search directions have greater freedom. Some of the search strategies in this category apply well-known heuristic-based search approaches, like ATRIS (Kononenko and Kovačič, 1992; Mladenić, 1993) and TABATA (Brézellec and Soldano, 1998), while others have developed their own search heuristic, as it is the case for Frog (Fensel and Wiese, 1993) and JoJo (Fensel and Wiese, 1994). These methods are analyzed in more detail in Sect. 6.4 on page 137 in the chapter dedicated to pattern generation.

2.4.3 Rule induction by recursive partitioning

Rule induction methods that apply recursive partitioning have less affinity with LAD than those described above. However, the principle is presented here with some detail, since it will be useful in later discussions in this dissertation.

The principle of *recursive partitioning*, or *divide-and-conquer* (Quinlan, 1986) consists in splitting the input space recursively into sub-partitions up to a level where every sub-partition contains data of a single class. This idea is closely related to the principle of *decision trees*, which are structures that can accommodate a classification model and that perfectly match the recursive partitioning principle. Each node of the tree contains a test and each possible outcome of a test leads to a different sub-node. At the same time, the test splits the input space in as

[4] Some authors, like Fürnkranz (1999), refer to top-down and bottom-up approaches in the reverse order. The meaning that we use here is consistent with the graphical representation of the search space of terms proposed originally by Mitchell (1982) and the usual representation of Hasse diagrams found in the literature about Boolean reasoning (Mendelson, 1970; Brown, 1990), that is, bottom-up specializations and top-down generalizations.



IF $a_1 \leq 4$ and $a_2 \leq 3$ THEN △

IF $a_1 \leq 4$ and $a_2 > 3$ THEN ○

IF $a_1 > 4$ and $a_2 \leq 7$ THEN □

IF $a_1 > 4$ and $a_2 > 7$ THEN △

(a) (b) (c)

Figure 2.6: Recursive partitioning.

many sub-spaces as outcomes (sub-nodes) of the test. Figure 2.6 shows this clearly. In (a), there is a simple example of an input space with two dimensions containing instances of three different classes. Considering that both attributes a_1 and a_2 take values in the range $[0, 10]$, the whole space is delimited by the gray square box. A recursive partitioning approach starts by splitting the space with the test $[a_1 > 4$?$]$. This corresponds to the higher-level node in the tree in Fig. 2.6 (b) (the root node). Then, each one of the two created sub-spaces may be split independently and generate other nodes. The lower-level nodes (the leaves) are labeled with the class of the examples contained in the corresponding sub-space. In practice, the final sub-spaces may contain a mixture of examples from different classes, which is intended to minimize the *overfitting* of the model to the training data. In this case the class label normally corresponds to the most frequent class in the sub-space.

Important issues related to the construction of decision trees include the choice of the attributes at which splits are made for each node, the specific split point, and also the tree pruning method (either during the construction phase or a posteriori), in order to prevent the tree to match the training data too perfectly and leave room for generalization to unseen data.

A set of rules can be directly derived from a decision tree. One rule can be formed from each one of the leaves. The antecedent of the rule contains the conjunction of the tests of every node in the path followed from the root node to that leaf, and the consequent of the rule contains the same class label as the leaf. Fig. 2.6 (c) shows the set of rules corresponding to the decision tree in (b). The most prominent decision tree methods are CART (Breiman et al., 1984), ID3 (Quinlan, 1986), and its successors C4.5 (Quinlan, 1993) and ASSISTANT (Kononenko et al., 1984; Cestnik et al., 1987). In the specific case of C4.5, Quinlan has developed an algorithm that derives the rule set from a completely-grown decision tree (no pruning is made) and then reduces the rule set both by removing rules and single conditions (literals) of rules.

It can be noted that rules generated by straight recursive partitioning cover the entire input space and, at the same time, are mutually exclusive, as any portion of the input space is covered by a single rule. However, when rule-reduction is done, these conditions no longer apply.

2.4.4 Rule induction with genetic algorithms

Genetic Algorithms (GA) (Holland, 1975; Goldberg, 1989) is the name of an adaptive approach for searching combinatorial spaces composed of a (usually large) set of possible solutions, and where each solution is characterized by a quality measure (evaluated by a *fitness* function). The search performed by GA on the space of solutions corresponds to an optimization process where the goal is to find a solution maximizing the fitness. Usually, very large search spaces (the usual case in nearly all interesting applications) cannot be searched exhaustively in acceptable computation time. GA is a heuristic-based technique that tries to find a near-optimal solution by searching only a subset of the entire search space. The use of heuristic techniques for combinatorial optimization is discussed in more detail in Sect. 6.1.1 on page 113.

In standard GA each solution is represented by a bit string. The search process is iterative, and based on a *population* of solutions, usually of fixed size, that changes across iterations. The way in which the population is modified is inspired by biological evolution. At each iteration, some of the most *fit* solutions are selected in order to diversify the population through so-called *genetic operators* such as *mutation* and *crossover*, which modify or combine the bit strings of the selected solutions to generate new ones. The best final solution is the most fit overall, that is, the one in the final population that optimizes the quality criterion.

Genetic algorithms can be applied to rule learning by coding the rules as bit strings, each bit corresponding to the value of an attribute. The condition of a rule is then a concatenation of bits. There have been two different lines of research in this domain with alternative approaches. One is known as the Michigan approach, where each rule is coded by a bit string as referred to above, and the population of rules forms the classification model (Holland, 1986; Wilson, 1987). Alternatively, the Pittsburg approach codes the entire classification model with a single bit string, containing the concatenation of all the rules in the set, and so a population consists of several classification models. The most rudimentary of these systems required a fixed number of rules (Grefenstette, 1989; Janikow, 1993), while later implementations can handle strings of variable length using more complex genetic operators (Smith, 1983; De Jong and Spears, 1991; De Jong et al., 1993), and hence support models with a variable number of rules.

The SIA algorithm by Venturini (1993) is a somewhat different approach that codes the rules in the same manner as the Michigan implementations, but where the search space is limited by forcing possible rules to match a seed positive example, as in AQ and TABATA (see Sect. 2.4.2 above). In addition, the model is built in a sequential covering manner.

Chapter 3

Mapping arbitrary data into Boolean data

Motivation

This chapter addresses the problem of transforming a set of data in arbitrary numerical format into a corresponding set in Boolean format, in the context of a classification task. This is intended at preparing the input data to be treated by the Boolean core of LAD, which assumes that all the examples are represented by a vector of Boolean values.

The applicability of the results presented here is not restricted to LAD, since several other techniques make use of Boolean concepts, according to the discussion about logic-based classification algorithms in Sect. 2.4. This is the case namely for rule-based and decision tree-based approaches. Most of the existing methods either binarize the original attributes during the construction of the model as needed, or define an extensive Boolean version of the original set based on all the possible combinations of the input attributes. Obviously, the latter approach can only be applied to attributes taking a discrete set of values, and in principle scale badly with the size of the problem, namely the number of attributes and the number of different values they can take. Another possible application of the transformation studied here is data compression, although the domain of application must be able to endure the associated information loss, as will be described.

The results presented in this chapter have been published in two conferences (Moreira et al., 1999; Mayoraz and Moreira, 1999), hence the text elaborates on those publications.

3.1 How data are mapped

The problem addressed here can be stated as defining a mapping $m : \Omega \rightarrow \mathbb{B}^B$ from an arbitrary input space $\Omega \subset \mathbb{R}^A$ into a binary output space $\mathbb{B}^B$, based on a set of data $\mathcal{X} \subset \Omega$, which is partitioned into classes. Coding data in this manner usually implies a loss of information. The mapping must therefore be defined in such a way that certain fundamental properties of the data are preserved during the transformation. At the same time, the size B of the Boolean images must be as small as possible, for reasons that are associated both with the computational complexity of the procedures that use the output produced by m, and of interpretability of the model that will be generated from that data. This creates an interesting optimization problem for which a solution is proposed in the following.

The attributes of the input space can be of any kind: binary, discrete ordered or unordered, and continuous. One of the factors determining the interpretability of the LAD classification model is that each literal of a conjunction (term) is associated with a single original attribute (cf. Sect. 2.3.2). Hence, for simplicity reasons, combinations of original attributes are not admitted. If we consider that the mapping m is composed of B Boolean functions $\Omega \to \mathbb{B}$, then each one of these functions must involve only one original attribute of the input space Ω. Such functions are called *discriminants*, and so every Boolean mapping m corresponds to a set $\mathcal{D}$ of B discriminants. It is easy to see as well that there is a one-to-one correspondence between each discriminant and one of the Boolean variables that will be used as input features of the Boolean core of LAD.

The next section describes how discriminants are generated for each type of attribute considered. To define discriminants, only two comparison operators are used: $=$ and $>$, respectively applied to unordered and ordered attributes. Some authors construct Boolean variables using additional and more complex operators, like $\neq$ or even grouping of different original values (Michalski, 1975; Bergadano et al., 1992). Figure 3.1 shows an example of a transformation

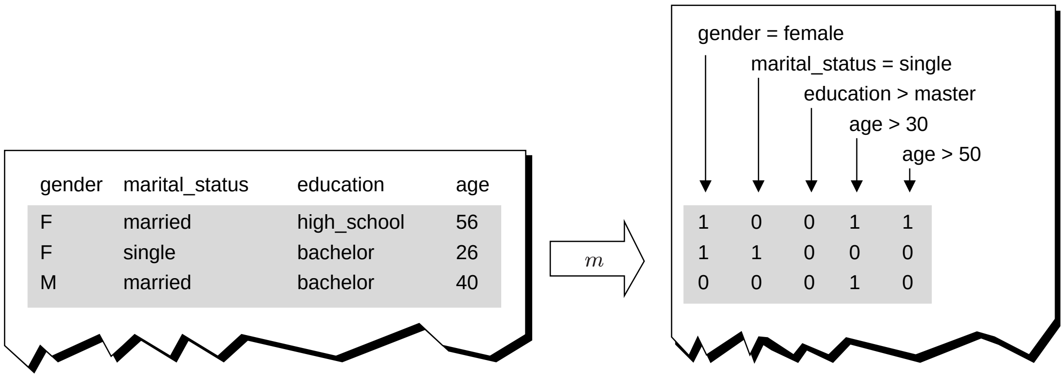


Figure 3.1: Transformation of data into Boolean format.

of data into Boolean format using a mapping m. Note that, as illustrated by the attribute *age* in the figure, there can be multiple discriminants associated with the same original attribute.

3.1.1 Basic attribute types

A description of basic attribute types is given below, as well as the coding of their respective discriminants. These types can represent adequately most of the attributes found in common practical applications.

Binary attributes These attributes can take only two possible values, and therefore correspond naturally to Boolean variables. Examples:

- gender: female, male;
- smoker: yes, no;
- driver's license: yes, no.

Binary attributes can be coded directly as Boolean, even if they are not intrinsically Boolean, like *gender* above[1]. The discriminants are of the type [*attribute=value*] and normally one single discriminant is created from each binary attribute, *value* being one of the two alternatives.

Discrete unordered attributes Attributes of this type can take values in a predefined unordered set. Examples are:

- color: red, green, blue, yellow;
- material: wood, stone, glass, iron;
- nationality: ...

The discriminants are of the type [*attribute=value*], and there can be as many discriminants as the number of different possible values of *attribute*.

Discrete ordered attributes This kind of attribute can take values in a predefined ordered set. Note that despite the existence of an order between different values, there is not necessarily an associated distance measure. Examples:

- academic level: high-school, bachelor, master, doctorate;
- military grade: private, sergeant, lieutenant, captain, major, colonel, general;
- expertise: beginner, advanced, expert;
- frequency: never, rarely, sometimes, often, always;
- quality: bad, fair, good, excellent.

The discriminants are of the type [*attribute*>*value*], and there are usually as many discriminants as the number of different possible values of *attribute* minus one.

Measurable attributes The values of these attributes have both an order and a distance measure. They are inherently numerical, contrary to the above attribute types, which can nevertheless be easily encoded as such. Most of the attributes found in real applications are measurable.

- temperature, age, altitude, cost, quantity, date, etc.

The discriminants are of the type [*attribute*>*value*] for any possible *value* that *attribute* can take.

[1] Note the distinction made here between *Boolean*, taking the values $\{0, 1\}$, and *binary*, taking any two possible distinct values.

The above descriptions show that the discriminants associated with ordered attributes are *thresholds* applied to the range of possible values, while those of unordered attributes are *selectors*, because they select a particular attribute value. In summary:

$$\text{unordered attributes} \begin{cases} \text{binary} \\ \text{discrete} \end{cases}$$

$$\text{ordered attributes} \begin{cases} \text{discrete} \\ \text{measurable} \end{cases}$$

3.1.2 Geometrical interpretation of discriminants

Considering only measurable attributes, the input space Ω can be seen as an Euclidean hyperspace with as many dimensions as the number of original attributes, each one of which being represented by an axis. So, a discriminant d of a measurable attribute a represents a hyperplane intersecting the axis of a at the corresponding threshold, perpendicular to the axis. In this way, d cuts the whole input space into two subspaces, where the examples positioned in the lower subspace or on the hyperplane itself have value 0 (false) for discriminant d, and those positioned in the upper subspace have value 1 (true). A similar interpretation is valid for discrete ordered attributes, although in this case the input space is cut into "slices", each corresponding to one attribute value. The same reasoning does not apply to unordered attributes, though. In fact, due to the absence of order, these are less intuitively interpreted as dimensions of an Euclidean space. Section 3.3 adds more detail to the interpretation of discriminants of different attribute types in the input space.

3.1.3 Consistency of a Boolean mapping

Plain consistency

The mapping m is created using a set of training data $\mathcal{X}$. One possible solution for the problem is to create the *maximal mapping*, i.e. the one composed of the maximal number of discriminants that can be obtained from the attributes that describe the data. This solution is likely to contain too many discriminants, not only from the point of view of the complexity of the model generation procedure, but also because part of them may not necessarily be useful. Still, it may be a good starting point for a reduction procedure aimed at finding a minimal discriminant set that still keeps the useful information contained in the data.

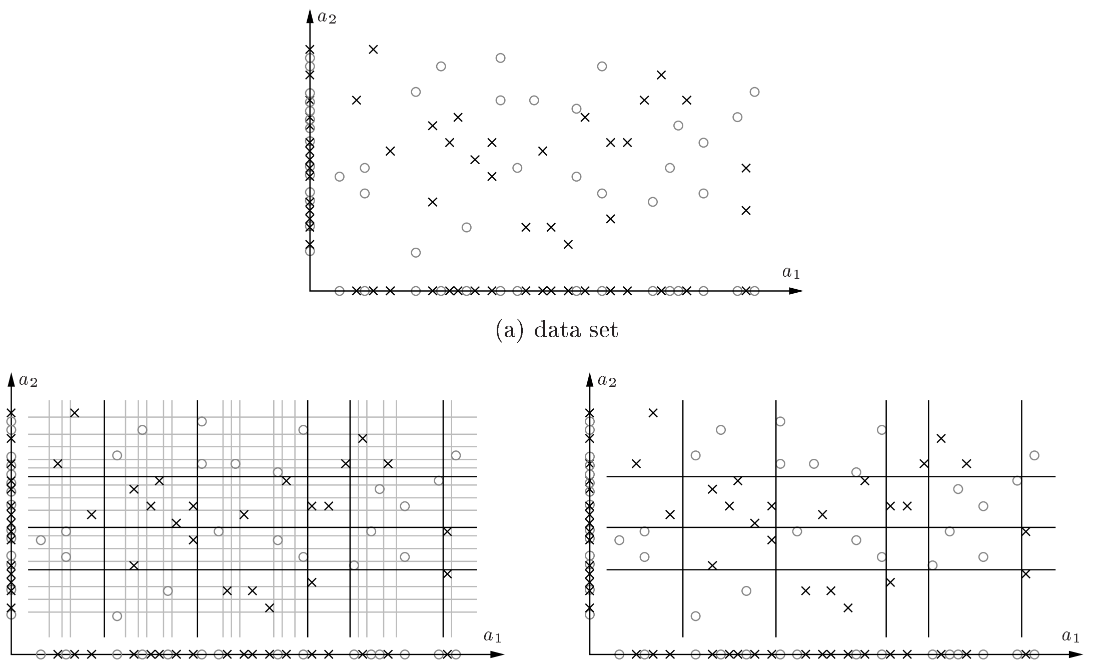
As we are in a classification context, the fundamental property that must be preserved by the mapping m is the class separability. So, we define the following *consistency constraint*: for every two data instances $\boldsymbol{x}$ and $\boldsymbol{x}'$ such that $F(\boldsymbol{x}) \neq F(\boldsymbol{x}')$, the condition $m(\boldsymbol{x}) \neq m(\boldsymbol{x}')$ must hold. We recall that F represents the unknown classification function (cf. Sect. 1.2 on page 4). This means that instances belonging to different classes in the original space must also have different Boolean images. This constraint can be too severe in the presence of noisy data although, as will be discussed in Sect. 3.6, this can be nearly harmless.

In light of the consistency constraint, the maximal mapping can be defined to include, from each attribute a, the following discriminants:

- if a is binary, a discriminant $[a = value]$ is created, if there are two examples $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ such that $F(\boldsymbol{x}) \neq F(\boldsymbol{x}')$, $\boldsymbol{x}_a = value$, and $\boldsymbol{x}'_a \neq value$;
- if a is discrete unordered, a discriminant $[a = value]$ is created for every *value* for which two examples $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ exist such that $F(\boldsymbol{x}) \neq F(\boldsymbol{x}')$, $\boldsymbol{x}_a = value$, and $\boldsymbol{x}'_a \neq value$;
- if a is discrete ordered, a discriminant $[a > value]$ is created for every *value* under the same conditions as for unordered attributes, except for the highest of those values.
- if a is measurable, a discriminant $[a > value]$ is created for every *value* $v = \frac{v_i + v_j}{2}$ under the following condition: there are two examples $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ such that $F(\boldsymbol{x}) \neq F(\boldsymbol{x}')$, $\boldsymbol{x}_a = v_i$ and $\boldsymbol{x}'_a = v_j$, and there is no example $\boldsymbol{x}'' \in \mathcal{X}$ such that $\boldsymbol{x}_a < \boldsymbol{x}''_a < \boldsymbol{x}'_a$, that is, $\boldsymbol{x}$ and $\boldsymbol{x}'$ are consecutive according to attribute a.

It is clear that this approach can be applied in a natural way to problems with any number of classes.

Figure 3.2 (a) shows an artificial data set of two attributes (dimensions) and two classes, while (b) contains the set of discriminants of a maximal mapping, and (c) contains the set of discriminants of a possible minimal consistent mapping. Note in (c) how the consistency constraint translates into the fact that no single "box" contains instances of more than one class.



(a) data set

(b) discriminants of a maximal mapping

(c) discriminants of a minimal consistent mapping

Figure 3.2: Discriminants of a maximal (b) and a minimal consistent (c) Boolean mappings for an artificial data set of two attributes and two classes (a).

Increased consistency

The above discussion is related to *plain consistency*, which refers to the property that every pair of examples from different classes in the training data is distinguished by at least one discriminant. This means that their Boolean images must differ in at least one bit. In practice, it may be useful to guarantee increased consistency. For this purpose, the consistency constraint can be extended in a natural way by establishing that a mapping m is M-consistent with a set of training data $\mathcal{X}$ if and only if for any two examples $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ such that $F(\boldsymbol{x}) \neq F(\boldsymbol{x}')$, the Hamming distance between $m(\boldsymbol{x})$ and $m(\boldsymbol{x}')$ is of at least $M \in \mathbb{N}$. From this it follows that 1-consistency is identical to plain consistency. The aim of using higher levels of consistency is to increase robustness, although the size of the mappings tend to increase as well.

We precise that the notion of M-consistency employed here is a distributed one, which means that each pair of instances from different classes must be discriminated in at least M different original attributes. This implies that the consistency level cannot be higher that the number A of original attributes.

Another issue that must be taken into account is that consistency levels higher than one cannot be guaranteed from the beginning for all the pairs of examples from different classes in a data set, at least using the discriminant coding procedure described above in Sect. 3.1.1. Consider for example the case where two instances $\boldsymbol{x}$ and $\boldsymbol{x}'$ from different classes have equal attribute values except for a certain discrete attribute. Given that discrete attributes are coded with one bit per value in the initial mapping, the Boolean images $m(\boldsymbol{x})$ and $m(\boldsymbol{x}')$ can never differ in more than two bits. So, the required consistency level can only apply to pairs of examples satisfying it already in the initial maximal mapping, but the non-conforming pairs must, on the other hand, never loose the consistency level that they held initially. Even if this situation seems problematic, there is no natural way of handling it without introducing artificial information.

3.1.4 Statement of goal

The goal of the current study is to define a procedure for finding a minimal set $\mathcal{D}$ of discriminants consistent with a given data set $\mathcal{X}$, and allowing to transform the data in $\mathcal{X}$ into a corresponding set $\mathcal{Y}$ in Boolean format. The study considers M-consistency both for $M = 1$ (plain) and for $M > 1$.

3.2 Existing approaches

Before presenting the approach that is proposed in this thesis for finding the mapping m, we describe two alternative approaches that have been proposed by other researchers for the same purpose.

3.2.1 Set covering problem

For an initial discriminant set $\mathcal{D}_{init}$ corresponding to the maximal mapping of a training set $\mathcal{X}$, the problem of finding the set $\mathcal{D}_{final}$ of a minimal consistent mapping for $\mathcal{X}$ can be formulated as a minimum *set covering problem* (SCP). We have already discussed the application of a set covering problem, earlier in Sect. 2.3.5, for reducing pattern sets in LAD. Here we use a similar formulation.

Let $\boldsymbol{Z}$ be a matrix of Boolean values with a column for each discriminant $d \in \mathcal{D}_{init}$ and a row for each pair of instances $(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{X}$ from different classes ($F(\boldsymbol{x}) \neq F(\boldsymbol{x}')$):

$$Z_{(\boldsymbol{x},\boldsymbol{x}'),d} = \begin{cases} 1 & \text{if } d \text{ distinguishes } \boldsymbol{x} \text{ from } \boldsymbol{x}' \\ 0 & \text{otherwise} \end{cases}$$

The solution of the SCP for plain consistency corresponds to finding the minimal set of columns in $\boldsymbol{Z}$ such that for every row there is at least one non-null value. A vector $\boldsymbol{y} \in \{0,1\}^D$ is used to store the solution, and it has value 1 for positions corresponding to discriminants included in the solution, and value 0 for the others. The problem is formulated as follows:

$$\begin{aligned} \min \quad & \sum_{i=1}^{D} y_i \\ \text{s.t.} \quad & \boldsymbol{z}_{(\boldsymbol{x},\boldsymbol{x}')} \geq M, \quad \forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X},\ F(\boldsymbol{x}) \neq F(\boldsymbol{x}'), \qquad \boldsymbol{z} = \boldsymbol{Z}\boldsymbol{y} \\ & y_i \in \{0,1\}, \quad i = 1, \ldots, D \end{aligned}$$

The solution thus obtained guarantees M-consistency. This is the approach for finding the Boolean mapping m used in the LAD implementation described by Boros et al. (1996).

Finding the optimal solution for a minimal set covering problem is known to be NP-hard (Garey and Johnson, 1979), so it must be solved by heuristic approaches. A common greedy heuristic providing good (though potentially sub-optimal) solutions, consists in the following (Chvátal, 1979):

1. start with an empty solution;
2. find the column i in $\boldsymbol{Z}$ containing the largest number of 1's and add the corresponding discriminant to the solution;
3. remove the covered rows from $\boldsymbol{Z}$ (the rows containing a 1 in position i);
4. if $\boldsymbol{Z}$ is empty, stop;
5. go to 2.

Complexity

The size of the matrix $\boldsymbol{Z}$ determines the complexity of the set covering procedure. The number of rows is bounded by $\mathcal{O}(N^2)$, where $N = |\mathcal{X}|$, while the number of columns is bounded by $\mathcal{O}(D)$, the initial number of discriminants. Given that B discriminants are added to the solution on total, that is the number of iterations performed, so the procedure has time complexity in $\mathcal{O}(N^2DB)$ and space complexity in $\mathcal{O}(N^2D)$. Note that for each continuous attribute there may be at most N discriminants, so the quantity D is also bounded by $\mathcal{O}(AN)$, where A is the number of attributes. If this is taken into account, the polynomial bounding the global procedure has degree three on the number of examples. Whether in reality this applies or not has low practical significance, because already in the quadratic case the applicability of the set covering approach to problems with a large number of examples is limited by its complexity with regard to both time and space.

3.2.2 Simple greedy

Almuallim and Dietterich (1991, 1994) have proposed an alternative heuristic-based procedure that attempts to solve the minimum set covering problem mentioned above in lower computational time. Every individual discriminant d splits a set of data into two subsets (*clusters*), one containing the instances for which $d=1$, and the other the instances for which $d=0$. This fact is the key element used to avoid considering pairs of examples, by keeping track of the data clusters that are fragmented by the discriminants selected so far. After the first discriminant is added to the solution, the data is split into two clusters, and the second discriminant splits each one of these clusters in turn, and so on. Whenever new discriminants are added, the existing clusters are further fragmented. When a cluster contains instances of a single class, it is discarded. So, discriminants are added in an iterative manner to split the existing clusters, until no cluster remains, which means that consistency is attained. This is outlined in the following procedure:

1. start with an empty solution and a set $\mathcal{L}$ of clusters containing a single cluster: the whole set of training data ($\mathcal{L} = \{\mathcal{X}\}$);
2. choose the discriminant d that maximizes a given merit function $w(d)$ and add it to the solution;
3. split each cluster $\mathcal{K} \in \mathcal{L}$ into two sub-clusters, according to d ;
4. eliminate from $\mathcal{L}$ all the clusters containing instances from a single class;
5. if $\mathcal{L}$ is empty, stop;
6. go to 2.

At each iteration all the not yet chosen discriminants are tested in order to select the best one, according to a merit function. The authors propose three different merit functions for measuring the interest of a discriminant d, given a subset of already selected discriminants. The first one measures how the entropy of the data would vary with the addition of d. The second measure, called *Simple-Greedy* (SG), counts the number of pairs of instances from different classes that are not yet separated and that d distinguishes. This measure is identical to the one used in the greedy heuristic discussed in the previous section. A third measure is similar to Simple-Greedy and is called *Weighted-Greedy*, because to each pair of instances discriminated by d a weight is associated, which is inversely proportional to the number of other discriminants separating that pair.

Almuallim and Dietterich provide results of classification tasks solved by the ID3 decision tree algorithm (Quinlan, 1986) with a pre-processing phase of discriminant reduction. The results show that when Simple-Greedy is used as the merit function for discriminant reduction, the results are slightly better than with the entropy measure, and that Weighted-Greedy is slightly better than the other two measures. However, only entropy and Simple-Greedy can be implemented efficiently in $\mathcal{O}(NDB)$. This is done by calculating the merit of a discriminant inside each remaining cluster of data, and taking the sum over all the clusters. Note that although the complexity of the algorithm seems exponential due to the recursive splitting of clusters, there can be at most as many clusters as the number of examples. So, any operation that must visit all the clusters is bounded by complexity $\mathcal{O}(N)$.

This algorithm was originally designed for Boolean concepts with a positive and a negative class. The merit function calculated for a discriminant d is therefore:

$$w(d) \;=\; \sum_{\mathcal{K}\in\mathcal{L}} N^{+}_{d=1}(\mathcal{K}) \times N^{-}_{d=0}(\mathcal{K}) \;+\; N^{+}_{d=0}(\mathcal{K}) \times N^{-}_{d=1}(\mathcal{K}) \;, \tag{3.1}$$

where $\mathcal{K}$ is a cluster of data instances and $\mathcal{L}$ is the set of all clusters. In the first iteration, $\mathcal{L}$ contains a single cluster $\{\mathcal{X}\}$ corresponding to the whole set of training data. The quantities $N^{+}(\mathcal{K})$ and $N^{-}(\mathcal{K})$ are the number of positive and negative instances contained in cluster $\mathcal{K}$, respectively, while the subscript indicates that only the subset of them for which the discriminant d is true $(d\!=\!1)$ or false $(d\!=\!0)$ is considered. The value $w(d)$ is the count for each cluster $\mathcal{K}$ of the number of required separations that d would make, accumulated over all the remaining clusters in $\mathcal{L}$.

Simple-Greedy and consistency

The recursive cluster partitioning strategy employed by SG has the benefit of being able to solve the SCP in acceptable time. Because of this, the method has been integrated in the empirical tests performed later in this chapter. However, the relation of SG with M-consistency, for $M > 1$, is not so favorable. Although plain consistency can be verified using clusters, there is no natural way of extending that principle to higher consistency levels.

Generalization of Simple-Greedy

We now provide an extension of the SG heuristic to the general case of K classes, and to the occurrence of examples with missing attribute values, which is not considered by SG. Originally, when a cluster $\mathcal{K}$ is split by a discriminant d, the examples for which $d\!=\!1$ are sent to one side of the split and those for which $d = 0$ to the other. We have introduced a modification in the method that consists in sending all the examples in a cluster $\mathcal{K}$ for which $d\!=\,?$ (meaning unknown value) to both sides of the split made by d in $\mathcal{K}$.

The generalization of the merit function of Eq. (3.1) to the case of K classes is straightforward, and consists in applying it to all different pairs of classes:

$$w(d) = \sum_{\mathcal{K}\in\mathcal{L}} \sum_{i<j} N^{i}_{d=1}(\mathcal{K}) \times N^{j}_{d=0}(\mathcal{K}) \;+\; N^{i}_{d=0}(\mathcal{K}) \times N^{j}_{d=1}(\mathcal{K}) \;.$$

As in the previous formula, the quantities $N^{i}(\mathcal{K})$ and $N^{j}(\mathcal{K})$ are the number of instances contained in cluster $\mathcal{K}$ from classes c_i and c_j, respectively.

Complexity

During the execution of SG, B discriminants are selected to integrate the solution. For each one of them, all the D available discriminants are tested, their merit is evaluated (bounded by NK^2), and the remaining clusters are split (bounded by N). The overall complexity is then $\mathcal{O}(\,B\,(D(NK^2)+N)\,)$ or, put more simply, $\mathcal{O}(BDNK^2)$, which is linear in the number of examples but quadratic in the number of classes and the number B of discriminants in the solution, since $B \leq D$.

3.3 The IDEAL algorithm

The approach that is proposed in this dissertation for finding a minimal mapping that transforms data into Boolean format is called *Iterative Discriminant Elimination Algorithm (IDEAL)*. Contrary to the alternative approaches mentioned in the previous section, it is an eliminative algorithm that starts with an exhaustive discriminant set and then reduces it progressively (Alg. 3.1). The procedure **MaximalMapping** corresponds to the creation of an exhaustive set of discriminants according to the guidelines provided in Sect. 3.1.1. The following sections describe the IDEAL algorithm, explain how the discriminants are chosen for elimination, describe the data structures used and how they are updated, and the stopping criterion.

Algorithm 3.1 MinimalConsistentMapping($\mathcal{X}$ **)**

Inputs: $\mathcal{X}$: set of training examples

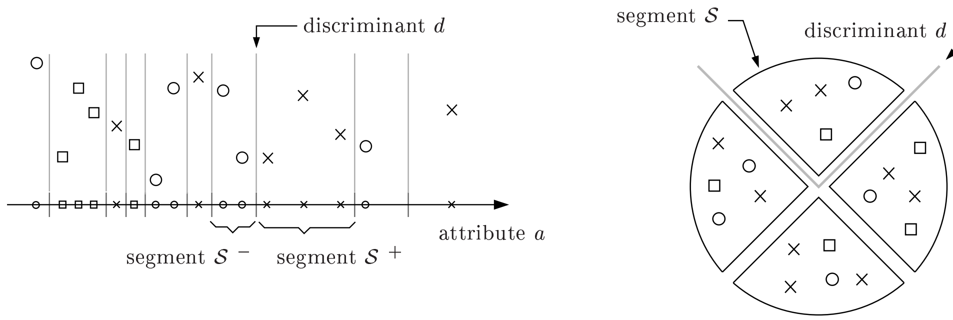
Parameters: M : consistency level

$\mathcal{D}_{init} \leftarrow$ **MaximalMapping**($\mathcal{X}$)
$\mathcal{D}_{final} \leftarrow$ **IDEAL**($\mathcal{D}_{init}$, $\mathcal{X}$)
return($\mathcal{D}_{final}$)

3.3.1 The segment

An important building block of the discriminant reduction procedure is the *segment*, a portion of the input space delimited by discriminants in a manner that depends on the attribute type. For measurable attributes, a segment is the subspace delimited by the hyperplanes of two consecutive discriminants (Fig. 3.3 (a)), which can easily be generalized to discrete ordered attributes. Concerning unordered attributes (binary or discrete), the situation is different, due to the absence of an ordered space. Figure 3.3 (b) is an attempt to represent this case, where the one-to-one relation between discriminants and segments is visible. Note that in this case there is at most one discriminant d per attribute value, according to the discussion of Sect. 3.1.3. Binary attributes, as they contain at most one discriminant, can be considered as a special case of an ordered attribute with regard to segment implementation: there are two segments separated by one discriminant.

When a discriminant is eliminated, a merging of segments occurs. The merging process is explained in the following section.



(a) Ordered attribute

(b) Unordered attribute

Figure 3.3: Segments and discriminants for ordered and unordered attributes.

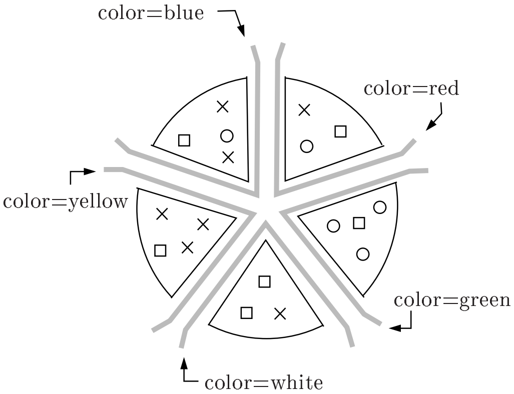
3.3.2 Discriminant elimination

The elimination process chooses discriminants iteratively for elimination. At each iteration, a candidate discriminant is tested for redundancy, which determines whether it is eliminated or kept. The order in which the discriminants are selected is discussed in the next section. For now, let us consider that the discriminants are ranked in the beginning according to some weighting w, and at each iteration the one with the lowest weight is the candidate to be eliminated. Each discriminant d is a candidate at most once: when it is chosen as candidate and its elimination is rejected, it receives infinitively high weight ($w(d) \leftarrow \infty$) so that it is not chosen again.

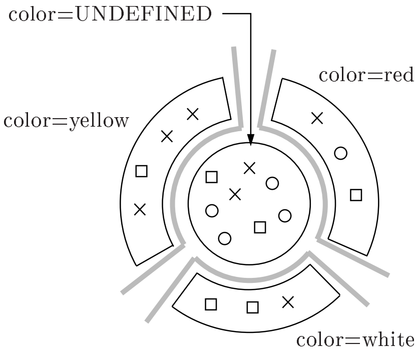
Merging of segments

Merging two consecutive segments of an ordered attribute is quite straightforward: it implies creating a new segment containing the instances of the merged segments, and replacing these in the set of segments by the newly created one. For unordered attributes this operation is less obvious. We illustrate the procedure with an example. Figure 3.4 (a) shows an attribute (color) with five discriminants and corresponding segments. As such, there is redundancy in this discriminant set, since each instance in a segment is discriminated from the instances in the other segments by two discriminants. So, if one of the discriminants is removed, the existing instance separations remain. When this happens, the segment corresponding to the removed discriminant is eliminated, and the instances that it contained are placed in a new segment [*color=UNDEFINED*]. Whenever other discriminants of the same attribute are removed, the instances of the corresponding segment are also placed in the "undefined" segment. Figure 3.4 (b) shows the situation after the elimination of two discriminants: [*color=blue*] and [*color=green*]. Note that every different discrete unordered attribute must have its own "undefined" segment.

The pseudo-code of Alg. 3.2 on the next page illustrates the segment merge procedure.



(a) Initial segments and discriminants



(b) After the elimination of two discriminants

Figure 3.4: Segment processing when the discriminants of an unordered attribute are eliminated.

Algorithm 3.2 MergeSegments(d, $\mathcal{R}$)

Inputs: d : discriminant
$\mathcal{R}$: set of segments

$a \leftarrow$ attribute of d
if a is ordered (discrete or measurable) **then** [*merge the two adjacent segments of* d]
  $\mathcal{S}^-, \mathcal{S}^+ \leftarrow$ adjacent segments of d
  $\mathcal{S}_{new} \leftarrow \mathcal{S}^- \cup \mathcal{S}^+$
  $\mathcal{R} \leftarrow \mathcal{R} \setminus \{\mathcal{S}^-, \mathcal{S}^+\} \cup \{\mathcal{S}_{new}\}$ [*replace in the set of segments*]
else if a is discrete unordered **then** [*merge the segment of* d *into the "undefined" segment*]
  $\mathcal{S} \leftarrow$ segment of d
  $\mathcal{S}_{undef} \leftarrow$ undefined segment of a
  $\mathcal{R} \leftarrow \mathcal{R} \setminus \{\mathcal{S}, \mathcal{S}_{undef}\}$ [*remove both from the set of segments*]
  $\mathcal{S}_{undef} \leftarrow \mathcal{S}_{undef} \cup \mathcal{S}$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathcal{S}_{undef}\}$ [*re-insert in the set of segments*]
end if

Redundancy test

We define a *conflict* as a pair of examples from different classes that need to be discriminated. More specifically, the *local conflicts* resolved by a discriminant d are the conflicts that only d and no other discriminant of the same attribute resolves, even if they are resolved by discriminants of other attributes. This notion is important for testing the redundancy of a discriminant. Indeed, a discriminant is redundant if its local conflicts are resolved by discriminants of other dimensions (attributes) of the input space.

For a discriminant d of an ordered (discrete or measurable) attribute, the local conflicts are the pairs of examples $(\boldsymbol{x}^-, \boldsymbol{x}^+)$ from different classes such that $\boldsymbol{x}^-$ lies in the lower adjacent segment of d, and $\boldsymbol{x}^+$ lies in the its higher adjacent segment. For discrete unordered attributes, Fig. 3.4 (b) gives a clear idea of the local conflicts of a discriminant d : the pairs $(\boldsymbol{x}, \boldsymbol{x}')$ such that $\boldsymbol{x}$ lies in the segment of d, and $\boldsymbol{x}'$ lies in the "undefined" segment of the corresponding attribute.

The redundancy test for d basically consists of the following procedure: for each one of its local conflicts, test if the corresponding pair of examples is already separated by at least one discriminant of another dimension (attribute). If the outcome of the test is positive for all the pairs, d is considered redundant and can thus be eliminated. The pseudo-code of Alg. 3.3 summarizes the procedure. It may be observed that testing whether two examples are separated in another dimension corresponds simply to testing if they lie in different segments in that dimension. We remark also that as soon as the redundancy test fails for one pair of instances, the procedure is immediately aborted.

In case of missing attribute values, a simple mechanism can be included in the procedure **IsRedundant** in order to handle them: for every tested attribute a', if $\boldsymbol{x}^-$ or $\boldsymbol{x}^+$ (or both) have unknown value, they are considered non-separated therein.

Algorithm 3.3 IsRedundant(d, $\mathcal{R}$)

Inputs: d : discriminant
$\mathcal{R}$: set of segments

```
a ← attribute of d
for all local conflicts (x⁻, x⁺) resolved by d do
  conflictResolved ← FALSE
  for all attributes a' ≠ a do
    R_a' ← subset of R containing the segments of attribute a'
    S⁻ ← segment in R_a' where x⁻ lies
    S⁺ ← segment in R_a' where x⁺ lies
    if S⁻ ≠ S⁺ then
      conflictResolved ← TRUE
      go to ◊                                   [jump to next conflict]
    end if
  end for
  if conflictResolved = FALSE then
    return( FALSE )        [at least one conflict is not resolved in another attribute]
  end if
  ◊
end for
return( TRUE )             [all the local conflicts are resolved in other attributes]
```

Higher consistency levels

The implementation of the procedure **IsRedundant** in Alg. 3.3 corresponds to the case of plain consistency, but it can easily be generalized to support M-consistency. For this, we simply initialize a counter to zero before starting the test of a pair of instances in conflict. Then, during the test, instead of immediately stating *conflictResolved* ← TRUE when a conflict resolution is detected in another attribute, the counter is simply incremented. The discriminant is considered redundant for that conflict only after the counter has attained a value of M, which means that the conflict in question is resolved in at least M other attributes.

The algorithm

Algorithm 3.4 presents the pseudo-code of the general IDEAL procedure. The efficient execution of the algorithm relies to a great extent on the existence of data structures representing the sequential positioning of segments and discriminants of the same attribute, as well as the membership of each training instance to its corresponding segment for each attribute. The latter avoids searching all discriminants of an attribute for finding those separating a particular pair of examples, inside the procedure **IsRedundant**. The usefulness of these data structures extends to the updating operations performed after each discriminant elimination. In Alg. 3.4, the

---

Algorithm 3.4 IDEAL($\mathcal{D}$, $\mathcal{X}$)

Inputs: $\mathcal{D}$: initial discriminant set
$\mathcal{X}$: set of training examples

$\mathcal{R} \leftarrow$ **InsertSegments**($\mathcal{X}$, $\mathcal{D}$) *[create the initial set of segments]*
for all $d \in \mathcal{D}$ **do** $w(d) \leftarrow$ **WeightDiscriminant**(d, $\mathcal{R}$)
while there exists a discriminant $d \in \mathcal{D}$ such that $w(\ d\ ) < \infty$ **do**
  $d \leftarrow \arg\min_{d \in \mathcal{D}}(\ w(d)\ )$ *[choose the best candidate for elimination]*
  if IsRedundant(d, $\mathcal{R}$) **then**
    MergeSegments(d, $\mathcal{R}$)
    UpdateWeights(d, $\mathcal{D}$, $\mathcal{R}$)
    $\mathcal{D} \leftarrow \mathcal{D} \setminus \{d\}$
  else
    $w(d) \leftarrow \infty$ *[it must not be candidate for elimination again]*
  end if
end while
return($\mathcal{D}$)

---

procedure **InsertSegments** creates the set of segments corresponding to the initial discriminant set $\mathcal{D}$, according to the description given in Sect. 3.3.1. Algorithm 3.5 on the facing page shows the procedure **UpdateWeights**, which defines which discriminants must be re-weighted, among those that remain after an elimination. The procedure **WeightDiscriminant** is defined in the next section, which discusses the actual discriminant weighting approach. The global complexity of the IDEAL algorithm is given afterwards in Sect. 3.3.4 on page 50.

3.3.3 Weighting the discriminants

Due to the greedy nature of the elimination procedure, the order in which the candidate discriminants are chosen is of major importance. In this section we study four different weighting functions allowing to sort the discriminants in order of importance.

Considering that a discriminant is responsible for making separations between examples of different classes, we can reasonably assume that the more separations it makes the more important it is. So, it seems logical to try to first eliminate those discriminants charged of fewer separations, since: (i) the task of verifying their redundancy will be easier, as there will be less conflicts to test; (ii) given that these discriminants will most probably be redundant,

Algorithm 3.5 UpdateWeights(d, $\mathcal{D}$, $\mathcal{R}$)

Inputs: d : discriminant
$\mathcal{D}$: set of discriminants
$\mathcal{R}$: set of segments

$a \leftarrow$ attribute of d
if a is ordered (discrete or measurable) **then**
  $d^-, d^+ \leftarrow$ adjacent discriminants of d in $\mathcal{D}$
  if $w(d^-) \neq \infty$ **then** $w(d^-) \leftarrow$ **WeightDiscriminant**(d^-, $\mathcal{R}$)
  if $w(d^+) \neq \infty$ **then** $w(d^+) \leftarrow$ **WeightDiscriminant**(d^+, $\mathcal{R}$)
else if a is discrete unordered **then** [*re-weight all the remaining discriminants of* a]
  for all discriminants d' of attribute a ($d' \neq d$) **do**
    if $w(d') \neq \infty$ **then** $w(d') \leftarrow$ **WeightDiscriminant**(d', $\mathcal{R}$)
  end for
end if

that task will be better payed back (less elimination trials failed); and (iii) most importantly, given that the amount of instance pairs that must by separated is fixed, fewer discriminants will remain in the final solution: those that "work the most". The formulations adopted to define the importance of a discriminant are described below.

Number of local conflicts

The number of local conflicts has already been described in the previous section, in the discussion about the redundancy test. It is an intuitive way of measuring the utility of a discriminant. The more local conflicts a discriminant resolves, the more important it is, and the later it will be chosen as a candidate for elimination. The procedure **WeightDiscriminant** used in Algorithms 3.4 and 3.5 calculates the weight of discriminant d based on the following expression:

$$\sum_{i<j} N^i(\mathcal{S}') \times N^j(\mathcal{S}'') \; + \; N^i(\mathcal{S}'') \times N^j(\mathcal{S}') \; .$$

The quantities $N^i(\mathcal{S})$ and $N^j(\mathcal{S})$ represent the number of instances from classes i and j, respectively, contained in segment $\mathcal{S}$. The segments $\mathcal{S}'$ and $\mathcal{S}''$ used in the expression depend on the type of attribute. For ordered attributes, they are the two adjacent segments of d, while for unordered attributes one of them is the segment of d and the other is the undefined segment of the original attribute of d.

Entropy

Given $u_k^{\mathcal{X}}$ as the relative frequency of class k in a data set $\mathcal{X}$ with K classes, the *entropy* $e(\mathcal{X})$, of the data can be computed as

$$e(\mathcal{X}) \;=\; -\sum_{k=1}^{K} u_k^{\mathcal{X}} \;\ln u_k^{\mathcal{X}} \; . \tag{3.2}$$

The word "entropy" has different interpretations. For example, in the domain of physics it has important physical implications as the amount of "disorder" (Weisstein, 1996). The formula above corresponds to the entropy as defined in the domain of information theory (Shannon, 1948), and represents the amount of information needed to transmit the class labels of the examples in $\mathcal{X}$. By convention, when $u = 0$, then $u \ln u = 0$, which allows us to plot this quantity for u in the interval $[0, 1]$ as shown in Fig. 3.5.

From the figure, it can be concluded that the entropy of $\mathcal{X}$ in Eq. (3.2) is lower when each class has either very high or very low frequency. Based on this, we explain in the following the reasoning used in applying entropy for weighting the discriminants. The procedure is based on that of Quinlan (1993) used for the choice of the splits in his decision tree building algorithm C4.5 .

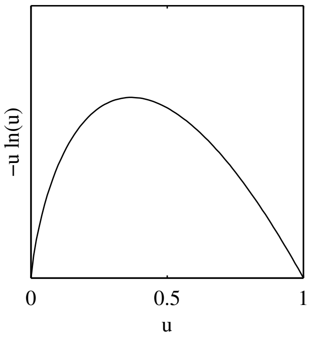


Figure 3.5: Plot of $-u \ln u$.

If a discriminant d splits the data set $\mathcal{X}$ into two subsets $\mathcal{X}^-$ and $\mathcal{X}^+$, then the *split entropy* $e_{split}(\mathcal{X}, d)$ is the sum of the two sub-entropies weighted by the frequency of the respective subsets:

$$e_{split}(\mathcal{X}, d) \;=\; \frac{|\mathcal{X}^-|}{|\mathcal{X}|}\, e(\mathcal{X}^-) \;+\; \frac{|\mathcal{X}^+|}{|\mathcal{X}|}\, e(\mathcal{X}^+) \;,$$

The discriminating power of a discriminant can then be measured as the difference between the two entropy values, before and after the split. This is called the information *gain*:

$$gain(d) \;=\; e(\mathcal{X}) \;-\; e_{split}(\mathcal{X}, d) \;, \tag{3.3}$$

and it represents the reduction in the quantity of information needed to describe the class labels of the data. It follows that the higher the gain of a discriminant, the higher its utility. Equation (3.3) is the one used in the procedure **WeightDiscriminant** for weighting the discriminants based on the information gain, but given that the factor $e(\mathcal{X})$ is constant it can be dropped.

Local and global measures

The locality of the measure based on the number of conflicts (it acts at the segment level) contrasts with the global nature of the entropy, defined over the whole data set. No clear indication exists, however, about the advantages of a local measure over a global one or vice-versa, a fact that led to the development of alternative versions of each of the two measures: a local version of the entropy, calculated over the two adjacent segments of d, and a global version of the number of conflicts, calculated over the two complete subsets of $\mathcal{X}$ generated by the split of d.

After every discriminant elimination some discriminants must be re-weighted, as indicated in Alg. 3.5 on the preceding page. Note, however, that in the case of the global measures the weights are static and only need to be calculated once in the beginning of the IDEAL algorithm.

Empirical comparison of weighting measures

We have conducted experiments aimed at comparing the different weighting functions empirically. We measured the size of the discriminant sets obtained with the IDEAL algorithm using each one of the functions: number of conflicts, both local and global, as mentioned above, and the local and global versions of the entropy. In addition, we used random choice to check whether the other measures find practical justification.

The tests are based on the 5×2cv scheme described in Appendix B, but here we did not do cross-validation tests, we just applied the IDEAL algorithm to the training data and discarded the testing data, since there is no testing procedure. The data sets were split into two equal-sized parts and each part was considered separately as training data. This was repeated five times, and then the means and standard deviations of the measured quantities were calculated over the ten experiments. All the twenty-two data sets described in Appendix A were used.

Table 3.1 on page 56 at the end of this chapter contains the average size of the initial discriminant set, as defined by the **MaximalMapping** procedure, followed by the average final sizes obtained with each different weighting function. The numbers in smaller font size show the standard deviations, and the bottom row gives the average final/initial size ratio of the discriminant sets for each column, considering all the data sets. Figure 3.6 gives a different perspective of the same results, with the plots of the final/initial size ratio by data set and by weighting function. Smaller values are better, both in the table and in the figure.

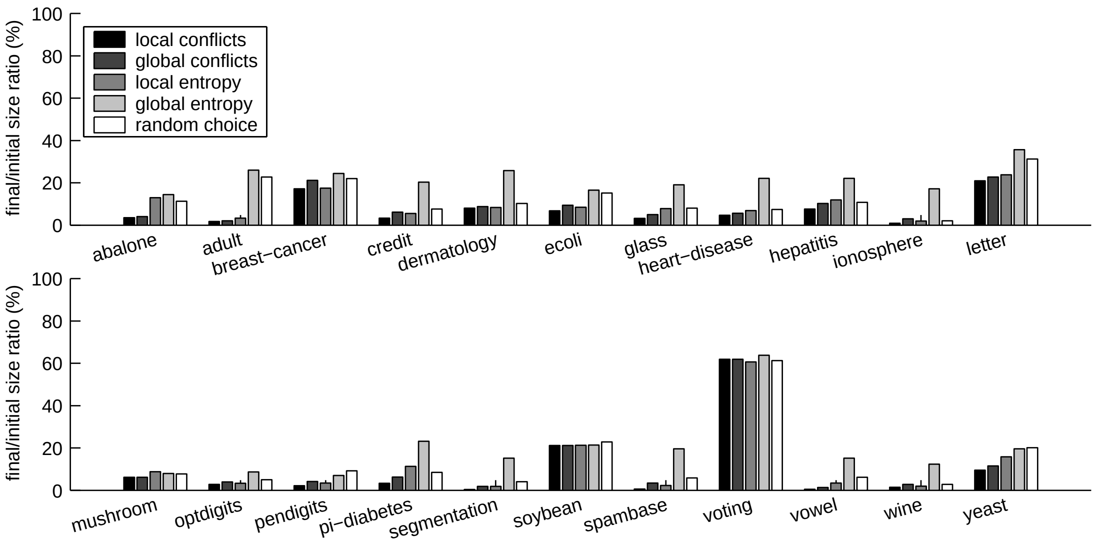


Figure 3.6: Ratio between the final and the initial size of the discriminant sets, using the IDEAL algorithm with different discriminant weighting functions. Lower bars are better. Table 3.1 on page 56 contains equivalent results in numerical format.

There is a clear trend in the results. The measure based on the number of local conflicts gives systematically the greatest size reduction, while the global entropy is the worst measure overall, even worse than random on average, which suggests that this is not a good criterion to optimize. The table shows that these two measures (global entropy and random) have a very unstable behavior, with high standard deviations in many cases. The measure based on the number of local conflicts is also the most stable, as its standard deviations are the lowest overall.

In general, the local versions are better than the global ones. We explain this by the fact that the local measures provide a better account of the interactions between the different discriminants, while the global ones consider each discriminant individually. If we also consider that the entropy measure is more adapted to work globally when the information gain criterion is applied, then the advantage of the number of local conflicts seems natural. But the most logical explanation for the success of this weighting measure may be the fact that it is closely related to the notion of redundancy used in the IDEAL algorithm.

In light of the obtained results, the number of local conflicts is adopted in all the remaining experiments as the standard discriminant weighting function of IDEAL.

3.3.4 Computational complexity

In this section we analyze the time complexity of IDEAL. The algorithm needs $\mathcal{O}(AN \log N)$ to sort the instances by attribute and insert the segments, with A being the number of attributes. For each one of the D initial discriminants, the following operations are executed: calculate its weight (bounded by K^2 if the number of local conflicts is used as weighting measure), sort it relatively to other discriminants ($\log D$), and test its redundancy (AN^2). Concerning the latter, the bound N^2 is actually quite loose, since it concerns only a small subset of the training data located in the adjacent segments of the discriminant. The global bound is given by:

$$\mathcal{O}(AN \log N + DK^2 + D \log D + DAN^2) .$$

However, N is usually much higher than A or K, and is higher than D, so the bound reduces to:

$$\mathcal{O}(DAN^2) .$$

3.4 Empirical evaluation

As can be observed from the above descriptions, IDEAL is an eliminative algorithm, as it builds a maximal mapping and then reduces it iteratively. The set covering problem (SCP) usually employed by LAD (Sect. 3.2.1) and the Simple-Greedy (SG) algorithm (Sect. 3.2.2) are constructive methods that start with an empty discriminant set and add discriminants iteratively. All the three approaches take the consistency constraint into account, but while SCP and SG add discriminants until consistency is attained, IDEAL never looses consistency during the elimination phase; it eliminates redundancy instead. This is similar to the relation between *primal* and *dual* methods usually considered in optimization theory (Cook et al., 1998), i.e. in eliminative methods (the primal) the initial solution is already feasible (in our case, it means that it respects the consistency constraint), as well as all the other considered solutions, while in constructive ones (the dual) feasibility is attained during the optimization process. There is another difference between SG and the other two approaches that concerns the level of consistency that can be guaranteed. As already mentioned, SG cannot be used for finding mappings with a consistency level higher than one.

In this section we measure empirically the performance of the IDEAL algorithm when the consistency level is raised (cf. Sect. 3.1.3), and compare it with the constructive procedure implemented by the SG algorithm. There are three quantities measured: (i) the size of the obtained mapping, (ii) the execution time of the procedure, and (iii) the generalization ability allowed by the mapping.

The goal of measuring the generalization ability is to evaluate how the consistency enforced on the training data is maintained on the testing data. To do this, we have constructed classification models using the Boolean attributes obtained by the different tested mappings as input features. We assume that, with a fixed noise level for each data set, the differences in the classification accuracy (cf. Sect. 1.1.2) obtained with different mappings are due to the mappings themselves. The models have been built using the C4.5 decision tree learning algorithm (Quinlan, 1993), which is able to solve multi-class problems. C4.5 is an evolution of a more rudimentary decision tree algorithm ID3 (Quinlan, 1986), and which handles continuous-valued attributes and missing attribute values. At learning time, it first generates a decision tree model that closely fits the training examples, and subsequently operates a pruning phase, in order to collapse certain sub-trees into single nodes. This aims at reducing overfitting of the model to the training data and improving the estimated generalization error. As mentioned before (Sect. 3.3.3 on page 48), C4.5 uses an entropy measure in order to decide how to split each node into sub-nodes. This algorithm is largely cited in the Machine Learning literature and is well-known for its very good tradeoff between short execution time and good classification accuracy in a large variety of domains (see eg. Lim et al. (2000)). Another advantage is that it shows a good performance without the need for parameter tuning for each task.

Similarly to the tests made for evaluating different discriminant weighting functions above (Sect. 3.3.3) the current experiments are based on a 5×2cv scheme. For the results concerning the number of discriminants and the execution time no cross-validation is done, only the means and standard deviations are calculated for the training sets. But for the results concerning classification accuracy the generated classification model is indeed applied to the testing data and the 5×2cv statistical test is applied, according to the procedure defined in Appendix B. All the twenty-two data sets described in Appendix A have been used here.

The obtained results are presented and discussed in Sections 3.4.1 to 3.4.3 below. Tables 3.2 to 3.4, at the end of the chapter, contain the detailed numerical results corresponding to each of the presented figures.

3.4.1 Discriminant reduction rate

As expected, Fig. 3.7 shows that the final number of discriminants grows with increasing consistency levels. For certain data sets, it grows more than linearly, notably for `adult` and `ecoli`, as can clearly be seen in Table 3.2. An interesting result is obtained by SG on two of the largest problems, `abalone` and `adult`, for which a higher reduction is obtained as compared to IDEAL. For all the other data sets, the performances of SG and IDEAL with consistency level one are comparable.

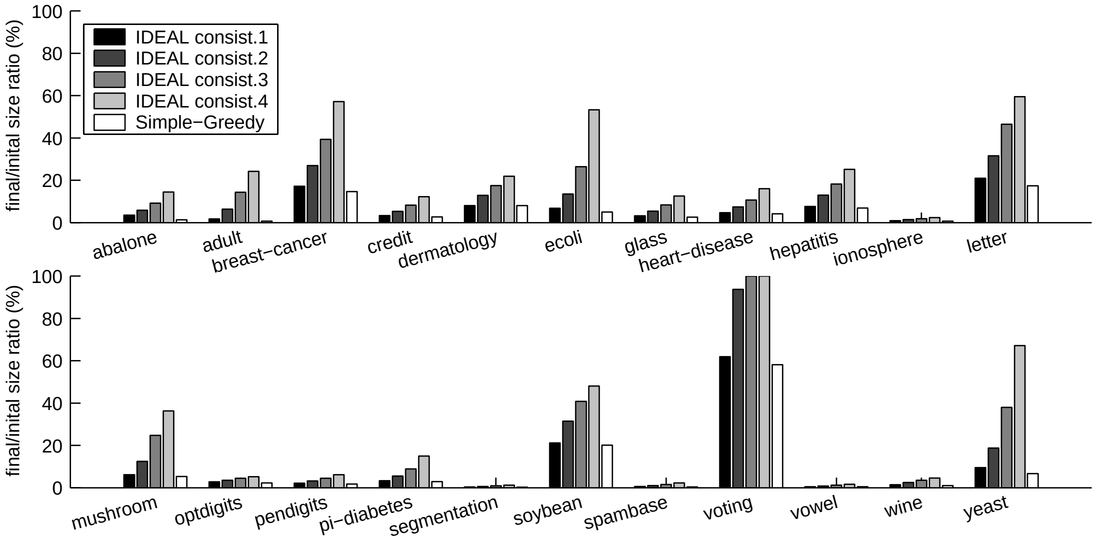


Figure 3.7: Ratio between the final and the initial size of the discriminant sets, using the IDEAL algorithm with different consistency levels and the Simple-Greedy algorithm. Lower bars are better. Table 3.2 on page 56 contains equivalent results in numerical format.

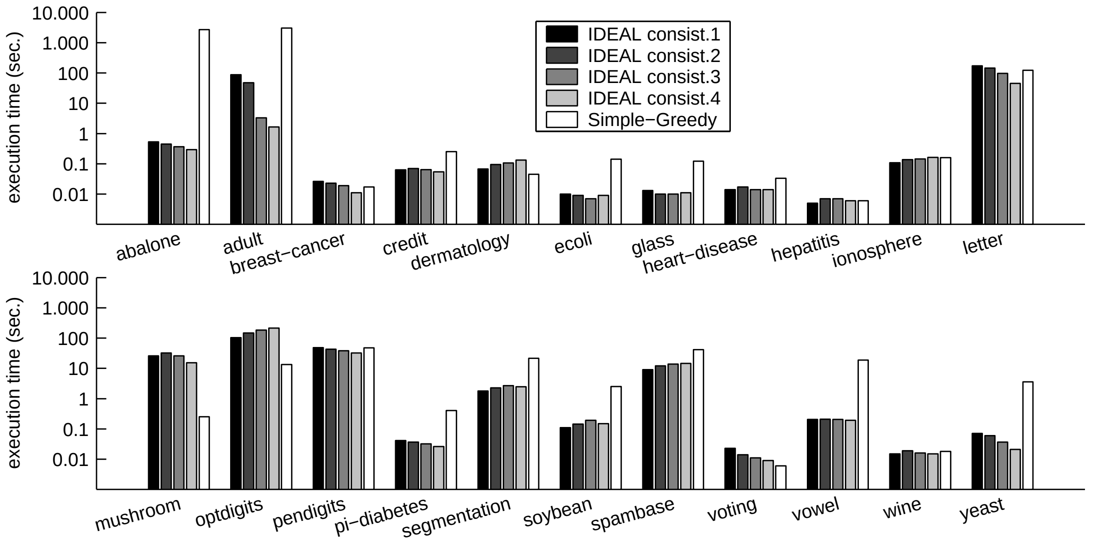


Figure 3.8: Execution time of IDEAL with different consistency levels and of Simple-Greedy. The bars give the average calculated on 5×2cv runs, in seconds, so lower bars are better. Note the logarithmic scale of the vertical axis. Table 3.3 on page 57 contains equivalent results in numerical format.

3.4.2 Execution time

It can be seen in Fig. 3.8 that both procedures are rather fast in all cases, except SG that takes a very long time to solve the problem for the `abalone` and `adult` data sets. It is interesting to verify that these are precisely the cases where SG achieves considerably better reduction rates than IDEAL (see Table 3.2 on page 56). In the majority of the cases IDEAL is faster. Concerning IDEAL with different consistency levels, it can be observed that there is no significant change in execution time, which increases with the consistency level for some data sets, and decreases for others. These tests have been performed on a Sun Sparc Ultra 10 workstation running at 440 MHz.

A closer look at Tables 3.2 and 3.3 reveals that the data sets for which SG is considerably slower than IDEAL are exactly those where the initial number of discriminants is high. Specifically the first two, `abalone` and `adult`, where the execution time of SG is exceedingly high, combine a large initial number of discriminants (D) with a large number of examples (N). Indeed, the discriminant weighting procedure of SG is bounded by $\mathcal{O}(DNK^2)$. In the case of IDEAL, the few discriminants that are affected by an elimination are immediately re-weighted and so all the weights are permanently kept sorted. At each iteration the best discriminant is immediately retrieved, there is no need to test the weights of all of them to determine the best. In addition, the operation consisting in testing the redundancy of the remaining discriminants is fast, because many redundancy tests fail prematurely for non-redundant discriminants.

Yet another difference in the algorithms that may condition the execution time is related to the fact that SG is incremental while IDEAL is eliminative. Incremental procedures are supposed to be faster when few discriminants are needed to guarantee consistency, comparing to the number of discriminants in the maximal mapping. Eliminative procedures, on the other hand, must eliminate all the redundant discriminants until a minimal state is attained.

3.4.3 Classification accuracy

These results provide the most important measure for comparing different procedures. What the plots of Fig. 3.9 show is that in general small differences are obtained. An important conclusion is that by raising the consistency level we can effectively raise the accuracy in most cases, but only in a few of them the difference is statistically significant. The comparison between IDEAL with plain consistency and SG shows that in general their performance is quite comparable. Overall, the higher consistency levels allow the best results with a few exceptions.

3.5 Related work

There are two subjects usually treated by Machine Leaning researchers which have some similarity with the task of finding an appropriate set of discriminants. One of them involves the discretization of continuous attributes and the other consists of the selection of relevant subsets of input attributes. These subjects are discussed below and their relation to the current problem is identified.

3.5.1 Feature discretization

Feature discretization is a technique that involves the transformation of attributes taking continuous values into discrete attributes with a fixed set of possible values. This is usually done

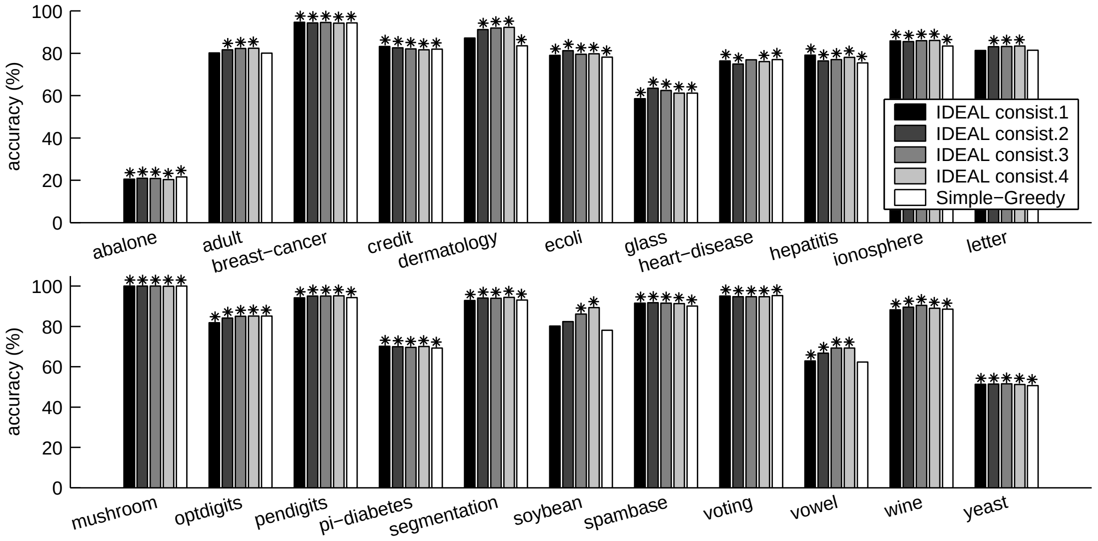


Figure 3.9: Classification accuracy obtained when using IDEAL with different consistency levels or Simple-Greedy, as a pre-processing step. The bars give the average calculated on 5×2cv runs, and higher bars are better. Table 3.4 on page 57 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value (or group of values), with statistical significance at a confidence level of 0.95.

by splitting the range of each attribute into intervals, the equivalent of segments (Sect. 3.3.1). The technique is usually applied as a preprocessing step for certain learning algorithms suited for symbolic attributes, like classification rules and decision trees (cf. Sect. 2.4), although some of them discretize continuous attributes during the construction of the model. But even for algorithms handling continuous-valued attributes, preliminary discretization may accelerate the induction process, simplify the obtained models, and even improve accuracy (Pfahringer, 1995; Dougherty et al., 1995; Hussain et al., 1999).

The discriminants that form the Boolean mapping m obtained with the IDEAL algorithm can be used to discretize continuous attributes. On the other hand, every symbolic attribute can be transformed into a set of Boolean attributes, one for each one of its values. But despite this close relation there is a clear distinction between the approach discussed here and the task of discretizing continuous attributes: usual discretization methods consider one continuous-valued attribute at a time and transform it into a discrete attribute, whereas IDEAL considers all the original attributes at once, independently of their type, and globally optimizes, subject to a constraint (consistency), a mapping whose images are placed in a Boolean feature space.

3.5.2 Feature selection

Yet another domain where approaches similar to the one discussed here can be found is that of *feature selection*. The general goal consists in selecting subsets of attributes by rejecting those that are irrelevant or misleading for the learning task and keeping the most relevant ones. The main motivations include accuracy improvements, reduced training times, and simpler classification models. John et al. (1994) classify existing procedures within this domain into *filter* and *wrapper* methods. The latter use the accuracy of the learned models as objective function during the subset search, while the filter methods consist strictly of preprocessing steps using only the training data.

The task described here can be seen as a feature selection problem, where each discriminant is a feature. In that sense, IDEAL and the other mentioned approaches are filter methods. Another filter method named RELIEF (Kira and Rendell, 1992; Kononenko, 1994) assigns a quality measure to each feature and selects only those that are above a quality threshold.

3.6 Discussion

In this chapter we have presented the IDEAL algorithm, which tries to find an optimal mapping allowing to transform classification data into Boolean format. The algorithm has been compared with Simple-Greedy, an alternative method proposed earlier in the Machine Learning community, and we have shown that they have comparable performance in the vast majority of the twenty-two tested tasks. Simple-Greedy has proved capable of finding significantly shorter mappings in two of the tasks, but at the cost of a highly increased execution time. In any case, the classification accuracy obtained by a learning algorithm using either of the two approaches as a pre-processing step does not differ significantly. However, the IDEAL procedure was able to create the mapping in a very short computational time (less than four minutes) in all the twenty-two data sets used, even those containing tens of thousands of examples.

Another aspect that has been tested concerns the degree of robustness imposed on the mapping, represented by the level of differentiability between the Boolean images obtained for examples of different classes (the consistency level). While Simple-Greedy is limited to plain consistency, this is a parameter that can be changed by the user in the IDEAL algorithm, and we have shown that raising the consistency level can eventually improve the classification results significantly, although in some cases this implies the use of large mappings where the image associated with each example is a long vector of Boolean values. This can be a problem for the classification algorithm that uses the transformed data, as it has to deal with a very high dimensional input space.

A weakness of the discussed procedures concerns their inability to deal with noisy data sets in an efficient manner. As they all rely on the notion of consistency, the occurrence of a misclassified example inside a cluster of examples of another class may demand certain additional discriminants just to cope with that particular situation. Of course, this may not harm the learning algorithm, as long as it is able to deal itself with noise and to ignore the information provided by the corresponding additional Boolean attributes. Nevertheless, it would be desirable to introduce mechanisms in the IDEAL algorithm allowing this useless information to be discarded from the mapping.

	AVERAGE	FINAL NUMBER OF DISCRIMINANTS				
	INITIAL	NUMBER OF CONFLICTS		ENTROPY		RANDOM
DATA SET	N.DISCR.	LOCAL	GLOBAL	LOCAL	GLOBAL	CHOICE
abalone	4140.9	**147.1** ±4.0	171.6 ± 8.0	538.0 ±23.7	600.4 ±101.3	471.2 ±127.6
adult	8008.7	**141.6** ±2.9	171.6 ± 5.7	268.3 ±21.5	2083.7 ± 27.3	1819.7 ±210.2
breast-cancer	67.3	**11.6** ±1.1	**14.3** ± 1.3	**11.8** ± 1.3	16.4 ± 1.4	**14.8** ± 1.4
credit	511.0	**17.1** ±1.0	31.9 ± 3.6	28.7 ± 2.9	104.4 ± 14.2	**39.1** ± 23.4
dermatology	128.0	**10.4** ±0.8	**11.3** ± 1.3	**10.8** ± 1.6	33.0 ± 3.6	**13.1** ± 1.7
ecoli	235.7	**16.0** ±1.2	22.2 ± 2.2	**20.1** ± 3.6	39.0 ± 2.7	36.0 ± 3.3
glass	401.1	**12.9** ±1.3	20.3 ± 2.3	31.5 ± 8.3	76.8 ± 17.3	**32.5** ± 9.6
heart-disease	220.1	**10.5** ±0.7	**12.4** ± 2.6	**15.1** ± 2.7	48.7 ± 5.5	**16.3** ± 3.4
hepatitis	112.6	**8.6** ±0.7	**11.6** ± 3.5	**13.5** ± 2.8	24.8 ± 5.0	**12.1** ± 2.6
ionosphere	1201.2	**10.9** ±1.1	**35.2** ±32.7	**24.5** ± 7.7	**204.6** ±106.6	**25.0** ± 7.5
letter	231.5	**48.6** ±1.4	**52.7** ± 2.6	55.1 ± 2.6	82.6 ± 2.1	72.5 ± 6.2
mushroom	112.9	**7.0** ±0.0	**7.0** ± 0.0	**9.9** ± 1.3	**9.0** ± 0.0	**8.8** ± 1.0
optdigits	816.9	**22.9** ±0.9	**32.8** ± 3.7	27.5 ± 1.3	71.5 ± 20.6	41.4 ± 2.9
pendigits	1569.2	**34.0** ±0.6	**66.2** ±19.9	54.4 ± 3.7	110.1 ± 6.3	144.8 ± 24.3
pi-diabetes	561.1	**18.6** ±1.6	35.1 ± 3.3	63.6 ±10.7	129.9 ± 5.9	47.6 ± 12.8
segmentation	6436.5	**24.6** ±1.6	124.0 ±22.8	118.3 ±26.1	981.4 ±209.3	262.6 ± 80.7
soybean	98.0	**20.8** ±1.0	**20.8** ± 1.0	**20.9** ± 0.8	**21.0** ± 0.8	**22.4** ± 1.8
spambase	5575.4	**34.7** ±2.1	194.9 ±12.2	128.1 ± 6.3	1096.0 ± 22.8	**328.9** ±366.4
voting	16.0	**9.9** ±0.9	**9.9** ± 0.9	**9.7** ± 1.1	**10.2** ± 0.9	**9.8** ± 1.2
vowel	3842.1	**21.8** ±1.2	52.1 ±10.7	134.7 ±13.8	583.4 ±291.4	236.0 ± 55.9
wine	428.0	**6.4** ±1.3	**12.0** ± 6.4	**8.6** ± 2.5	53.0 ± 27.1	**11.9** ± 4.9
yeast	330.5	**31.7** ±1.3	38.1 ± 3.8	52.3 ± 4.9	65.0 ± 3.6	66.6 ± 7.5
AVER. % OF INITIAL SIZE		8.6%	10.2%	11.1%	20.8%	13.8%

Table 3.1: Final number of discriminants obtained with IDEAL using different discriminant weighting functions. The numbers give the average ± standard deviation calculated over 5×2cv runs, and lower values are better.

	AVERAGE	FINAL NUMBER OF DISCRIMINANTS				
	INITIAL	IDEAL WITH DIFFERENT CONSISTENCY LEVELS				SIMPLE
DATA SET	N.DISCR.	LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4	GREEDY
abalone	4140.9	147.1 ±4.0	245.4 ± 4.0	380.9 ± 6.1	601.5 ±11.9	**57.3** ±10.0
adult	8008.7	141.6 ±2.9	510.5 ±13.2	1151.0 ±10.2	1939.8 ±13.1	**58.8** ± 8.5
breast-cancer	67.3	11.6 ±1.1	18.1 ± 1.6	26.5 ± 1.7	38.5 ± 1.6	**9.9** ± 0.7
credit	511.0	**17.1** ±1.0	27.4 ± 2.0	42.5 ± 2.8	62.5 ± 2.2	**14.2** ± 1.2
dermatology	128.0	**10.4** ±0.8	16.5 ± 1.0	22.4 ± 1.6	28.0 ± 1.6	**10.4** ± 0.9
ecoli	235.7	16.0 ±1.2	32.0 ± 2.0	62.4 ± 4.1	125.7 ± 3.7	**11.8** ± 1.5
glass	401.1	12.9 ±1.3	22.0 ± 1.2	33.7 ± 2.7	50.3 ± 3.6	**10.4** ± 1.1
heart-disease	220.1	**10.5** ±0.7	16.4 ± 1.6	23.6 ± 1.7	35.3 ± 4.1	**9.3** ± 0.6
hepatitis	112.6	**8.6** ±0.7	14.6 ± 1.0	20.5 ± 1.6	28.3 ± 1.8	**7.8** ± 0.6
ionosphere	1201.2	10.9 ±1.1	17.1 ± 1.5	22.7 ± 2.2	28.3 ± 3.2	**8.2** ± 1.0
letter	231.5	48.6 ±1.4	73.2 ± 1.3	107.6 ± 2.7	137.7 ± 2.1	**40.3** ± 2.1
mushroom	112.9	**7.0** ±0.0	14.1 ± 0.3	28.0 ± 1.0	41.0 ± 0.8	**6.0** ± 0.0
optdigits	816.9	22.9 ±0.9	29.5 ± 0.5	36.5 ± 0.9	42.9 ± 1.1	**18.6** ± 0.5
pendigits	1569.2	34.0 ±0.6	51.2 ± 1.8	71.2 ± 2.2	96.8 ± 1.5	**27.2** ± 0.7
pi-diabetes	561.1	**18.6** ±1.6	31.1 ± 1.7	50.2 ± 1.7	84.1 ± 2.9	**16.3** ± 1.3
segmentation	6436.5	24.6 ±1.6	41.9 ± 1.6	60.6 ± 2.3	83.7 ± 2.1	**20.2** ± 1.2
soybean	98.0	**20.8** ±1.0	30.9 ± 0.9	40.0 ± 1.1	47.1 ± 1.6	**19.7** ± 1.7
spambase	5575.4	34.7 ±2.1	58.0 ± 2.3	85.1 ± 4.6	127.0 ± 8.2	**24.4** ± 2.3
voting	16.0	**9.9** ±0.9	15.0 ± 0.6	16.0 ± 0.0	16.0 ± 0.0	**9.3** ± 0.9
vowel	3842.1	21.8 ±1.2	33.8 ± 0.7	47.9 ± 1.4	66.3 ± 1.6	**18.7** ± 0.8
wine	428.0	**6.4** ±1.3	10.7 ± 0.9	15.2 ± 1.2	19.6 ± 1.6	**4.7** ± 0.6
yeast	330.5	31.7 ±1.3	62.2 ± 1.2	125.5 ± 2.9	222.0 ± 5.7	**22.2** ± 2.5
AVER. % OF INITIAL SIZE		8.6%	13.8%	19.5%	26.7%	7.5%

Table 3.2: Final number of discriminants obtained with IDEAL for different consistency levels, and for Simple-Greedy. The numbers give the average ± standard deviation calculated on 5×2cv runs, and lower values are better.

	Execution time of the procedure, in seconds				
	Ideal with different consistency levels				Simple
data set	level 1	level 2	level 3	level 4	Greedy
abalone	0.53 ± 0.02	0.45 ± 0.02	0.36 ± 0.02	**0.29** ±0.01	2709.01 ±697.33
adult	87.88 ± 8.50	47.66 ± 9.62	3.32 ± 0.31	**1.66** ±0.07	3065.14 ± 75.16
breast-cancer	0.03 ± 0.00	**0.02** ± 0.00	**0.02** ± 0.00	**0.01** ±0.00	**0.02** ± 0.00
credit	0.06 ± 0.00	0.07 ± 0.01	**0.06** ± 0.01	**0.05** ±0.00	0.25 ± 0.02
dermatology	0.07 ± 0.00	0.09 ± 0.00	0.11 ± 0.01	0.13 ±0.03	**0.04** ± 0.01
ecoli	**0.01** ± 0.00	**0.01** ± 0.00	**0.01** ± 0.00	**0.01** ±0.01	0.14 ± 0.02
glass	**0.01** ± 0.00	**0.01** ± 0.00	**0.01** ± 0.00	**0.01** ±0.00	0.12 ± 0.02
heart-disease	**0.01** ± 0.00	**0.02** ± 0.00	**0.01** ± 0.00	**0.01** ±0.00	**0.03** ± 0.00
hepatitis	**0.01** ± 0.01	**0.01** ± 0.00	**0.01** ± 0.00	**0.01** ±0.00	**0.01** ± 0.00
ionosphere	**0.11** ± 0.02	**0.14** ± 0.03	**0.15** ± 0.00	0.16 ±0.01	**0.16** ± 0.03
letter	173.42 ±11.98	144.79 ±14.97	96.70 ± 5.30	**45.05** ±2.69	123.85 ± 3.60
mushroom	25.96 ± 0.31	32.35 ± 2.40	26.07 ± 2.67	15.29 ±1.20	**0.25** ± 0.00
optdigits	104.60 ±11.82	147.86 ±20.68	183.97 ±10.66	215.72 ±9.50	**13.33** ± 0.42
pendigits	48.54 ± 2.65	42.99 ± 3.39	**38.30** ± 2.05	**32.25** ±2.15	48.07 ± 1.32
pi-diabetes	**0.04** ± 0.01	0.04 ± 0.01	**0.03** ± 0.00	**0.03** ±0.00	0.40 ± 0.01
segmentation	**1.79** ± 0.09	**2.28** ± 0.16	2.67 ± 0.13	2.46 ±0.12	21.69 ± 0.98
soybean	**0.11** ± 0.01	0.14 ± 0.01	**0.19** ± 0.13	0.15 ±0.01	**2.51** ± 1.06
spambase	**9.08** ± 0.47	12.12 ± 0.68	13.91 ± 1.30	14.56 ±0.83	41.80 ± 1.79
voting	**0.02** ± 0.00	**0.01** ± 0.00	**0.01** ± 0.01	**0.01** ±0.01	**0.01** ± 0.00
vowel	**0.21** ± 0.01	0.21 ± 0.01	0.21 ± 0.00	**0.19** ±0.01	18.72 ± 0.89
wine	**0.01** ± 0.01	**0.02** ± 0.00	**0.02** ± 0.00	**0.02** ±0.01	**0.02** ± 0.00
yeast	0.07 ± 0.01	0.06 ± 0.02	0.04 ± 0.00	**0.02** ±0.00	3.61 ± 0.13

Table 3.3: Execution time of IDEAL with different consistency levels and of Simple-Greedy. The numbers give the average ± standard deviation calculated on 5×2cv runs, in seconds, so lower values are better.

	Classification accuracy				
	Ideal with different consistency levels				Simple
data set	level 1	level 2	level 3	level 4	Greedy
abalone	**20.57** ±1.08	**21.02** ±0.90	**20.89** ±0.77	**20.34** ±1.07	**21.64** ±0.83
adult	80.22 ±0.59	**81.63** ±0.40	**82.22** ±0.20	**82.39** ±0.31	80.06 ±0.33
breast-cancer	**94.62** ±1.32	**94.33** ±0.82	**94.59** ±1.21	**94.19** ±1.77	**94.36** ±1.36
credit	**83.22** ±1.71	**82.58** ±2.33	**82.06** ±2.56	**81.60** ±2.10	**81.94** ±1.71
dermatology	87.22 ±2.28	**91.20** ±3.07	**91.92** ±3.64	**92.19** ±3.72	**83.49** ±3.68
ecoli	**79.05** ±1.98	**81.25** ±3.21	**79.57** ±4.32	**79.74** ±3.49	**78.21** ±2.07
glass	**58.54** ±4.74	**63.45** ±6.17	**62.43** ±6.71	**61.20** ±5.66	**61.13** ±2.57
heart-disease	**76.43** ±1.74	**74.91** ±4.87	76.90 ±3.20	**76.04** ±4.03	**77.03** ±3.10
hepatitis	**79.10** ±2.70	**76.37** ±5.92	**77.01** ±4.98	**78.06** ±4.02	**75.48** ±3.19
ionosphere	**85.87** ±3.25	**85.47** ±3.96	**85.93** ±2.99	**86.04** ±2.90	**83.42** ±2.86
letter	81.32 ±0.66	**83.08** ±0.67	**83.26** ±0.49	**83.40** ±0.45	81.43 ±0.60
mushroom	**100.00** ±0.00	**100.00** ±0.00	**99.99** ±0.04	**99.98** ±0.05	**100.00** ±0.00
optdigits	**81.84** ±1.23	**84.14** ±1.29	**85.07** ±0.85	**85.21** ±1.50	**85.12** ±0.60
pendigits	**94.21** ±0.71	**95.12** ±0.37	**95.03** ±0.37	**95.20** ±0.29	**94.29** ±0.44
pi-diabetes	**70.16** ±1.95	**69.97** ±1.61	**69.66** ±1.24	**70.08** ±2.68	**69.30** ±1.31
segmentation	**92.84** ±1.51	**94.08** ±0.79	**94.00** ±0.82	**94.46** ±0.98	**93.07** ±0.94
soybean	80.23 ±3.22	82.40 ±2.39	**86.15** ±2.59	**89.37** ±1.93	78.13 ±2.37
spambase	**91.61** ±0.94	**91.81** ±0.63	**91.59** ±0.79	**91.33** ±0.50	**90.17** ±1.01
voting	**95.13** ±1.15	**94.71** ±0.80	**94.71** ±0.80	**94.71** ±0.80	**95.31** ±1.04
vowel	**62.89** ±3.62	**66.75** ±2.31	**69.33** ±1.49	**69.33** ±2.43	62.36 ±2.13
wine	**88.21** ±3.74	**89.56** ±3.06	**90.46** ±2.78	**89.01** ±3.74	**88.53** ±3.45
yeast	**51.36** ±1.95	**51.48** ±1.77	**51.54** ±1.23	**51.28** ±1.30	**50.73** ±1.47

Table 3.4: Classification accuracy obtained when using IDEAL with different consistency levels or Simple-Greedy, as a pre-processing step. The numbers give the average ± standard deviation calculated on 5×2cv runs, and higher values are better. The numbers typed in bold indicate the best value (or group of values) of a row, with statistical significance at a confidence level of 0.95.

Chapter 4

Solving multi-class classification problems using binary classifiers

Motivation

In this chapter, we study the application of binary (two-class) classification algorithms to solve multi-class classification problems. We analyze some common approaches to this problem, their strengths and weaknesses, and discuss the general principles and guidelines involved. At the same time we also propose some novel solutions to the problem.

While it is true that this study has been motivated by the need of providing LAD with multi-class classification capabilities, it is also a fact that the discussed methods can be applied to other learning algorithms, be they binary or not. This study can thus be interpreted as an intermediate step towards the development of a more specific approach, closely adapted to LAD, which is discussed in the next chapter.

We start by providing the basic notions associated with the decomposition of classification problems in Sect. 4.1, followed by an analysis of certain existing approaches to that purpose in Sect. 4.2. In Sect. 4.3 we propose an alternative approach and finally in Sect. 4.4 we present and discuss an empirical comparison involving a total of eight methods. Part of the contents of this chapter have been elaborated based on two previously published articles (Mayoraz and Moreira, 1997; Moreira and Mayoraz, 1998).

4.1 Decomposing a classification problem

4.1.1 Basic procedure

We recall from Sect. 1.2 in the introductory chapter (page 4) that learning a classification task consists in finding an approximation $\hat{F}$ of an unknown function $F : \Omega \to \Sigma$, which maps an arbitrary input space $\Omega \equiv \mathbb{R}^A$, where A is the number of input variables, into a discrete, usually unordered set $\Sigma \equiv \{c_1, ..., c_K\}$ of classes. Therefore, the approximation $\hat{F}$ implements a *decision rule* that partitions the input space Ω into K disjunctive *decision regions* $\Omega_k, k=1,\ldots,K$, which means that the classification of unseen instances is determined by the region Ω_k in which they lie. The generality of the learning algorithms used for classification generate a model which implicitly or explicitly represents the *decision boundaries* between the regions $\Omega_k, k=1,\ldots,K$.

A single decision boundary is able to separate two distinct regions, hence it is naturally suited for two-class problems. Let us assume that such a boundary is implemented by a function $\hat{f}$: $\Omega \rightarrow \{-1,+1\}$ whose outputs represent the two regions located in each side of the boundary. Note that the actual form of $\hat{f}$ is arbitrary: it may be a simple separating line or hyperplane placed in the input space, or a much more complex structure. What is fundamental is that it can only distinguish two different kinds of objects. Hereafter we will name a function of this kind a *dichotomy*, which is an implementation of a binary classification model.

The models created by LAD, as well as those created by Linear Discriminant Functions and Support Vector Machines for example (cf. Sect. 1.4.2), are all dichotomies, since these methods are adapted to solve classification problems with only two classes. In this case, the classification model is composed of a single dichotomy and the decision rule $\hat{F}$ is simply equivalent to the output of the respective dichotomy:

$$\hat{F}(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}) \qquad\qquad \hat{f} : \Omega \rightarrow \{-1,+1\}\ . \tag{4.1}$$

To allow these methods to be applied to problems with a larger number of classes we require techniques that decompose the original problem into simpler problems involving only two classes. Namely, the use of a *decomposition scheme* allows the transformation of a K-partition $F : \Omega \rightarrow \{c_1, ..., c_K\}$ into a series of Q bipartitions $f_1, \ldots, f_Q$. Each one of the bipartitions usually assigns a positive or a negative character to the different classes in the problem, and they are approximated by a set of dichotomies $\hat{f}_1, \ldots, \hat{f}_Q$ generated by the binary learning algorithm. A *reconstruction scheme* is coupled with each decomposition scheme in order to combine the answers of all the Q dichotomies for a particular input and select one of the K classes.

The above reasoning presents the decomposition as a necessary operation, since there is no other way of applying the concerned methods to multi-class tasks. However, there are other reasons for decomposing a classification problem, independently of the capabilities of the learning method. Several studies have demonstrated the advantages of solving simpler classification sub-problems rather than the whole problem at once, and these include both gains in accuracy and reductions in the complexity of the classification sub-models. For example we cite several studies in the domain of Automatic Speaker Recognition, where the mentioned benefits have been achieved by applying several classification models specialized on small subsets of speakers, instead of a large system trained to make the distinction between all the speakers at once (Rudasi and Zahorian, 1991; Castellano et al., 1997; Zahorian et al., 1997; Genoud et al., 1998). These benefits will be demonstrated empirically later in Sect. 4.4.

Well-known decomposition schemes

The most intuitive, well-known, and applied decomposition scheme consists in implementing K different dichotomies $\hat{f}_1, \ldots, \hat{f}_K$, each discriminating one of the K classes from all the others, and we call it *one-per-class* (OPC). Another natural scheme that is widely used is the *pairwise-coupling* (PWC), which contains a dichotomy to discriminate between each pair of different classes. This scheme requires $\frac{K(K-1)}{2}$ dichotomies and for each pair of classes (c_i, c_j) there is a dichotomy trained to discriminate between them and ignore all the others.

The decomposition matrix

In general, decomposition schemes are defined such that each dichotomy associates some classes in the original problem with a positive label and others with a negative one. The overall decomposition scheme can be specified by a *decomposition matrix* $\boldsymbol{D}$ of size $Q \times K$ such that

$$D_{qk} = \begin{cases} +1 & \text{if class } c_k \text{ is positively considered by } \hat{f}_q \\ -1 & \text{if class } c_k \text{ is negatively considered by } \hat{f}_q \\ 0 & \text{if class } c_k \text{ does not belong to the task of } \hat{f}_q \end{cases} .$$

Note how the balanced ternary set $\{-1, 0, +1\}$ can be used to denote the three possible status of a class in a dichotomy: positive $(+1)$, negative (-1), or ignored (0). For example, the decomposition matrices of the OPC and PWC schemes are shown in Fig. 4.1 (a) and (b) for the case $K = 4$. But a decomposition scheme must not necessarily be intuitive as in the two mentioned examples, it can take many different forms, provided that a basic condition is respected, which will be described later in Sect. 4.1.3. Figure 4.1 (c) shows an arbitrary decomposition matrix.

$$\begin{pmatrix} +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 \end{pmatrix}$$

(a) one-per-class

$$\begin{pmatrix} +1 & -1 & 0 & 0 \\ +1 & 0 & -1 & 0 \\ +1 & 0 & 0 & -1 \\ 0 & +1 & -1 & 0 \\ 0 & +1 & 0 & -1 \\ 0 & 0 & +1 & -1 \end{pmatrix}$$

(b) pairwise-coupling

$$\begin{pmatrix} +1 & -1 & -1 & -1 \\ +1 & -1 & -1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & -1 & +1 & +1 \\ +1 & +1 & -1 & -1 \\ +1 & +1 & -1 & +1 \\ +1 & +1 & +1 & -1 \end{pmatrix}$$

(c) arbitrary

Figure 4.1: Decomposition matrices of two classical decomposition schemes plus an arbitrary scheme, for problems with four classes. Each row corresponds to one dichotomy and each column to one class.

4.1.2 Reconstruction

Every decomposition scheme has an associated reconstruction scheme that combines the answers of the dichotomous classifiers into a global class decision. Below we present the general reconstruction framework, after making the distinction between hard and soft reconstructions.

Hard and soft decisions

The output of the vector $\hat{\boldsymbol{f}}$ of dichotomies is a vector $\hat{\boldsymbol{f}}(\boldsymbol{x})$ in $\{-1,+1\}^Q$ that can be compared with the columns of the decomposition matrix $\boldsymbol{D}$. In this case, the class corresponding to the column in $\boldsymbol{D}$ which is closest to $\hat{\boldsymbol{f}}(\boldsymbol{x})$ in Hamming distance should be the winning class[1]. We call this decision process a *hard reconstruction* because the outputs of the dichotomies are limited to the set $\{-1,+1\}$. As will be described below, there is an alternative way of selecting the winning class.

The general form of a dichotomy $\hat{f} : \Omega \to \{-1,+1\}$ can usually be split into different components. All the learning algorithms mentioned in the previous section for solving two-class problems are based on models that do not genuinely produce a value in $\{-1,+1\}$, but rather a real number that is combined with a decision threshold for producing the actual classification decision. For the case of LAD, this can be clearly observed in Eq. (2.1) (Sect. 2.3.3 on page 20) which shows its general decision rule. We note that in Eq. (2.1) the output is defined in $\{0,1\}$, whereas in the current discussion we assume that the output of a dichotomy is in $\{-1,+1\}$. The latter form is useful for the formalism employed here, and this difference should in no case create a problem, since either the internal theory can be modified, or the output can be easily scaled to $\{-1,+1\}$.

For reconstructing the global class decision from partial binary decisions, it may be useful to have access to the value produced by a dichotomy before the thresholding, since it carries more information. In general, if the decision threshold of a theory is placed at 0.0, an output value of 0.9 provides a stronger positive decision than one of 0.2, but if the threshold is applied they both produce the same output (+1). So, we will assume that each bipartition f is approximated by a dichotomy $\hat{f} : \Omega \to \mathbb{R}$, and in this case the decision rule for a two-class problem is:

$$\hat{F}(\boldsymbol{x}) = \begin{cases} +1 & \text{if } \hat{f}(\boldsymbol{x}) \geq 0 \\ -1 & \text{if } \hat{f}(\boldsymbol{x}) < 0 \end{cases} \qquad \hat{f} : \Omega \to \mathbb{R} \ , \tag{4.2}$$

where $+1$ and -1 correspond to the positive and negative class labels, respectively. In this we assume that 0 is the appropriate threshold for distinguishing between positive and negative outputs. We call a reconstruction approach like the one in Eq. (4.2) a *soft reconstruction*.

General reconstruction mechanism

In Fig. 4.2 we show the representation of a reconstruction scheme, where all the dichotomies $\hat{f}_q$ receive the same input $\boldsymbol{x}$, but each one of them concentrates on a different task. The role of the reconstruction block is to combine all the answers to produce a single class decision as output.

In order to study reconstruction schemes in more detail, we will decompose the general decision rule $\hat{F}$ itself, so that it can accommodate the transformation of the original problem

[1] We recall that the Hamming distance between two equal-sized vectors of bits is the number of bit positions in which they differ. If the bits are in $\{-1,+1\}$ this definition is unchanged. However, when the bits are in $\{-1,0,+1\}$, the notion of Hamming distance is somewhat different: it counts the number of bit positions in which one vector has value -1 and the other has value $+1$, hence a coefficient of 0 never contributes to the distance.

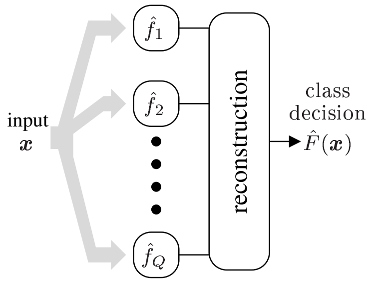


Figure 4.2: Decomposition of a multi-class learning task into dichotomies.

into sub-problems, as follows:

$$\hat{F} = \arg\max_k \circ z \circ \hat{\boldsymbol{f}} . \tag{4.3}$$

In Eq. (4.3), $\hat{\boldsymbol{f}}$ is the vector of dichotomies $\hat{f}_q : \Omega \to \mathbb{R}$, with $q = 1, \ldots, Q$, and $z : \mathbb{R}^Q \to \mathbb{R}^K$ is a linear mapping between the outputs of the dichotomies and the classes, represented by a *reconstruction matrix* $\boldsymbol{M} \in \mathbb{R}^Q \times \mathbb{R}^K$. The corresponding decision rule is then:

$$\hat{F}(\boldsymbol{x}) = \arg\max_k \hat{\boldsymbol{f}}(\boldsymbol{x}) \cdot \boldsymbol{M} = \arg\max_k \sum_q \hat{f}_q(\boldsymbol{x}) \cdot M_{qk} , \tag{4.4}$$

as shown in Fig. 4.3, with a detailed view of the reconstruction block of the diagram of Fig. 4.2 for the case of four classes. The linear mapping z establishes the relation between the di-

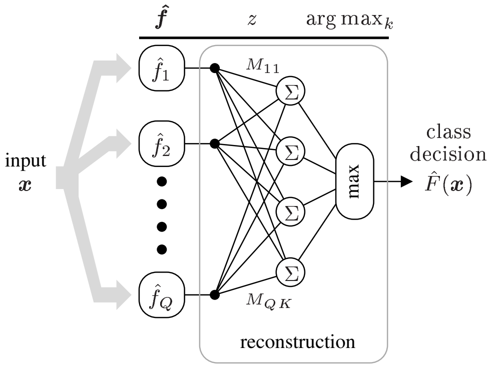


Figure 4.3: Detailed diagram of the decomposition of a multi-class learning task into dichotomies, and the associated reconstruction, for the case $K = 4$.

chotomies and the classes. As presented, the reconstruction matrix $\boldsymbol{M}$ contains values defined in $\mathbb{R}$. However, it is easier to understand its role if we consider them to be restricted to the set $\{-1, 0, +1\}$, in which case the mapping can be interpreted as a voting relation from the dichotomies with respect to the classes. Each value M_{qk} determines whether dichotomy $\hat{f}_q$ votes in favor of or against class c_k (or abstains, in case it is 0). In this setting, the function $\arg\max_k$ selects the class that receives the largest number of votes from the ensemble of dichotomies.

It seems clear that the matrix $\boldsymbol{M}$ defining the linear mapping z should be equal to, or at least based on the decomposition matrix $\boldsymbol{D}$. This is true, although it is possible to give them a certain independence. As will be discussed later in Sect. 4.3.3, the general form of $\boldsymbol{M}$, i.e. having its values defined in $\mathbb{R}$ instead of $\{-1, 0, +1\}$, offers some advantages. For example, it allows to establish more sophisticated (weighted) voting schemes where the decisions of the various dichotomies are given more or less importance according to certain criteria. Presently, and until indication is given in contrary, the two matrices are considered equivalent ($\boldsymbol{M}=\boldsymbol{D}$).

We remark that the proposed reconstruction procedure is valid for both soft and hard reconstructions. However, certain problems may occur when the outputs of the dichotomies are allowed to span an unlimited range in $\mathbb{R}$, even if centered around 0.0. If a dichotomy $\hat{f}$ outputs a very large value (positive or negative) for one of the classes, it may outweigh the decisions of other dichotomies, even if there is no sound reason for giving that much importance to $\hat{f}$ compared to the other dichotomies. Hence, it is better to normalize the outputs of $\hat{\boldsymbol{f}}(\boldsymbol{x})$ to a stable interval, in order to guarantee a correct balance.

We also precise, for reasons of clarity, that the notions of hard and soft reconstruction are related to whether the outputs of the dichotomies are discrete or continuous, and not to the values of the matrix $\boldsymbol{M}$.

A typical decomposition/reconstruction scheme

The form of Eq. (4.3), with an additional sigmoid activation function σ, can be used to describe the output of a conventional Multi-Layer Perceptron[2] (cf. Sect. 1.4.2):

$$\hat{F} = \arg\max_k \circ \; \sigma \; \circ \; z \; \circ \hat{\boldsymbol{f}} \;, \tag{4.5}$$

where the functions $\hat{f}_q$ are computed by the neurons of the last hidden layer, $\boldsymbol{M}$ is the matrix of weights connecting the hidden neurons with the output neurons, and σ is the activation function of the output neurons. This is shown if Fig. 4.4, which can be compared with the scheme of Fig. 4.3. It can be concluded from this description that although common neural

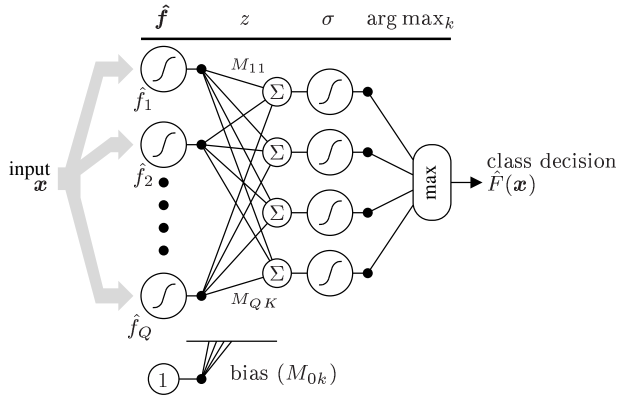


Figure 4.4: The last level of connections in a Multi-Layer Perceptron as an instance of a decomposition scheme.

networks are capable of handling multi-class classification problems, this is usually achieved

[2]Multi-layer Perceptron is the name of the most common topology of Artificial Neural Networks.

through the use of a standard one-per-class decomposition scheme. However, there are examples of alternative approaches, as is the case of Price et al. (1995) who code the outputs of a neural network in a pairwise coupling fashion and successfully apply it to a handwritten character recognition task.

4.1.3 A priori decompositions

With the above descriptions, we now have the basic framework for describing different decomposition schemes. Each scheme can be characterized by its decomposition matrix $\boldsymbol{D}$, which can be an arbitrary matrix with values in $\{-1, 0, +1\}$ fulfilling certain basic conditions, which are described below.

The usual manner of defining decomposition schemes is *a priori*, since the task of each sub-model (dichotomy) is determined before the training process begins. Another possibility that will be discussed later in Sect. 4.3 is to use *a posteriori decompositions* where the relative positioning in the input space of data belonging to different classes is taken into account to define the tasks of the dichotomies. In this manner, these tasks can be made as simple as possible, by choosing classes that are easier to separate. This procedure implies that the decomposition scheme be constructed iteratively, and that the classes chosen as negative and positive for each new dichotomy be determined during the construction phase of the classification model.

Validity of a decomposition scheme

A *valid* decomposition scheme must allow reconstruction, i.e. for any two instances $\boldsymbol{x}, \boldsymbol{x}' \in \Omega$ belonging to different classes, the condition $\hat{F}(\boldsymbol{x}) \neq \hat{F}(\boldsymbol{x}')$ should be satisfied. So, the labels must be assigned to the classes such that for every two columns in $\boldsymbol{D}$, there is at least one row for which the coefficients in the two columns are $+1$ and -1. This means that every pair of classes must be distinguished by at least one dichotomy in the scheme, otherwise there is no information allowing their instances to be distinguished using the Q trained dichotomies.

Minimal and maximal decompositions

If a compact decomposition is sought, the minimal valid scheme contains $Q = \lceil \log_2(K) \rceil$ dichotomies, as shown in Fig. 4.5 (a) for the case of four classes. On the other hand, Fig. 4.5 (b) illustrates a decomposition containing all the possible dichotomies based on four classes and including all of them simultaneously. Note that the dichotomies $\hat{f}$ and $-\hat{f}$ are equivalent for most learning algorithms, and trivial dichotomies with only positive or only negative classes are not interesting. Consequently, the total number of dichotomies that can be formed with K classes including all of them simultaneously is equal to $2^{K-1} - 1$. However, certain dichotomies may contain less than K classes (but at least two, in any case). Considering that each dichotomy is a word of length K written with the alphabet $\{-1, 0, +1\}$, the following expression gives the total number of such words containing at least one symbol -1 and one symbol $+1$, and considering that two words such that one is obtained from the other by exchanging the -1 and the $+1$ symbols are the same:

$$\sum_{i=2}^{K}(2^{i-1}-1)\binom{K}{i} .$$

Using a well-known equality of a sum of powers derived from the Binomial Theorem (Weisstein, 1996):

$$(1+a)^K = \sum_{i=0}^{K}\binom{K}{i}a^i ,$$

combined with the previous expression, and setting $a=2$, it can be verified that a learning task with K classes can be decomposed into a total of $\frac{1}{2}(3^K+1)-2^K$ different dichotomies.

$$\begin{pmatrix} +1 & +1 & -1 & -1 \\ +1 & -1 & +1 & -1 \end{pmatrix}$$

(a) minimal decomposition

$$\begin{pmatrix} +1 & +1 & +1 & -1 \\ +1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & -1 & -1 & +1 \\ +1 & -1 & -1 & -1 \end{pmatrix}$$

(b) maximal decomposition

Figure 4.5: Minimal and maximal decomposition matrices for the case $K=4$.

Robustness of a decomposition scheme

We have seen above that the decomposition matrix $\boldsymbol{D}$ can be chosen arbitrarily in $\{-1,0,+1\}$ as long as it respects the class distance condition (every class is distinguished by at least one dichotomy). Here we mention two important guidelines, proposed by Dietterich and Bakiri (1995), that can be used for the design of decomposition schemes which are not only valid but also robust.

Large column (class) distance. After the decomposition has been designed and the various dichotomies have been trained, it is expected that each dichotomy will provide, for a new example $\boldsymbol{x}$ to be classified, an answer that is coherent with the assigned binary classification task. In ideal conditions, the vector $\hat{\boldsymbol{f}}(\boldsymbol{x}) \in \{-1,+1\}^Q$ of binary answers will be equal to the column in $\boldsymbol{D}$ corresponding to the class of $\boldsymbol{x}$. In practice, dichotomies are classification models that can fail to provide the correct answer in some cases, and so the column vector produced can correspond to the incorrect class column in $\boldsymbol{D}$ or even to no column at all. This suggests the need for some error correcting mechanism, so that when one or a few dichotomies produce an erroneous answer, the correct class can still be identified. For example, in the decomposition scheme of Fig. 4.5 (b) every two columns are at Hamming distance four from each other. Therefore, if for a given example $\boldsymbol{x}$ from class k no more than one dichotomy provides an erroneous answer, the Hamming distance between the output vector created by $\hat{\boldsymbol{f}}(\boldsymbol{x})$ and the k^{th} column of $\boldsymbol{D}$ is at most one, while the distance between that vector and any other column of $\boldsymbol{D}$ is at least three. Consequently, $\hat{F}$ can still output the correct answer.

In general, the error-correcting ability of a decomposition scheme is determined by its class distance, represented by the minimal Hamming distance between the columns of the respective decomposition matrix. If that distance is d_{Hamm}, the number of errors that can be corrected is given by

$$\left\lfloor \frac{d_{\text{Hamm}} - 1}{2} \right\rfloor \; .$$

The class distance of a scheme can be larger with a larger number of dichotomies. In Sect. 4.2.3 below we will describe a decomposition scheme that is able to make intentional use of this property in order to increase the robustness (ECOC).

Large row (task) distance. The robustness of a decomposition scheme is also dependent on the distance between the rows of the matrix $\boldsymbol{D}$. The purpose of having learning tasks which are different is to avoid correlated errors made by different dichotomies. In fact, if several dichotomies are designed such that they have similar learning tasks, which means that there is a large overlap between the classes that are considered positive and negative in each task, they are likely to produce errors in similar situations. This reduces the error-correcting strength of the scheme. The benefit of having every pair of classes separated by several dichotomies, as indicated above, can only be achieved if those dichotomies produce diversified outputs. Having binary learning tasks whose mutual distances are sufficiently distributed can help in spreading the errors produced by each dichotomy over different classification situations, which effectively allows errors to be corrected by the remaining dichotomies. Another possibility of avoiding correlated errors is by employing different learning algorithms to learn similar dichotomies.

A remark about the distance between rows. There is an issue that deserves attention concerning the measure of distance between rows. We have stated before that, from the learning point of view, the dichotomies $\hat{f}$ and $-\hat{f}$ are usually equivalent. This is explained by the fact that the majority of the learning algorithms construct binary classification models that are symmetric, i.e. the model created for a given data set is the same as if the positive and negative class labels are interchanged. In these conditions, two dichotomies are equivalent both when the Hamming distance between their respective rows in $\boldsymbol{D}$ is null, and when it is maximal (equal to K). On the contrary, they are maximally different when that distance is at the mid-point between 0 and K.

4.2 Study of existing decomposition schemes

In this section we make an overview of some existing decomposition schemes. The first two (one-per-class and pairwise coupling) have already been referred to before in Sect. 4.1.1. All the presented schemes are defined a priori, using only the knowledge about the number of classes in the learning problem. Later in Sect. 4.3 we will introduce the notion of a posteriori decomposition scheme and propose a procedure for designing it.

4.2.1 One-per-class (OPC)

The OPC (cf. page 61) is certainly the most used of all decomposition schemes. As mentioned before, it is used in standard Artificial Neural Network topologies and this is just the most prominent case among many other isolated ones. The reasons why OPC is so widely applied are its great simplicity and intuitiveness, and the fact that it demands a single dichotomy per class, which does not require excessive training effort.

On the side of the disadvantages, OPC is a scheme which is not robust. An error in a single dichotomy can immediately affect the global decision and produce a classification error. This aspect is demonstrated empirically in Sect. 4.4.

4.2.2 Pairwise coupling (PWC)

The PWC decomposition scheme (cf. page 61) has been analyzed in a probabilistic framework by Friedman (1996) and by Hastie and Tibshirani (1996). Integrated in the current doctoral project, Moreira and Mayoraz (1998) have also studied the PWC scheme in some detail. One of the aspects studied was the comparison between a hard and several soft reconstruction procedures. The results obtained for five different data sets, with respect to classification accuracy, did not show a clear advantage of any of the tested methods. Below we present another important aspect that has been analyzed.

Partial training

A particular disadvantage of the PWC scheme is the large number of dichotomies that must be trained, at least compared to OPC, although each dichotomy is trained with the data of only two classes. This has both positive and negative effects. On the positive side is the fact that the training of each dichotomy is faster, both because only part of the data is used, and also because the leaning task is simpler: it is easier, in principle, to distinguish only between two classes than when more classes are involved. The negative consequence of training with partial data sets is that each dichotomy has only limited knowledge, since it is taught how to give answers for two classes only. During the reconstruction phase, when a new example $\boldsymbol{x}$ is to be classified, the system has no information of which dichotomies have authority to provide an answer for $\boldsymbol{x}$. This is a problem when we know that from the $\frac{K(K-1)}{2}$ existing dichotomies, only $K-1$ of them have ever been trained with data from the class of $\boldsymbol{x}$.

Corrected pairwise coupling

In the above mentioned study (Moreira and Mayoraz, 1998), we have suggested a solution to this problem, which attempts at adding more information to the scheme in order to allow it to select the answers from the appropriate dichotomies. The approach consists in adding what we call *correcting dichotomies*. For each dichotomy $\hat{f}_q$ distinguishing class c_i from class c_j, another dichotomy $\hat{g}_q$ is trained for distinguishing classes c_i and c_j (as positive) from all the other classes in the problem (as negative). During the reconstruction, the answer of dichotomy $\hat{f}_q$ for an instance $\boldsymbol{x}$ is conditioned by the answer of dichotomy $\hat{g}_q$ for the same instance, which is expected to be positive if $\boldsymbol{x}$ belongs to one of the classes c_i and c_j and negative otherwise. The decision rule of Eq. (4.4) becomes in this case:

$$\hat{F}(\boldsymbol{x}) = \arg\max_k \sum_q (\hat{f}_q(\boldsymbol{x})\, \hat{g}_q(\boldsymbol{x})) \cdot M_{qk} ,$$

Contrary to what has been stated before, it is better in the case of the correcting dichotomies to have the outputs in $\{0, 1\}$ so that they can either cancel or maintain the answer of their corresponding standard dichotomy. Figure 4.6 illustrates the matrix of correcting dichotomies (b) for the PWC scheme in the case of four classes (a).

$$\begin{pmatrix} +1 & -1 & 0 & 0 \\ +1 & 0 & -1 & 0 \\ +1 & 0 & 0 & -1 \\ 0 & +1 & -1 & 0 \\ 0 & +1 & 0 & -1 \\ 0 & 0 & +1 & -1 \end{pmatrix}$$

(a) regular PWC matrix

$$\begin{pmatrix} +1 & +1 & -1 & -1 \\ +1 & -1 & +1 & -1 \\ +1 & -1 & -1 & +1 \\ -1 & +1 & +1 & -1 \\ -1 & +1 & -1 & +1 \\ -1 & -1 & +1 & +1 \end{pmatrix}$$

(b) additional correcting dichotomies

Figure 4.6: Decomposition matrices for the PWC corrected scheme for the case $K = 4$.

This solution adds a noticeable learning overhead to the system, since twice as many dichotomies need to be trained, and also because each additional dichotomy is trained with the whole training set. However, the resulting scheme is considerably robust, as we will demonstrate empirically in Sect. 4.4. It is referred to under the name PWC-CC (which stands for *pairwise coupling with correcting classifiers*).

Correcting dichotomies, a decomposition scheme as such

After proposing the correcting approach for PWC, we have noticed that the set of correcting dichotomies constitute a decomposition scheme by itself, since it satisfies the conditions defined in Sect. 4.1.3 for the validity of a scheme. This can indeed be verified in Fig. 4.6 (b), which shows that every pair of classes (columns) is at Hamming distance four. This scheme requires fewer dichotomies than PWC-CC, although it maintains the most difficult ones, i.e. those that involve all the classes. In the experiments of Sect. 4.4, it is named CC (*correcting classifiers*), by derivation from PWC-CC.

Structured pairwise coupling reconstruction

More recently, an alternative solution to the problem of training the PWC scheme with partial data has been proposed by Platt et al. (2000) using only the original dichotomies of PWC but by reconstructing the answer in a structured way.

The idea basically consists in considering the answers from the vector $\hat{\boldsymbol{f}}(\boldsymbol{x})$ of dichotomies sequentially, and where the choice of an answer depends on the answers given by previously chosen dichotomies. The procedure starts by requesting the answer of a dichotomy $\hat{f}_q$ separating the classes c_i and c_j. If the output of $\hat{f}_q$ gives preference to class c_i, then the class c_j is discarded from further consideration, otherwise it is c_i that is discarded. Dichotomies are used sequentially in this manner, each allowing one class to be discarded, and the only class that remains at the end is the one that is chosen. Note that when a class is discarded, none of the remaining dichotomies concerned with that class are ever questioned again. The authors state that, according to empirical tests, the order in which the dichotomies are chosen does not significantly change the results.

The decision process used here can be represented by a directed acyclic graph (DAG) like the one shown in Fig. 4.7 for the case of four classes (reproduced from the authors). It has a decision tree-like structure where the root node represents the dichotomy whose answer is requested first. Then, subsequent dichotomies are questioned in turn, and the chain of responses determines a path that leads to one of the leafs with a final class decision.

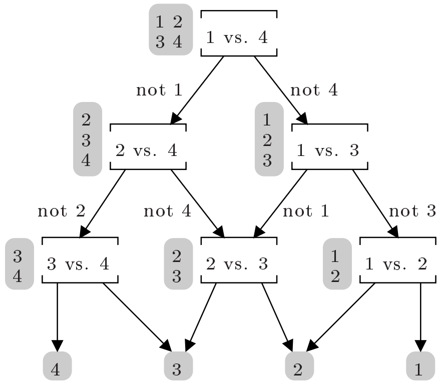


Figure 4.7: Directed acyclic graph (DAG) used as a reconstruction scheme for pairwise coupling. Each node corresponds to a dichotomy whose decision eliminates one of the classes. Associated to each node (in gray) is the list of classes that are still active, and each of the bottom nodes contains a single class each, which corresponds to the final decision. The reconstruction is made by following a path from the root node to one of the leaves.

Compared to the solution that we have proposed above (PWC-CC), based on correcting dichotomies, the current procedure has the advantage of avoiding additional training effort, which results in a final classification model equivalent to the one created for PWC. However, we sustain that it does not guarantee a robust scheme. Indeed, when an instance $\boldsymbol{x}$ from class c_i is being classified, if any of the dichotomies that distinguishes the class c_i from another class gives an incorrect answer, then c_i is immediately discarded, creating a classification error. So, there is the possibility that a single incorrect answer from one of the dichotomies causes an error on the global scheme. In order to verify the validity of our assertion concerning the robustness of this method we have included it in the experimental setup described in Sect. 4.4 with the name PWC-DAG.

4.2.3 Error Correcting Output Codes (ECOC)

The guidelines provided in Sect. 4.1.3 for constructing a robust decomposition scheme have been enunciated by Dietterich and Bakiri (1991, 1995). These authors have established a relation between the reconstruction procedure that is associated with a decomposition scheme and the theory of error-correcting codes (Peterson and Weldon, 1972), and applied the described principle of distributed codes in order to maximize the error correcting capability of the scheme. Their approach to the decomposition problem takes a number K of classes and a number Q of dichotomies as input and creates a decomposition matrix of size $Q \times K$ maximizing both the column and the row distance. This decomposition scheme is known by the name of ECOC (*error-correcting output codes*).

Dietterich and Bakiri (1995) use decomposition matrices with values in $\{0, 1\}$ and assume that the outputs of the dichotomies are in the interval $[0, 1]$. The reconstruction process selects the class whose column in the matrix is at minimal distance, in the L_1 norm, from the vector of outputs. Consider $\boldsymbol{D}' = \frac{\boldsymbol{D}+1}{2}$ and $\hat{\boldsymbol{f}}'(\boldsymbol{x}) = \frac{\hat{\boldsymbol{f}}(\boldsymbol{x})+1}{2}$ as the decomposition matrix and the dichotomy vector, respectively, used in ECOC, which are rescaled versions of those considered here, defined in $\{-1, +1\}$. The expression that selects the best class based on the L_1 norm is

$$\hat{F}(\boldsymbol{x}) = \arg\min_k \sum_q \left| \hat{f}'_q(\boldsymbol{x}) - D'_{qk} \right| \ . \tag{4.6}$$

In fact, this expression is equivalent to our standard decision rule of Eq. (4.4)

$$\hat{F}(\boldsymbol{x}) = \arg\max_k \sum_q \hat{f}_q(\boldsymbol{x}) D_{qk} \ ,$$

when the rescaling mentioned above is done and the reconstruction matrix is equal to the decomposition matrix ($\boldsymbol{M} = \boldsymbol{D}$). For any given class c_k, when the component $\hat{f}_q(\boldsymbol{x})$ of the dichotomy vector matches the component D_{qk} of the matrix, the expression $\hat{f}_q(\boldsymbol{x}) \cdot D_{qk}$ attains its maximal value, provided that both components are defined in $\{-1, +1\}$. In the same situation, the expression $|\hat{f}'_q(\boldsymbol{x}) - D'_{qk}|$ attains its minimal value. On the contrary, these two expressions attain their minimal and maximal value, respectively, when the components $\hat{f}_q(\boldsymbol{x})$ and D_{qk} do not match. Hence, the Equations (4.6) and (4.4) both return the same class index k for any given instance $\boldsymbol{x}$.

Concerning the definition of the actual decomposition scheme, the authors propose to use the maximal decomposition containing $2^{K-1} - 1$ dichotomies for problems with up to seven classes, otherwise the attainable column and row distances are not very high. For $K > 7$ the number Q of dichotomies is a user-defined parameter and a heuristic search technique is used to construct a matrix that maximizes the distances. One of the proposed heuristics, which we have implemented for the experiments presented here, starts by creating a random Boolean matrix of size $Q \times K$. The Hamming distance between pairs of columns of this matrix follows a binomial distribution with mean $\frac{Q}{2}$, which is already quite good. In order to augment the minimal distance, a *randomized hill-climbing* procedure is applied, combining local search with random search. It works as follows: at each iteration, the pair of least separated rows and the pair of least separated columns are chosen such that the four bit components in which they all intersect does not form an XOR-like configuration[3]. Next, the bit values of these

[3] The possible XOR configurations are $\begin{smallmatrix} 0\,1 \\ 1\,0 \end{smallmatrix}$ and $\begin{smallmatrix} 1\,0 \\ 0\,1 \end{smallmatrix}$. These two configurations provide the maximal distance

components are exchanged appropriately (in order to form an XOR), and the minimal distances of the matrix are recalculated. This sequence of operations is repeated iteratively until a local maximum is attained. After this stage, the method selects randomly the pairs of rows and the pairs of columns where the distance can be raised. In the mentioned article from 1995, the authors describe two other heuristic approaches for improving the decomposition matrix. Nevertheless, this optimization phase is not critical for the performance of the scheme. For example, James and Hastie (1997) suggested that the ECOC scheme is optimal, in terms of classification accuracy, when the decomposition matrix is randomly defined.

Note that the used measure of row distance is the one that takes the complement into account, as described above in Sect. 4.1.3, where two rows are maximally different when their Hamming distance is at the midpoint between 0 and K.

4.3 A posteriori decomposition — pertinent dichotomies

In the current section we describe how to design a decomposition scheme a posteriori, that includes only dichotomies that are *pertinent*. We explain what "pertinent" means and point out the advantage of such an approach comparing to the a priori schemes. The research material presented in this section derives from Mayoraz and Moreira (1997).

There is a problem associated with some of the decomposition schemes described in Sect. 4.2 above, which is the fact that each dichotomy is defined a priori without taking into account the distribution of the classes in the input space. For schemes like OPC and especially for standard PWC this is not really a problem, since there is at least one side of the decision boundary with only a single class. On the contrary, in schemes like ECOC and CC, the learning algorithm can sometimes be "forced" to learn certain particularly difficult classification tasks, caused by an unnatural grouping of classes. We illustrate this with a simple example in Fig. 4.8. It shows the ECOC decomposition scheme applied to a problem with four classes. On the

	A	B	C	D
$\hat{f}_1$	$+1$	$+1$	$+1$	-1
$\hat{f}_2$	$+1$	$+1$	-1	$+1$
$\hat{f}_3$	$+1$	$+1$	-1	-1
$\hat{f}_4$	$+1$	-1	$+1$	$+1$
$\hat{f}_5$	$+1$	-1	$+1$	-1
$\hat{f}_6$	$+1$	-1	-1	$+1$
$\hat{f}_7$	$+1$	-1	-1	-1

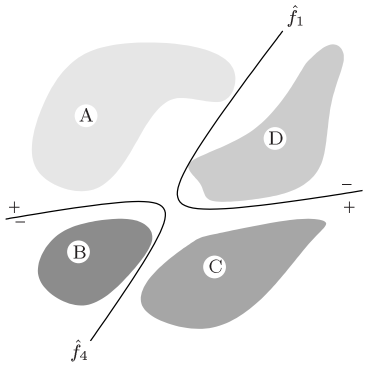


Figure 4.8: Example of the application of the ECOC decomposition scheme to a classification problem with four classes. The class distribution is represented by the regions shaded with different gray levels in the two-dimensional input space on the right-hand side.

right-hand side of the figure, the class distribution is represented by the regions shaded with

between the involved rows and columns when only these four bit components are taken into account.

different gray levels in a two-dimensional input space. On the left-hand side we show the ECOC decomposition matrix. Let us assume that the available learning algorithm is able to generate classification models based on either linear or quadratic discriminant functions. In this case, we can see that the dichotomies $\hat{f}_1$ and $\hat{f}_4$ (represented graphically in the figure) cannot be learned correctly using simple linear classifiers. In the case of $\hat{f}_5$ the situation is more complex (not shown graphically), since even a quadratic discriminant function can hardly learn the task of separating classes A and C from B and D.

The simplicity of the above example does not prevent it from being illustrative. While it is true that more capable learning algorithms could be used in this case, it is also true that most real-life learning tasks and their corresponding class distribution in the input space are much more complex than shown above. This problem led us to develop an alternative approach that defines the dichotomies according to the particular classification problem treated.

We consider that a dichotomy is pertinent when it is defined to adapt as closely as possible to the data of the different classes. The main idea consists in assigning positive and negative labels to the different classes in order to create a binary learning task that is as easy as possible. The notion of pertinence depends on the type of model and on the positioning of the classes in the input space. Given a set of classes, consider a dichotomy $\hat{f}$ that assigns them a particular positive/negative labeling and another dichotomy $\hat{f}'$ that assigns them a different labeling (each labeling contains at least one positive and one negative class). We say that $\hat{f}$ is more pertinent than $\hat{f}'$ if they are implemented by classifiers of equivalent capacity and the generalization error of $\hat{f}$ is lower than that of $\hat{f}'$. Likewise, $\hat{f}$ is more pertinent than $\hat{f}'$ if, to achieve the same level of generalization error, $\hat{f}$ can be implemented with a classifier of lower capacity than $\hat{f}'$.

In Fig. 4.9, we show an example of a decomposition scheme defined a priori for the example of Fig. 4.8 on the preceding page, in which the used dichotomies are more pertinent than those of the earlier example. We remark that they can all be implemented by simple linear discriminants.

	A	B	C	D
$\hat{f}_1$	+1	+1	−1	−1
$\hat{f}_2$	+1	−1	−1	−1
$\hat{f}_3$	+1	+1	−1	+1
$\hat{f}_4$	+1	−1	−1	+1

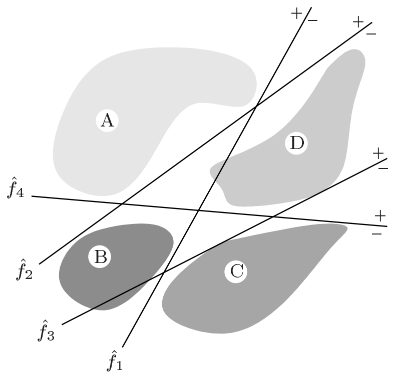


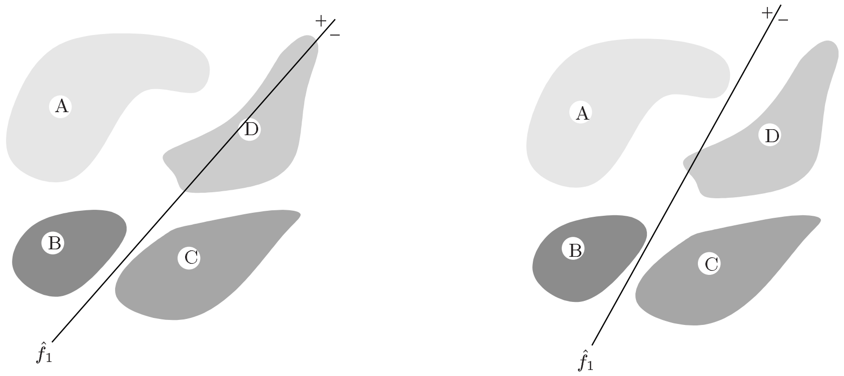
Figure 4.9: The classification problem of Fig. 4.8 solved using pertinent dichotomies.

The dichotomies of Fig. 4.9 are in fact a subset of those defined by ECOC in the example of Fig. 4.8. This is a logical consequence of the fact that ECOC builds schemes containing all the possible dichotomies, for problems with up to seven classes. For $K > 7$, the intersection between the sets of dichotomies in the two schemes is likely to be reduced, since different criteria are used to select them.

4.3.1 Generating a pertinent dichotomy

In the examples of the previous section, the dichotomies were implemented by linear or quadratic discriminant functions. In practice, each learning algorithm has its own notion of "easy" learning task, according to the representational language used. Hence, the same dichotomy can be more or less pertinent depending on the learning algorithm that is used for training it.

A pertinent dichotomy is created using a pair of classes (c_i, c_j) as *germ*. A first transitory dichotomy $\hat{f}_q$ is generated for separating c_i as being positive from c_j as being negative. Next, the training data of each one of the remaining classes is tested on $\hat{f}_q$. For a class c_k $(c_k \neq c_i, c_j)$, if the majority of its training instances is classified as positive by $\hat{f}_q$, then c_k is considered positive for this dichotomy, otherwise it is considered negative. After making this decision for all the remaining classes, the transitory dichotomy is discarded and a definitive one is trained using all the data. This process is illustrated in Fig. 4.10 for the creation of the dichotomy $\hat{f}_1$ of Fig. 4.9. In this case the germ classes are c_i = B and c_j = C, and it can be observed that



	A	B	C	D
$\hat{f}_1$	0	+1	−1	0

	A	B	C	D
$\hat{f}_1$	+1	+1	−1	−1

Figure 4.10: Construction of a dichotomy. On the left, the transitory dichotomy $\hat{f}_1$ is trained only on the germ classes B and C. After deciding the status of the remaining classes relatively to $\hat{f}_1$, the dichotomy is retrained using the whole data (on the right).

class A is considered positive by the transitory dichotomy, while class D is in its majority on the negative side, and so it is considered negative. At the end, the classes A and B are positive, while C and D are negative.

4.3.2 The PertinentDichotomies algorithm

Now that we have determined how each individual dichotomy is created, we will explain the general procedure for creating the whole decomposition scheme. As for any other scheme, we have to guarantee that there is at least one dichotomy separating each pair of classes. On the other hand, we must also favor the differentiability between different tasks (dichotomies). The algorithm **PertinentDichotomies** that we propose is iterative (Alg. 4.1). At each iteration, a different germ (pair of classes) is used for generating a dichotomy. The germ is selected among the pairs of classes that are separated by the minimal distance according to the dichotomies generated so far. (In the beginning all the pairs of classes are at null distance.) After the task for a dichotomy is defined, i.e. the positive and negative labels are assigned to all classes, we check whether it is different from the already existing dichotomies. We recall the remark made on page 67 about the distance between rows, where we explained that two dichotomies are non-equivalent only if the Hamming distance between their respective rows in $\boldsymbol{D}$ is both greater than zero and less than the number of classes. If the answer of the test is positive, the definitive version of the dichotomy is trained and added to the scheme, otherwise it is rejected.

Algorithm 4.1 PertinentDichotomies($\mathcal{X}$ **)**

Inputs: $\mathcal{X}$: set of training examples

Parameters: $d_{\min}$: minimal class distance

$\boldsymbol{D} \leftarrow$ empty matrix *[decomposition matrix]*
$\mathcal{G} \leftarrow$ {set of all pairs (i,j) s.t. $i,j \leq K,\ i < j$} *[set of available germs (pairs of classes)]*
$Q \leftarrow 0$ *[number of dichotomies]*
while $\mathcal{G} \neq \varnothing$ **do**
  [choose the pair of least separated classes among the available germs]
  $i,j \leftarrow \arg\min_{i,j} d_{\text{Hamm}}(\boldsymbol{D}_{\cdot i}, \boldsymbol{D}_{\cdot j}),\quad$ s.t. $\{(i,j)\} \in \mathcal{G}$
  $\mathcal{G} \leftarrow \mathcal{G} \setminus \{(i,j)\}$ *[remove the used germ]*
  $\mathcal{X}^+ \leftarrow \mathcal{X}_i$ *[$\mathcal{X}_i \subset \mathcal{X}$ is the subset of training examples from class c_i]*
  $\mathcal{X}^- \leftarrow \mathcal{X}_j$ *[$\mathcal{X}_j \subset \mathcal{X}$ is the subset of training examples from class c_j]*
  train a transitory dichotomy $\hat{f}$ to distinguish $\mathcal{X}^+$ from $\mathcal{X}^-$
  $\boldsymbol{d} \leftarrow \{0\}^K, \quad \boldsymbol{d}_i \leftarrow +1, \quad \boldsymbol{d}_j \leftarrow -1$ *[the code vector of a new dichotomy]*
  for all $k \in \{1,\ldots,K\} \setminus \{i,j\}$ **do**
    $\underline{\omega}_k \leftarrow$ proportion of training examples from class c_k classified as positive by $\hat{f}$
    if $\underline{\omega}_k \geq 0.5$ **then** *[decide whether $\hat{f}$ considers class c_k as positive or as negative]*
      $\boldsymbol{d}_k \leftarrow +1, \quad \mathcal{X}^+ \leftarrow \mathcal{X}^+ \cup \mathcal{X}_k$
    else
      $\boldsymbol{d}_k \leftarrow -1, \quad \mathcal{X}^- \leftarrow \mathcal{X}^- \cup \mathcal{X}_k$
    end if
  end for
  [make sure that the created dichotomy is different from the existing ones]
  if $0 < d_{\text{Hamm}}(\boldsymbol{d}, \boldsymbol{D}_{q\cdot}) < K,\ \ \forall q \in \{1,\ldots,Q\}$ **then**
    $Q \leftarrow Q + 1$
    train a new definitive dichotomy $\hat{f}_Q$ to distinguish $\mathcal{X}^+$ from $\mathcal{X}^-$
    add $\boldsymbol{d}$ as a new row to $\boldsymbol{D}$
  end if
  [exclude all the germs whose classes are already at distance d_{min}]
  $\mathcal{G} \leftarrow \mathcal{G} \setminus$ {set of all pairs (i,j) s.t. $d_{\text{Hamm}}(\boldsymbol{D}_{\cdot i}, \boldsymbol{D}_{\cdot j}) \geq d_{\min}$}
end while
return($\hat{\boldsymbol{f}}$, $\boldsymbol{D}$) *[the vector of dichotomies and the decomposition matrix]*

The algorithm **PertinentDichotomies**[4] takes a parameter as input ($d_{\min}$), strictly positive, which is is the desired minimal distance between pairs of classes. This parameter is used for the stopping criterion. At each iteration, the eligible germs are those that fulfill the following two conditions: (i) the distance between the germ classes c_i and c_j is below $d_{\min}$, and (ii) the germ has not yet been used. If no such germ exists, the algorithm stops. In the empirical tests, this algorithm is denoted PD .

4.3.3 Possible extensions

Below we propose two possible extensions for the standard version of **PertinentDichotomies**.

Limiting the complexity of dichotomies

The procedure presented in Alg. 4.1 creates dichotomies that include all the classes in the problem. The algorithm can be modified in order to limit the acceptability of classes in each dichotomy. This means that a class c_k is accepted to participate in a dichotomy $\hat{f}$ only if the proportion $\underline{\omega}_k$ of training examples in $\mathcal{X}_k$ classified as positive by $\hat{f}$ is above a threshold ω_{thr} in the range $]0.5, 1.0]$ (in which case c_k becomes positive) or below $1-\omega_{\text{thr}}$ (c_k becomes negative), otherwise the class is ignored by $\hat{f}$. This leads certain components of the matrix $\boldsymbol{D}$ to have value 0, and increases the number of dichotomies in the scheme.

We have implemented a version of the algorithm using different values for the threshold ω_{thr} and tested it on twelve data sets with more than two classes. The tested values were 0.5 (the standard applied in Alg. 4.1, used as baseline), together with 0.6, 0.8, and 0.9. We remark that a value of 1.0 for ω_{thr} leads the procedure **PertinentDichotomies** to generate a scheme which is equivalent to standard PWC, since every dichotomy will contain only the two classes of the germ, and every germ will be used once.

The tests have been done using the standard 5×2cv procedure described in Appendix B for splitting the data into training and testing sets. The learning algorithm used to implement the dichotomies was the decision tree learner C4.5, as in the previous chapter. The complexity of each classification model is represented by the number of nodes in the respective decision tree. The data sets that have been used are all those described in Appendix A that contain more than two classes.

From the results of the experiments we have observed that the classification accuracy does not change significantly using different values of ω_{thr}, and so we decided not to present the corresponding values. On the contrary, several modifications were detected concerning the generated number of dichotomies, as well as the complexity of the classification models. The plots of Figures 4.11 to 4.13 show these modifications. The number of dichotomies effectively increases with increasing value of the threshold (Fig. 4.11), which is a consequence of the smaller number of classes handled by each dichotomy. Another consequence of this fact is the decreasing complexity of the generated dichotomies, as shown clearly in Fig. 4.12. In contrast, the overall

[4] We use the subscript "$\cdot$" associated with a matrix to denote the set of all rows or the set of all columns in that matrix. For example, the expression $\boldsymbol{D}_{\cdot k}$ represents the column k of $\boldsymbol{D}$, while $\boldsymbol{D}_{q\cdot}$ represents the row q of $\boldsymbol{D}$.

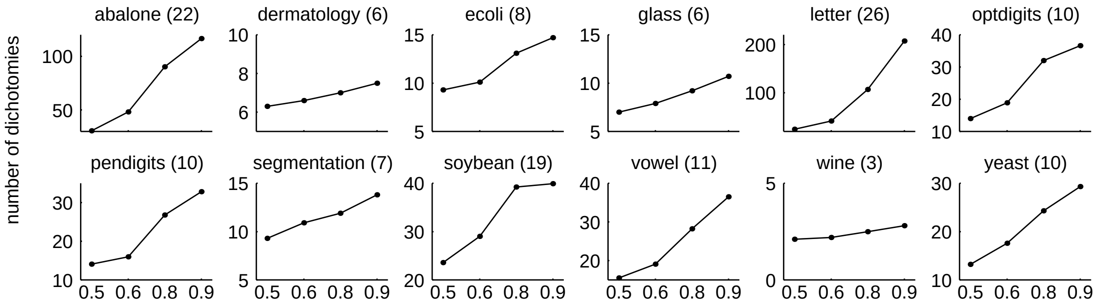


Figure 4.11: Number of dichotomies generated by the PD decomposition scheme with different values of ω_{thr}, shown in the x-axis. The plotted values correspond to the mean over 5×2cv experiments. For each data set, the number of classes is shown in parenthesis.

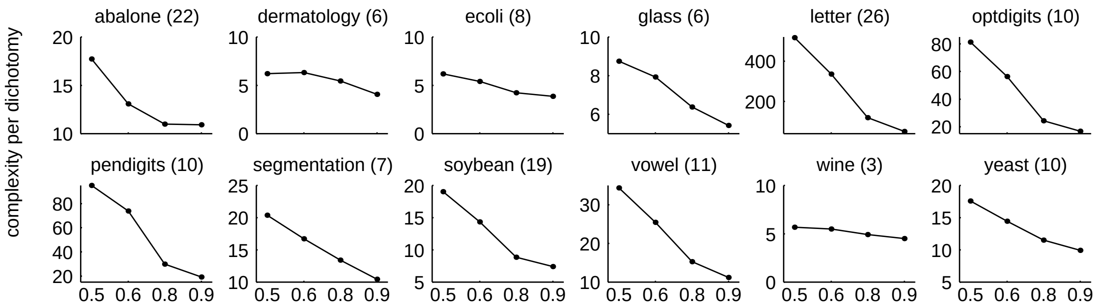


Figure 4.12: Average complexity of each dichotomy generated by the PD decomposition scheme with different values of ω_{thr}, shown in the x-axis. The plotted values correspond to the mean over 5×2cv experiments. For each data set, the number of classes is shown in parenthesis.

model complexity (Fig. 4.13), which includes the complexity of all the dichotomies trained for each scheme, behaves differently depending on the data set. This result derives directly from the previous two results, given that the overall complexity in a scheme is equal to the mean complexity per dichotomy multiplied by the number of dichotomies. As such, its evolution depends on the relative contributions of each of the other two.

Normalization of the reconstruction matrix. An important remark is necessary here. As mentioned in Sect. 4.1.2, the decomposition matrix $\boldsymbol{D}$ is a reasonable choice for the reconstruction matrix ($\boldsymbol{M} = \boldsymbol{D}$). However, if $\boldsymbol{D}$ contains components with value 0, it is possible that the different dichotomies contain variable numbers of classes. This has the consequence that the different classes will be included in variable numbers of dichotomies, and hence the number of zeros in the different columns of the matrix will also be variable. Therefore, the choice $\boldsymbol{M} = \boldsymbol{D}$ produces a bias towards the classes that participate in more dichotomies, since the scalar product $\hat{\boldsymbol{f}} \cdot D_{\cdot k}$ will tend to be larger for columns $D_{\cdot k}$ with fewer zeros. To account for this, the following normalized version of $\boldsymbol{M}$ is preferred:

$$M_{qk} = \frac{D_{qk}}{\sum_{q'} |D_{q'k}|} \; .$$

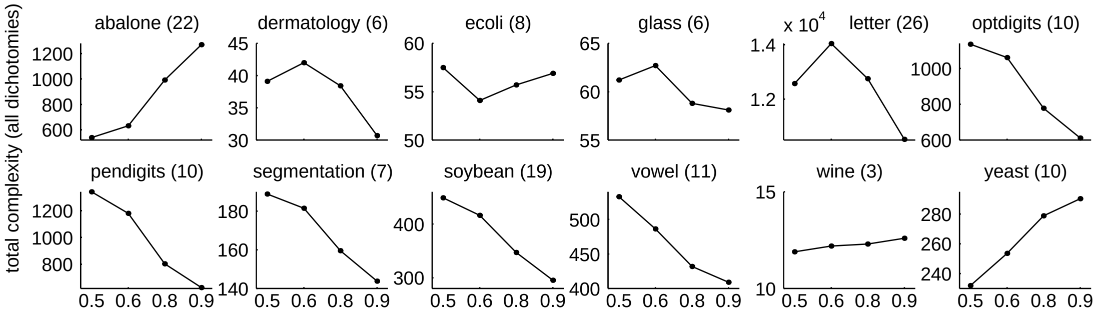


Figure 4.13: Total complexity of the dichotomies generated by the PD decomposition scheme with different values of ω_{thr}, shown in the x-axis (including all dichotomies). The plotted values correspond to the mean over 5×2cv experiments. For each data set, the number of classes is shown in parenthesis.

Avoiding repeated dichotomies

There is a limitation associated with the algorithm **PertinentDichotomies**, which is the fact that the pairs of classes which are more difficult to distinguish will not attain, in some cases, the desired distance $d_{\min}$. The algorithm guarantees that each pair of classes is distinguished by at least one dichotomy. For example, a pair of classes that is difficult to separate, and that is always put on the same side of every generated dichotomy, will be placed on opposite sides of at least the dichotomy where they are used as germ. However, each pair of classes can be used as germ only once. There is no interest in using the same germ several times, as it will always generate the same dichotomy, and this is the reason why the distance between pairs of classes of this type can possibly not be raised up to $d_{\min}$.

Perhaps one solution for this problem would be to retrain each dichotomy after deciding the status of every new added class, instead of retraining it only once after all the classes have been tested. In this case, classes that would normally fall in one side of the initial dichotomy might fall in the opposite side if the dichotomy has already been modified in the meantime. This procedure might allow different dichotomies to be created from the same germ if the order in which new classes are added is modified, although this cannot be guaranteed. And of course, such a procedure brings additional computational overhead. This subject is left open to further research.

Another possibility for solving the problem of repeated dichotomies is to employ different learning algorithms with the same germ. On the one hand, there is a higher probability of obtaining a dichotomy which is different from the previously obtained ones from the same germ. On the other hand, even if the same dichotomy is generated, it is based on a different model, using a representational language which is different from that of the previous models, which prevents them to produce correlated errors, in principle.

4.4 Experimental comparison

This section presents and discusses the results of an empirical comparison between all the decomposition schemes described in the previous sections. First, a short review of the tested methods is provided in Sect. 4.4.1, followed by the actual results in Sect. 4.4.2. The results evaluate four different parameters: (i) the classification accuracy, (ii) the execution time, (iii) the average model complexity of each dichotomy in the scheme, and (iv) the model complexity of the whole scheme, including all the dichotomies. In addition, we also evaluate the number of dichotomies generated by each scheme. All the studied schemes make use of the same general reconstruction procedure defined in Eq. (4.4) on page 63, except PWC-DAG, which is in fact an alternative reconstruction procedure for PWC.

These experiments have been made in the same conditions as those reported earlier in Sect. 4.3.3: using the same learning algorithm to implement the dichotomies and the same data sets. Concerning the learning algorithm (C4.5), it has the advantage of being capable of handling multi-class learning tasks directly, which allows it to be compared with the decomposition schemes. In addition, it is able to automatically adapt the capacity of the generated model to the difficulty of each particular learning task, through the use of pruning techniques (cf. Sect. 3.4), which is an important feature for the current experiments. Again, the number of nodes in each decision tree gives a measure of the complexity of the respective classification model.

4.4.1 Tested methods

From all the decomposition schemes described in the following paragraphs, PWC-CC, CC, and PD are those that have been proposed in the current project.

OPC One-per-class. It trains K dichotomies, each separating one of the K classes from all the remaining classes.

PWC Pairwise-coupling. It contains $\frac{K(K-1)}{2}$ dichotomies, each one distinguishing between a pair of classes and ignoring the others.

PWC-CC Pairwise-coupling with correcting classifiers. It trains the standard PWC dichotomies plus an additional set of correcting dichotomies, each trained with the whole data. The resulting number of dichotomies is $K(K-1)$. This is the approach that we proposed in order to solve the partial training problem that affects the standard PWC scheme.

CC Correcting classifiers. It trains only the correcting dichotomies of the PWC-CC scheme, which constitute a decomposition scheme by itself. It contains a total of $\frac{K(K-1)}{2}$ dichotomies.

PWC-DAG Pairwise-coupling using directed acyclic graphs for the reconstruction. It is an alternative way of solving the partial training problem of PWC, proposed by Platt et al. (2000).

ECOC Error-correcting output codes. This is the scheme proposed by Dietterich and Bakiri (1991, 1995) with optimized robustness. The number of dichotomies is equal to $2^{K-1}-1$ for problems with up to seven classes. For the others, that number is a user-defined parameter. In the tests, we have set it equal to $\frac{K(K-1)}{2}$, the same number as in PWC.

PD Pertinent dichotomies. This is the standard version of the scheme presented in Alg. 4.1, without using the parameter ω_{thr} proposed in Sect. 4.3.3. In this scheme, the number of dichotomies is determined by the algorithm according to the particular problem treated. The parameter $d_{\min}$ has been set to 4.

ECOC 2 The ECOC scheme, but where the number of dichotomies is the same as in PD. This is intended at comparing PD and ECOC in similar conditions. This scheme is equivalent to ECOC above for problems with up to seven classes.

Plain C4.5 This is not a decomposition scheme. C4.5 is the learning algorithm that is used for training each one of the dichotomies in all the tested decomposition schemes. This algorithm can handle multi-class problems directly, and we have applied it as such in order to compare the direct approach with the decomposed ones.

4.4.2 Results

Classification accuracy

The classification accuracy achieved by each decomposition scheme is an important parameter for measuring its quality. The first important observation provided by the plots of Fig. 4.14 on the facing page is that the accuracy of the base learner can be increased through the use of decomposition schemes; in seven out of twelve data sets there is at least one decomposition that performs significantly better than plain C4.5. Among all the methods shown, ECOC is clearly the one capable of achieving the highest levels of accuracy, while OPC is at the other end of the spectrum with the poorest performance. The rest of the methods have variable performance that depends on the problem.

To make the connection with what has been presented in the previous sections, we note that the standard PWC scheme is effectively improved by the proposed correcting scheme (PWC-CC) in all problems except `abalone`, and that the derived correcting scheme alone (CC) is even more competitive in half of the problems. Furthermore we confirm the poor performance of PWC-DAG which is indeed worse than the standard PWC that it is supposed to improve.

Another conclusion is that our proposed procedure based on pertinent dichotomies (PD) has a performance comparable to that of ECOC using the same number of dichotomies (ECOC 2). We recall that for all the data sets with seven classes or less (`dermatology`, `glass`, `segmentation`, and `wine`) ECOC uses the maximal scheme, and so it is likely to contain more dichotomies than PD, and thus to be more robust.

Despite its importance, the classification accuracy is only one of the perspectives by which

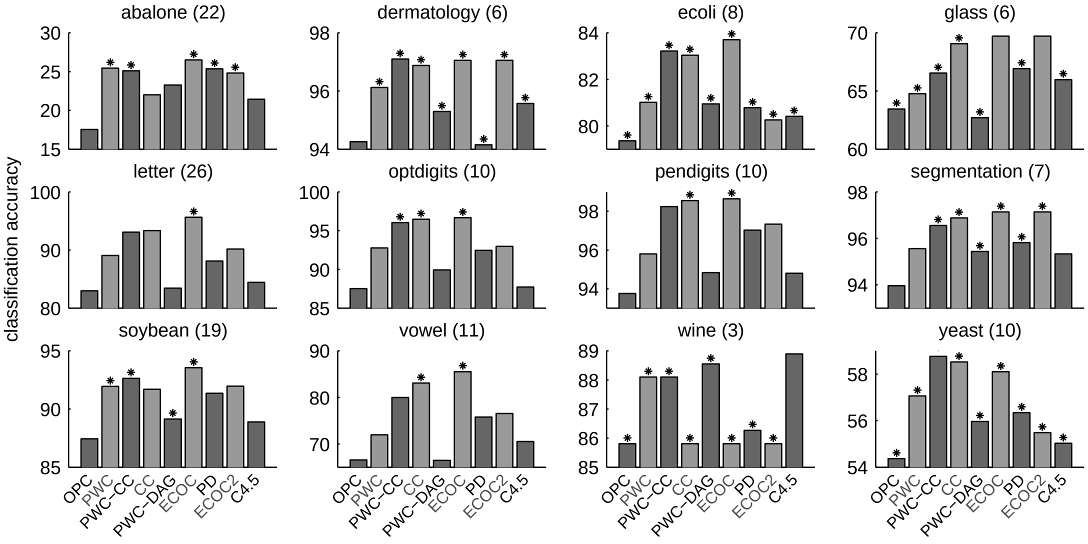


Figure 4.14: Classification accuracy obtained by different decomposition schemes. Higher bars are better. Table 4.2 on page 86 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value (or group of values), with statistical significance at a confidence level of 0.95. For each data set, the number of classes is shown in parenthesis.

the methods can be compared. With a performance comparable to that of other methods, the effective advantages of PD will be shown below.

Execution time

The measured execution time takes into account the duration of the learning process for the ensemble of dichotomies in each decomposition scheme. This parameter is indicative of the learning effort spent by each scheme and is closely related to the total model complexity, as can be observed by comparing Figures 4.15 and 4.17.

From Fig. 4.15 we can observe that the methods with the highest accuracy also need longer training times. The ECOC scheme, in particular, stands above all the others even more markedly than in the previously discussed results, although negatively in this case. The schemes PWC-CC and CC are also considerably time-consuming. On the contrary, plain C4.5 is the clear winner in all cases.

Average complexity

The complexity of each dichotomy in a scheme is also an important factor to be considered. One of the advantages of decomposing a classification problem into sub-problems is the fact that the sub-problems are easier to learn. This opens the possibility of employing learning algorithms with limited capacity to solve complex learning problems. In this perspective, PWC appears as the favorite method (PWC-DAG is equivalent to PWC for that matter, since it only

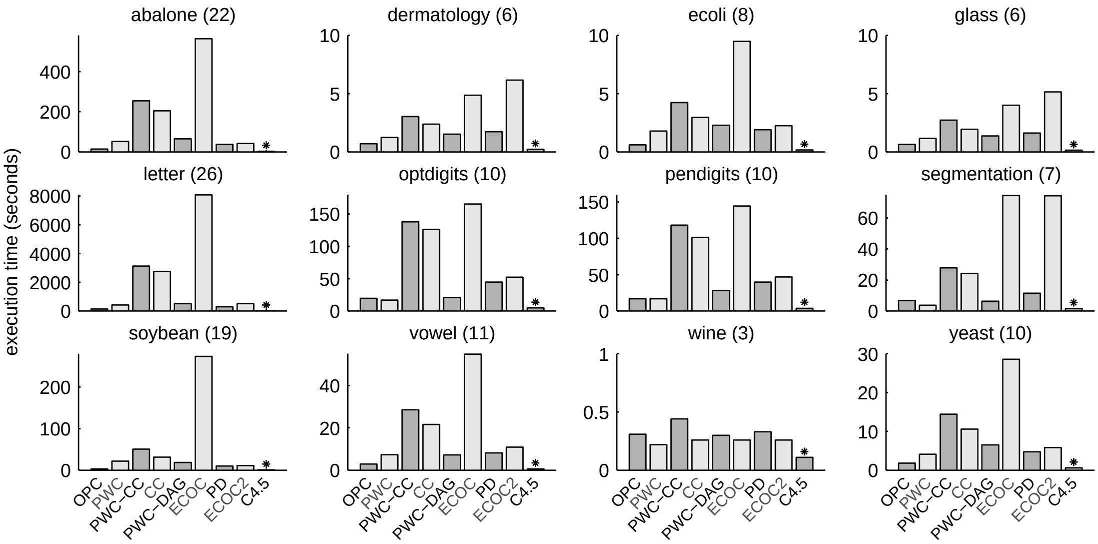


Figure 4.15: Execution time of different decomposition schemes. Lower bars are better. Table 4.2 on page 86 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value (or group of values), with statistical significance at a confidence level of 0.95. For each data set, the number of classes is shown in parenthesis.

changes the reconstruction phase), which is a natural consequence of its learning tasks involving two classes only. In the same spirit, the scheme based on pertinent dichotomies (PD) proves effective in generating dichotomies of low complexity, even considering that all the classes are involved in each dichotomy. This provides a clear indication that the intent of the scheme is effectively achieved.

With the average complexity of its dichotomies, ECOC shows one of its disadvantages, and CC also, although to a smaller extent. Plain C4.5 appears in the results with the largest average complexity, but this is due to the use of a single classification model for solving the entire problem.

Total complexity

Along with the execution time, the total model complexity provides a measure of the learning effort required by each scheme. From Fig. 4.17 we confirm the PD scheme to be among the least demanding decompositions in all problems except `optdigits`, `pendigits`, and `letter`, where it stands behind OPC and PWC, but is still considerably better that the more complex schemes like ECOC and CC. We also observe that plain C4.5 is simpler, in general, than any of the decompositions.

Number of dichotomies

The number of dichotomies generated by each decomposition scheme is shown in Table 4.1 on the next page. The column labeled PWC is valid also for PWC-DAG, CC, and ECOC as

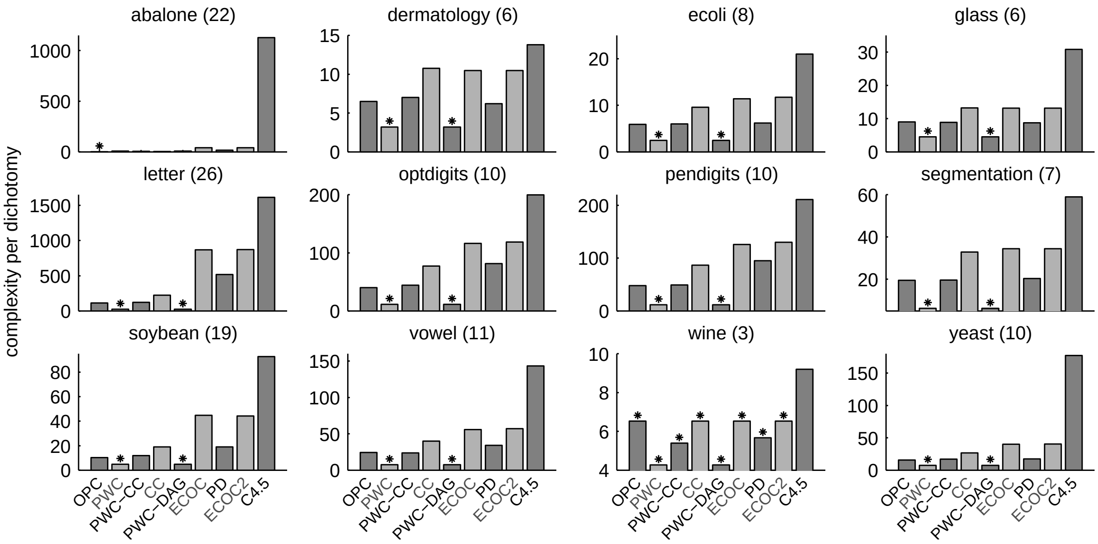


Figure 4.16: Mean complexity of the dichotomies generated by different decomposition schemes. Lower bars are better. Table 4.2 on page 86 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value (or group of values), with statistical significance at a confidence level of 0.95. For each data set, the number of classes is shown in parenthesis.

it has been used here. It is interesting to note that PD always generates a small number of

	NUMBER OF DICHOTOMIES			
DATA SET	OPC	PWC	PWC-CC	PD
abalone	22	231	462	30.7 $\pm$1.7
dermatology	6	15	30	6.3 $\pm$0.5
ecoli	8	28	56	9.3 $\pm$0.9
glass	6	15	30	7.0 $\pm$1.0
letter	26	325	650	24.4 $\pm$1.6
optdigits	10	45	90	14.0 $\pm$1.0
pendigits	10	45	90	14.1 $\pm$1.3
segmentation	7	21	42	9.3 $\pm$0.6
soybean	19	171	342	23.6 $\pm$1.7
vowel	11	55	110	15.5 $\pm$0.9
wine	3	3	6	2.1 $\pm$0.3
yeast	10	45	90	13.2 $\pm$1.1

Table 4.1: Number of dichotomies employed by different types of decomposition schemes. In the case of PD this number is variable due to the stochastic nature of the algorithm, hence the values show the mean and standard deviation calculated over a 5×2cv set of experiments. Note that the values in the column of OPC correspond to the exact number of classes in each data set.

dichotomies, which is comparable to that of OPC but providing better classification accuracy than the latter in all the tested problems. On the other hand, we can clearly notice the quadratic factor $\mathcal{O}(K^2)$ of all the PWC-like schemes resulting in a much larger number of dichotomies.

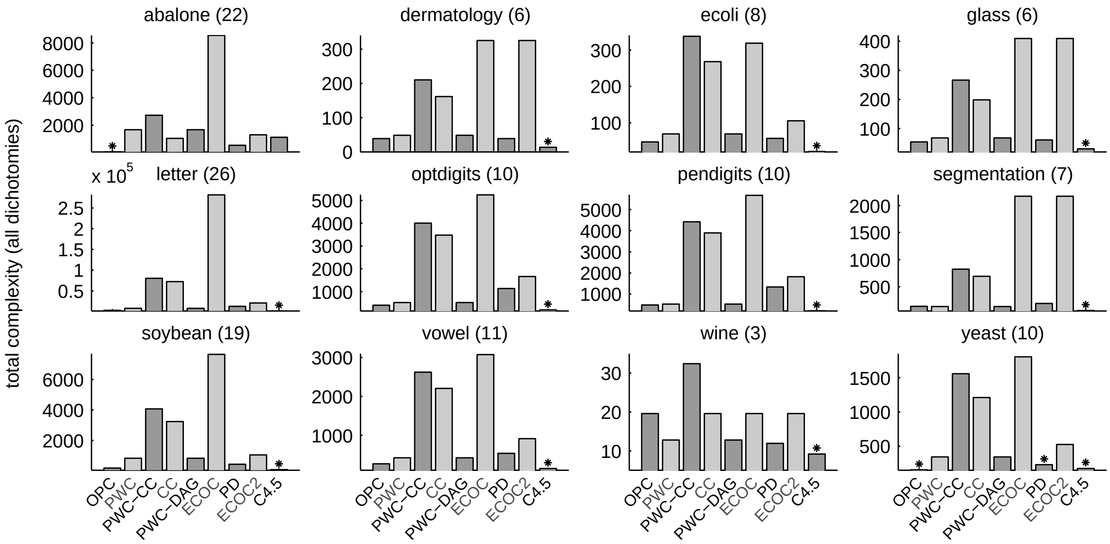


Figure 4.17: Total complexity of the dichotomies generated by different decomposition schemes, including all dichotomies. Lower bars are better. Table 4.2 on page 86 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value (or group of values), with statistical significance at a confidence level of 0.95. For each data set, the number of classes is shown in parenthesis.

4.5 Discussion

This chapter provides an overview of several possible ways of applying two-class learning algorithms to solve multi-class learning problems, through the use of decomposition schemes. We have stated in the beginning of the chapter that this study has been motivated by the need to enable LAD to handle multi-class problems. Nevertheless, the presented results can provide useful guidelines for the application of many other classification algorithms, either Boolean-based or not, to the resolution of problems with more than two classes.

We can conclude that besides enabling two-class classifiers to be used in more general tasks, certain decompositions also allow algorithms with limited capacity to be applied to the resolution of complex learning problems by means of problem decomposition and simplification. In addition, we have observed significant gains in accuracy obtained by the decomposed problem compared to the direct multi-class approach.

Other authors have shown that the accuracy of learning systems can be raised using ensembles of classifiers (Dietterich and Bakiri, 1995; Quinlan, 1996; Bauer and Kohavi, 1999; Dietterich, 2000). For example, methods like *bagging* (Breiman, 1994) and *boosting* (Schapire, 1990; Freund, 1995) improve the classification results by means of voting procedures that collect the answers of different learners applied to the same problem, but where each learner has access to a differently distributed set of training data. Here we have considered ensembles composed of two-class classifiers and have obtained similar conclusions. However, we have also observed that a raise in the accuracy must be compensated by a higher learning effort and an increased complexity of the classification models, which tends to reduce the possibilities of interpreting them.

From the analyzed decomposition schemes, it is interesting to note that they all have different characteristics, and that each one solves differently the tradeoff involving classification accuracy, learning effort, and model complexity. Some of them, like PD and ECOC can even be parameterized, where the user can choose the complexity of each sub-task or the size of the decomposition.

The PD scheme that we have proposed has an interesting set of characteristics compared to the other discussed schemes. Although it does not achieve the same level of accuracy as the very large and complex schemes in a few problems, it still offers a very reasonable performance without demanding an excessive learning effort. It demands few dichotomies to be trained on sub-problems that are in general easy to learn, and this results in an acceptable overall model complexity. In addition, we have shown in Sect. 4.3.3 that the method can be parameterized in order to create even simpler learning tasks, in exchange of a raise in the number of tasks.

ECOC has been shown to allow the highest levels of accuracy among all the tested schemes, and in addition it can be used with an arbitrarily high number of dichotomies (up to $2^{K-1}-1$, where K is the number of classes), which possibly allows additional improvements. Obviously, the model complexity and the training effort also grow proportionally. On the negative side, this scheme will always require certain very difficult tasks to be learned and hence impose the use of algorithms with large capacity.

According to the experiments done, the performance of the plain C4.5 algorithm is only exceeded with regard to the classification accuracy. With respect to all the other measured parameters, C4.5 is better: it is faster, and its classification model is simpler in general than the ensemble of models generated by the decomposition.

To conclude, the choice of the most appropriate decomposition scheme will depend on the size of the problem (notably the number of classes) and on the capacity of the available learning algorithm. For example, with problems containing a large number of classes, the use of schemes like PWC and others derived from it can be infeasible.

	CLASSIFICATION ACCURACY								
DATA SET	OPC	PWC	PWC-CC	CC	PWC-DAG	ECOC	PD	ECOC 2	PLAIN C4.5
abalone	17.55 ±1.44	**25.44** ±1.14	**25.11** ±1.44	22.01 ±1.35	23.28 ±1.03	**26.51** ±0.79	**25.37** ±0.73	**24.81** ±0.93	21.45 ±0.75
dermatology	94.26 ±1.13	**96.12** ±1.21	**97.10** ±1.45	**96.88** ±1.32	**95.30** ±1.76	**97.05** ±1.25	**94.15** ±2.79	**97.05** ±1.25	**95.57** ±1.48
ecoli	**79.36** ±2.61	**81.01** ±1.44	**83.22** ±1.50	**83.04** ±1.47	**80.95** ±2.20	**83.70** ±1.59	**80.78** ±2.29	**80.26** ±2.59	**80.41** ±1.74
glass	**63.46** ±3.66	**64.77** ±3.81	**66.54** ±1.81	**69.06** ±2.85	**62.71** ±3.03	**69.72** ±3.08	**66.92** ±4.16	**69.72** ±3.08	**65.98** ±3.05
letter	83.00 ±0.23	89.08 ±0.27	93.07 ±0.26	93.36 ±0.21	83.45 ±0.45	**95.64** ±0.10	88.10 ±0.60	90.16 ±0.31	84.45 ±0.32
optdigits	87.53 ±0.62	92.77 ±0.42	**96.03** ±0.43	**96.48** ±0.25	89.94 ±0.58	**96.66** ±0.33	92.47 ±0.94	92.96 ±0.99	87.72 ±0.82
pendigits	93.76 ±0.36	95.80 ±0.39	98.24 ±0.19	**98.55** ±0.17	94.83 ±0.40	**98.64** ±0.21	97.03 ±0.60	97.34 ±0.38	94.80 ±0.34
segmentation	93.97 ±1.01	95.56 ±0.63	**96.56** ±0.53	**96.88** ±0.55	**95.44** ±0.71	**97.14** ±0.46	**95.82** ±0.70	**97.14** ±0.46	95.33 ±0.48
soybean	87.45 ±2.11	**91.94** ±1.17	**92.64** ±0.85	91.70 ±0.78	**89.15** ±1.56	**93.55** ±0.60	91.35 ±1.23	91.97 ±1.27	88.89 ±1.09
vowel	66.58 ±2.52	71.96 ±2.31	79.98 ±2.80	**83.09** ±2.27	66.46 ±3.00	**85.53** ±2.16	75.78 ±2.64	76.56 ±3.65	70.53 ±3.03
wine	**85.81** ±4.20	**88.10** ±4.90	**88.10** ±4.26	**85.81** ±4.20	**88.55** ±4.01	**85.81** ±4.20	**86.27** ±4.55	**85.81** ±4.20	**88.89** ±4.27
yeast	**54.37** ±1.12	**57.06** ±1.54	**58.76** ±1.43	**58.52** ±1.50	**55.97** ±1.27	**58.10** ±0.86	**56.35** ±1.83	**55.49** ±2.35	**55.03** ±1.54

Table 4.2: Classification accuracy obtained by different decomposition schemes. The numbers give the average ± standard deviation calculated over a 5×2cv set of experiments, and higher values are better. The numbers typed in bold indicate the best value (or group of values) of a row, with statistical significance at a confidence level of 0.95.

	EXECUTION TIME IN SECONDS								
DATA SET	OPC	PWC	PWC-CC	CC	PWC-DAG	ECOC	PD	ECOC 2	PLAIN C4.5
abalone	14.72 ±0.41	52.09 ±0.51	254.70 ± 5.58	204.31 ± 3.75	65.82 ±0.23	563.24 ± 5.04	37.78 ± 1.81	42.29 ± 1.12	**3.66** ±0.08
dermatology	0.71 ±0.20	1.25 ±0.16	3.04 ± 0.37	2.38 ± 0.18	1.52 ±0.25	4.86 ± 0.37	1.74 ± 0.30	6.17 ± 0.51	**0.23** ±0.14
ecoli	0.62 ±0.02	1.79 ±0.05	4.24 ± 0.07	2.96 ± 0.26	2.29 ±0.03	9.48 ± 0.11	1.91 ± 0.17	2.25 ± 0.10	**0.17** ±0.00
glass	0.64 ±0.13	1.17 ±0.16	2.73 ± 0.24	1.95 ± 0.08	1.38 ±0.15	4.01 ± 0.35	1.62 ± 0.30	5.16 ± 0.30	**0.15** ±0.04
letter	142.85 ±1.31	414.06 ±3.38	3125.86 ±41.05	2753.44 ±165.70	506.06 ±2.86	8088.61 ±221.27	293.42 ±17.69	517.06 ±14.75	**18.34** ±0.57
optdigits	19.71 ±0.49	16.72 ±0.32	138.13 ± 2.03	126.31 ± 4.39	20.83 ±0.40	165.51 ± 3.43	44.87 ± 2.85	52.18 ± 2.04	**4.92** ±0.84
pendigits	16.91 ±0.54	16.69 ±0.41	118.04 ± 2.14	101.38 ± 1.32	28.16 ±0.38	144.45 ± 3.10	39.80 ± 3.79	46.92 ± 3.50	**3.78** ±0.15
segmentation	6.81 ±0.49	3.72 ±0.05	27.84 ± 0.39	24.29 ± 0.54	6.31 ±0.09	74.49 ± 1.44	11.42 ± 0.85	74.27 ± 1.26	**1.63** ±0.08
soybean	3.11 ±0.22	21.88 ±2.31	50.99 ± 0.39	31.35 ± 0.20	18.57 ±0.57	273.81 ± 3.85	10.33 ± 0.78	11.42 ± 0.34	**0.96** ±0.05
vowel	2.97 ±0.11	7.30 ±0.84	28.52 ± 0.55	21.61 ± 0.34	7.22 ±0.38	54.82 ± 1.17	8.18 ± 0.58	10.96 ± 0.78	**0.67** ±0.02
wine	0.31 ±0.01	0.22 ±0.01	0.44 ± 0.01	0.26 ± 0.01	0.30 ±0.02	0.26 ± 0.01	0.33 ± 0.07	0.26 ± 0.01	**0.11** ±0.00
yeast	1.83 ±0.08	4.14 ±0.08	14.45 ± 0.32	10.60 ± 0.33	6.51 ±0.08	28.57 ± 0.42	4.78 ± 0.31	5.86 ± 0.21	**0.63** ±0.03

Table 4.3: Execution time of different decomposition schemes. The numbers give the average ± standard deviation calculated over a 5×2cv set of experiments, and lower values are better. The numbers typed in bold indicate the best value (or group of values) of a row, with statistical significance at a confidence level of 0.95.

	MEAN COMPLEXITY OF EACH DICHOTOMY								
DATA SET	OPC	PWC	PWC-CC	CC	PWC-DAG	ECOC	PD	ECOC 2	PLAIN C4.5
abalone	**2.73** ±0.97	8.00 ±0.44	6.50 ±0.32	4.98 ±0.80	8.00 ±0.44	40.70 ±4.01	17.73 ± 3.60	41.78 ± 5.76	1127.10 ±29.54
dermatology	6.50 ±0.70	**3.22** ±0.12	7.00 ±0.45	10.77 ±0.84	**3.22** ±0.12	10.49 ±0.73	6.21 ± 0.61	10.49 ± 0.73	13.80 ± 0.98
ecoli	5.92 ±1.10	**2.47** ±0.20	6.03 ±0.49	9.58 ±0.88	**2.47** ±0.20	11.38 ±1.66	6.18 ± 0.62	11.75 ± 2.03	21.00 ± 4.56
glass	9.03 ±1.30	**4.54** ±0.21	8.89 ±0.56	13.23 ±1.08	**4.54** ±0.21	13.19 ±0.97	8.75 ± 1.45	13.19 ± 0.97	30.80 ± 5.62
letter	114.36 ±9.67	**24.49** ±0.32	123.69 ±1.07	222.89 ±2.11	**24.49** ±0.32	866.60 ±7.62	516.52 ±27.21	871.08 ±17.59	1611.80 ±27.82
optdigits	39.98 ±2.40	**11.58** ±0.30	44.45 ±0.96	77.33 ±1.74	**11.58** ±0.30	116.44 ±2.89	81.36 ± 6.24	118.57 ± 8.74	199.60 ±10.88
pendigits	47.98 ±1.99	**11.58** ±0.54	49.07 ±1.55	86.55 ±2.78	**11.58** ±0.54	125.86 ±4.11	94.96 ± 5.82	129.89 ± 7.72	210.80 ±10.60
segmentation	19.51 ±1.32	**6.25** ±0.41	19.57 ±0.65	32.88 ±1.18	**6.25** ±0.41	34.48 ±1.32	20.39 ± 2.00	34.48 ± 1.32	59.00 ± 4.90
soybean	10.34 ±0.55	**4.88** ±0.07	11.92 ±0.26	18.96 ±0.53	**4.87** ±0.08	44.79 ±1.10	19.04 ± 1.50	44.28 ± 2.26	92.70 ± 6.20
vowel	24.38 ±1.91	**7.60** ±0.15	23.84 ±0.31	40.08 ±0.71	**7.60** ±0.15	55.97 ±2.33	34.34 ± 1.99	56.98 ± 5.26	143.20 ±13.19
wine	**6.53** ±1.15	**4.27** ±0.47	**5.40** ±0.63	**6.53** ±1.15	**4.27** ±0.47	**6.53** ±1.15	**5.67** ± 1.00	**6.53** ± 1.15	9.20 ± 1.08
yeast	15.70 ±2.18	**7.68** ±0.61	17.30 ±1.47	26.91 ±2.84	**7.68** ±0.61	40.12 ±4.44	17.58 ± 2.98	40.54 ± 8.61	177.20 ±21.25

Table 4.4: Mean complexity of the dichotomies generated by different decomposition schemes. The numbers give the average ± standard deviation calculated over a 5×2cv set of experiments, and lower values are better. The numbers typed in bold indicate the best value (or group of values) of a row, with statistical significance at a confidence level of 0.95.

	TOTAL MODEL COMPLEXITY, INCLUDING ALL THE DICHOTOMIES								
DATA SET	OPC	PWC	PWC-CC	CC	PWC-DAG	ECOC	PD	ECOC 2	PLAIN C4.5
abalone	**57.40** ±20.30	1680.20 ±92.75	2727.20 ±134.61	1047.00 ±168.59	1680.20 ±92.75	8547.40 ±842.81	539.50 ± 93.97	1295.10 ±178.53	1127.10 ± 29.54
dermatology	39.00 ± 4.22	48.40 ± 1.80	210.00 ± 13.42	161.60 ± 12.56	48.40 ± 1.80	325.20 ± 22.60	39.10 ± 4.23	325.20 ± 22.60	**13.80** ± 0.98
ecoli	47.40 ± 8.77	69.20 ± 5.74	337.40 ± 27.32	268.20 ± 24.52	69.20 ± 5.74	318.60 ± 46.55	57.50 ± 8.09	105.80 ± 18.29	**21.00** ± 4.56
glass	54.20 ± 7.77	68.20 ± 3.12	266.60 ± 16.95	198.40 ± 16.22	68.20 ± 3.12	409.00 ± 30.08	61.20 ± 13.14	409.00 ± 30.08	**30.80** ± 5.62
letter	3.0E3 ±0.3E3	8.0E3 ±0.1E3	80.4E3 ± 0.7E3	72.4E3 ± 0.7E3	8.0E3 ±0.1E3	281.6E3 ± 2.5E3	12.6E3 ± 0.5E3	20.9E3 ± 0.4E3	**1.6e3** ±0.03E3
optdigits	399.80 ±24.02	521.00 ±13.71	4001.00 ± 86.20	3480.00 ± 78.41	521.00 ±13.71	5239.80 ±130.23	1135.00 ± 71.67	1660.00 ±122.42	**199.60** ± 10.88
pendigits	479.80 ±19.89	521.00 ±24.12	4415.80 ±139.42	3894.80 ±125.21	521.00 ±24.12	5663.80 ±185.06	1339.70 ±154.22	1818.40 ±108.07	**210.80** ± 10.60
segmentation	136.60 ± 9.24	131.20 ± 8.55	821.80 ± 27.24	690.60 ± 24.88	131.20 ± 8.55	2172.40 ± 83.16	188.90 ± 16.46	2172.40 ± 83.16	**59.00** ± 4.90
soybean	196.40 ±10.37	833.60 ±12.75	4075.50 ± 88.01	3241.90 ± 89.75	831.90 ±13.94	7659.50 ±188.30	449.00 ± 44.53	1062.70 ± 54.33	**92.70** ± 6.20
vowel	268.20 ±21.06	417.80 ± 8.21	2622.20 ± 33.92	2204.40 ± 39.20	417.80 ± 8.21	3078.20 ±128.42	532.90 ± 51.71	911.60 ± 84.18	**143.20** ± 13.19
wine	19.60 ± 3.47	12.80 ± 1.40	32.40 ± 3.77	19.60 ± 3.47	12.80 ± 1.40	19.60 ± 3.47	11.90 ± 2.62	19.60 ± 3.47	**9.20** ± 1.08
yeast	**157.00** ±21.77	345.60 ±27.50	1556.60 ±132.44	1211.00 ±127.57	345.60 ±27.50	1805.40 ±199.95	**232.00** ± 42.00	527.00 ±111.92	**177.20** ± 21.25

Table 4.5: Total complexity of the dichotomies generated by different decomposition schemes, including all dichotomies. The numbers give the average ± standard deviation calculated over a 5×2cv set of experiments, and lower values are better. The numbers typed in bold indicate the best value (or group of values) of a row, with statistical significance at a confidence level of 0.95.

Chapter 5

LAD in a multi-class context

Motivation

In the previous chapter we have studied several methods for decomposing a classification problem involving more than two classes into several simpler binary sub-problems, and also how to recombine the solutions to the sub-problems in order to generate a final classification decision. The main motivation for the study has been to allow binary learning algorithms to be applied to problems with any number of classes. Despite the fact that a specific learning algorithm has been used in the reported experiments (C4.5, based on decision trees) the tested procedures can be applied to a wide range of algorithms. As we have shown, even those that are able to handle multi-class problems directly can benefit from the use of decomposition techniques in order to solve simpler problems and even to raise their accuracy.

In this chapter, we extend some of the decomposition ideas to the particular case of LAD, and try to integrate them into its core, with the intent of creating a multi-class version of the method. If LAD can, like many other methods, make use of decomposition schemes in order to be able to solve multi-class problems, here we attempt to go one step further, and try to take advantage of the decomposition principles in order to integrate them in a natural way into the internal functioning of the method.

The chapter starts by briefly discussing the option of applying decomposition schemes to standard LAD[1] in Sect. 5.1. Section 5.2 enunciates the basic principles allowing the proposed goal to be attained, and then provides a detailed description of the algorithm. Section 5.3 presents and discusses experimental results. Section 5.4 concerns the type of patterns that are generated by this new type of model, and how the interpretability is maintained.

[1] We use the mention "standard LAD" to refer to the implementation described by Boros et al. (1996).

5.1 Evident approach to multi-class LAD

5.1.1 Standard LAD revisited

We have seen in Chap. 2 how the standard LAD classification model is constructed. For training, we are given two sets of data in Boolean format, $\mathcal{Y}^+$ and $\mathcal{Y}^-$, which respectively contain the positive and the negative training instances. To construct the model we need to generate a positive and a negative pattern set ($\mathcal{P}^+$ and $\mathcal{P}^-$). The former contains a set of patterns that together cover all the instances in $\mathcal{Y}^+$ and (nearly) no instances in $\mathcal{Y}^-$, while the set of patterns in $\mathcal{P}^-$ cover all the negative instances and (nearly) no positive ones. We employ the term "nearly" to account for noise in the training data, according to the discussion of Sect. 2.3.6 on page 26. After the patterns have been generated, they all receive a certain weight according to one of the procedures mentioned in Sect. 2.3.3, and this is all that is required to form the LAD classification theory that we have presented in Eq. (2.1) on page 21 and that we recall below:

$$\hat{F}(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \;\; \Delta(\boldsymbol{x}) > \tau^+ \\ 0 & \text{if} \;\; \Delta(\boldsymbol{x}) < \tau^- \end{cases} , \qquad \Delta(\boldsymbol{x}) = \sum_{p_i \in \mathcal{P}^+} w_i \cdot p_i(\boldsymbol{x}) + \sum_{p_j \in \mathcal{P}^-} w_j \cdot p_j(\boldsymbol{x})$$

$$\tau^+ \geq \tau^- , \quad w_i \geq 0 \;\; \forall i, \quad w_j \leq 0 \;\; \forall j .$$

where $\boldsymbol{x}$ is an instance to be classified, $\hat{F}$ is the decision rule that determines whether $\boldsymbol{x}$ is a positive instance (if the output is 1), or a negative one (if the output is 0), p is a pattern, and w represents its weight. Note that the weights w_i have a positive value while the w_j have a negative one.

5.1.2 Problem decomposition

LAD can be applied to solve multi-class problems through the use of a decomposition scheme, which is a natural consequence of the discussion held in the previous chapter. Using this option is straightforward: each dichotomy is implemented by a different LAD theory, and the dichotomies are defined using one of the mentioned decomposition schemes.

There is one aspect worth of notice concerning this option: we have seen in Chap. 2 that prior to generating the patterns, a pre-processing phase is usually required in order to transform the raw input data into Boolean format. Chapter 3 is entirely dedicated to this problem and therein we have demonstrated that this Boolean transformation can be done naturally in a multi-class manner. This means that we need to transform the whole data set only once and then generate each dichotomy from the same obtained set of Boolean images, i.e. there is no necessity to transform the raw data independently for every generated dichotomy.

5.2 A multi-class model based on patterns

One of the conclusions of the previous chapter concerning the choice of whether or not to use decomposition schemes with methods that can be directly applied to multi-class problems was that there is usually a tradeoff between the classification accuracy, the complexity of the model, and the computational time that is required to generate it. The empirical experiments have shown that the decomposed version of the problem usually yields a better classification accuracy compared to the monolithic multi-class classifier but, on the other hand, that the global complexity of the former, including all the generated sub-models, also increased, as well as the amount of time spent to generate them.

We would like to open the possibility of employing LAD at different levels of this tradeoff, by creating a learning algorithm that builds a single multi-class classification model based on patterns, and that is what we describe in the following. We propose to achieve this goal by establishing a connection between the main principles of the basic LAD model and the decomposition schemes studied in the previous chapter.

5.2.1 Description of the model

The procedure that we propose here to generate a classification model based on patterns relies on the three following main ideas:

- the model is composed of a unique, common pattern set that is shared by all classes;
- the relation between the patterns and the classes is described by a matrix which is similar to the decomposition matrix used in the schemes described in the previous chapter;
- the set of patterns is constructed by an iterative procedure, that creates one pattern at a time.

The interest of the latter is twofold: (i) at each iteration, the next pattern generation task can be appropriately defined, according to the current state of the model, and (ii) every pattern aimed at separating two particular classes can eventually be shared by the other classes after it has been created.

The constituents of the model

In Sect. 2.3.1 on page 17 we have defined a pattern as a term covering positive examples and no negative ones. This definition is suitable to two-class problems, where only positive and negative data are considered. In the current context, we have to adapt the definition, because there can be more than two classes involved. Inspired by the way in which a dichotomy (which is also a binary unit) deals with a multi-class situation, we consider that a pattern is more generally a term that covers instances of one or more classes and covers no instances of one or more classes.

As suggested above, we intend to create a model containing a single set of patterns. In order to define the relation between each pattern and the various classes we use a decomposition matrix $\boldsymbol{D}$. We determine that each value D_{pk} of the matrix corresponds to the number of instances of class c_k covered by a pattern p, divided by the total number of instances of class c_k. However, when the coverage of p on class c_k is null, then D_{pk} takes the value -1. The reason why it does not take the value 0 instead is related to robustness. A small coverage of a pattern on a class makes it acquire a positive status, which is represented by a small (positive) value in the respective component of the decomposition matrix. If the negative character of a class is represented by 0 in $\boldsymbol{D}$, then this does not allow sufficient distinguishing properties to be created between positive and negative classes for each pattern.

Nevertheless, the choice of -1 is somewhat arbitrary. A better choice might be a value that takes the class prior probability into account, where classes with large cardinality receive a higher negative value than those of smaller cardinality.

Hence, the model is composed of a pair $\langle \mathcal{P}, \boldsymbol{D} \rangle$ containing the pattern set $\mathcal{P}$ and the respective decomposition matrix $\boldsymbol{D}$ of size $P \times K$, where P is the number of patterns in $\mathcal{P}$. Below we will describe the basic property that such a model must have in order to be successfully used for classification tasks, and afterwards we will present an algorithm for constructing it.

Basic property of the model

We have shown in Sect. 2.3.1 that the LAD model contains two pattern sets that together distinguish the data of one class from the other and vice-versa. This binary-based principle can also be applied to multi-class models. We formulate it as follows: for any two training instances $\boldsymbol{x}$ and $\boldsymbol{x}'$ such that $F(\boldsymbol{x}) \neq F(\boldsymbol{x}')$ (i.e. they belong to different classes), there must be at least one pattern in the set $\mathcal{P}$ that covers $\boldsymbol{x}$ without covering $\boldsymbol{x}'$ nor any other instance of the same class as $\boldsymbol{x}'$, and another pattern that covers $\boldsymbol{x}'$ without covering $\boldsymbol{x}$ nor any other instance of the same class as $\boldsymbol{x}$.

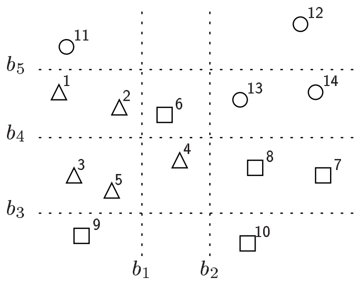
This condition can be re-stated in an alternative, more general form, oriented towards classes instead of instances: for any two classes c_i and c_j, there must exist a pattern subset $\mathcal{P}_{ij} \subset \mathcal{P}$ whose patterns together cover the totality of the instances of c_i without covering any instance of c_j. The opposite must also be verified, i.e. there must exist a subset $\mathcal{P}_{ji} \subset \mathcal{P}$ of patterns covering the totality of the instances of c_j without covering any instance of c_i.

An example

In order to give a general idea of the type of model that we intend to generate, we will use a small example. Figure 5.1 on the facing page shows a simple three-class data set in a two-dimensional space (on the left). The horizontal and vertical dotted lines represent the discriminants created for the Boolean mapping (cf. Chap. 3), and therefore correspond to the Boolean attributes $b_1, \ldots, b_5$ that are used in the table on the right to represent the data in Boolean format. We recall that the Boolean value of an instance for a binary attribute b is 1 if the instance is above or to the right of the respective discriminant, and 0 if it is below or to the left of it[2].

Table 5.1 on the facing page shows a possible model for the data set of the previous figure. It contains nine patterns $p_1, \ldots, p_9$, each charged of a specific separation task. Observe how in the current context the patterns are still binary units, but instead of being related to positive and negative instances, as originally, now they are related to data of various classes, and where

[2]This is valid for ordered attributes. See Sect. 3.1.1 on page 34 for other attribute types.



LABELS	MULTIPL.	CLASS	b_1	b_2	b_3	b_4	b_5
1,2	2	△	0	0	1	1	0
3,5	2	△	0	0	1	0	0
4	1	△	1	0	1	0	0
6	1	□	1	0	1	1	0
7,8	2	□	1	1	1	0	0
9	1	□	0	0	0	0	0
10	1	□	1	1	0	0	0
11	1	○	0	0	1	1	1
12	1	○	1	1	1	1	1
13,14	2	○	1	1	1	1	0

Figure 5.1: Example of a two-dimensional data set with three classes (△, □, and ○). The table on the right shows a binary coding of the data using the discriminants represented by the horizontal and vertical dotted lines in the figure on the left.

PATTERN		COVERAGE △	□	○	MATRIX $\boldsymbol{D}$ △	□	○
p_1 :	$\bar{b}_1 b_3$	4	–	1	0.80	-1	0.25
p_2 :	$\bar{b}_4$	3	4	–	0.60	0.80	-1
p_3 :	b_2	–	3	3	-1	0.60	0.75
p_4 :	b_5	–	–	2	-1	-1	0.50
p_5 :	$b_2 b_4$	–	–	3	-1	-1	0.75
p_6 :	$\bar{b}_2 \bar{b}_5$	5	2	–	1.00	0.40	-1
p_7 :	$\bar{b}_3$	2	–	–	-1	0.40	-1
p_8 :	$b_1 \bar{b}_2 \bar{b}_4$	1	–	–	0.20	-1	-1
p_9 :	$b_1 b_4$	–	1	3	-1	0.20	0.75

Table 5.1: A possible classification model for the data set of Fig. 5.1. It contains nine patterns, each covering data of one or more classes and covering no data of one or more classes. The decomposition matrix $\boldsymbol{D}$ is derived from the ratio of examples of each class covered by each pattern, except when that coverage is null, it which case the value in the matrix is -1.

each class takes a positive or a negative status in view of each pattern. This has a similarity with the notion of dichotomy as presented in the previous chapter.

It can be verified that this model contains the basic properties presented above. In Sect. 5.2.5 we will explain how a model of this type is used to output classification decisions.

5.2.2 Handling noise

In the presence of noise, the basic properties of the model described in the previous section can be very restrictive, since they will possibly demand the creation of patterns with very specific coverage to handle noisy examples, and which may degrade the generalization performance of the model. There are two basic mechanisms that can be used to avoid this, to a certain extent. One of them consists in relaxing the definition of pattern in order to allow a small negative coverage, as already proposed in Sect. 2.3.6 on page 26 in the framework of standard LAD. Another possibility for handling noise and avoiding the model to overfit the training data is by reducing the amount by which each pair of classes must be distinguished, i.e. the ratio of examples of one class that must be separated from the examples of the other class.

We call the *differentiability rate* of a class c_i from another class c_j as the ratio of examples of c_i that are covered by patterns in $\mathcal{P}$ which do not cover the examples of c_j (or that cover only a small amount of them, as suggested above). In the following we will denote differentiability rates by r, and their values lie in the range $[0, 1]$. Note that this concept is not a symmetric one, i.e. the differentiability rate of c_i from c_j gives no information about the differentiability rate of c_j from c_i.

In the algorithm that we propose, there is a user-defined parameter $r_{\min}$ that determines the target differentiability rate that must be achieved between all the pairs of classes. If we set $r_{\min} = 1$, we impose a complete separability between the training data of every pair of different classes. So, a value below 1 can be used in noisy contexts.

5.2.3 The MultiClassLAD algorithm

In the previous section we have described the type of classification model proposed. Below, we present an iterative algorithm for generating such a model, inspired by the algorithm of pertinent dichotomies (Alg. 4.1 on page 75) proposed in Chap. 4.

In Alg. 4.1 each dichotomy was generated from a pair of classes (c_i, c_j) (the germ). We use a similar approach in order to generate a pattern. c_i becomes the positive class and c_j the negative one. After the pattern p has been created, it is tested on the remaining classes, so that the values in the respective row of the decomposition matrix can be defined. More details are given below.

Pattern sharing

Each pattern is generated based on a pair of classes. After it has been created, its behavior with regard to the remaining classes in the problem is examined. These classes can acquire three different status with regard to a new pattern: positive, negative, and neutral. Consider $\underline{\omega}_k^p$ as the *coverage rate* of p on c_k, i.e. the proportion of examples from class c_k covered by pattern p, lying in the range $[0, 1]$. We have defined two parameters `maxNegCovRate` and `minPosCovRate` that are used as thresholds placed in the range of $\underline{\omega}_k^p$, in order to decide this status. They work as follows:

- if $\underline{\omega}_k^p$ is above `minPosCovRate`, then class c_k can be considered positive with regard to p ;
- else, if $\underline{\omega}_k^p$ is below `maxNegCovRate`, then c_k is a negative class;
- otherwise, $\underline{\omega}_k^p$ is between `maxNegCovRate` and `minPosCovRate`, and c_k is considered neutral.

Obviously, this assumes that `maxNegCovRate` $<$ `minPosCovRate`. The gap between these two values creates a "neutral security region" between positive and negative status (see Fig. 5.2). The parameter `maxNegCovRate` must have a small value close or equal to 0, where a non-null value can be used in order to tolerate noise. The value of `minPosCovRate` must be greater than `maxNegCovRate`, but too high a value may prevent the generated patterns to be shared by other classes.

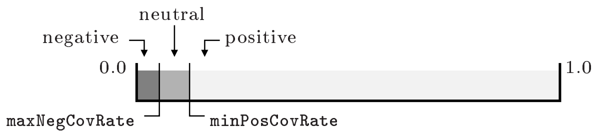


Figure 5.2: Parameters used to determine the status of each class with regard to a pattern. The bar shows the possible range of $\underline{\omega}_k^p$, i.e. the proportion of training examples from class c_k covered by pattern p.

Choice of the germ classes and stopping criterion

There are two important questions that must be answered in order to appropriately define the iterative model constructing procedure:

- what is the pattern that must be generated at each iteration (more precisely, what is the germ);
- when should the procedure stop, i.e. when is the model terminated.

We use a matrix $\boldsymbol{R}$ of size $K \times K$ (K is the number of classes), where each component R_{ij} stores the differentiability rate of class c_i from class c_j (defined in Sect. 5.2.2 on page 93). In the algorithm, the values of $\boldsymbol{R}$ are initialized to 0, except for the values in the diagonal, which are initialized to 1 (they are meaningless, anyway). In the course of the procedure, the values of $\boldsymbol{R}$ must be progressively raised by the added patterns.

The matrix of differentiability rates serves two purposes, related to the two aspects mentioned above: on the one hand, it allows the choice of the appropriate germ classes for generating the next pattern, which are those corresponding to the minimal value in $\boldsymbol{R}$ (the least separated classes); on the other hand, it serves at verifying whether the stopping criterion is attained. This is done by defining a minimal differentiability rate $r_{\min}$ to be attained by all the pairs of classes in the problem, as already explained in Sect. 5.2.2 on page 93.

The learning algorithm

We now have the necessary elements to define an algorithm for generating a classification model based on patterns. We are given a set of training data $\mathcal{Y}$ in Boolean format containing K classes. Each subset $\mathcal{Y}_k$ contains the data of class c_k ($\mathcal{Y} \equiv \bigcup_{k=1}^{K} \mathcal{Y}_k$). The algorithm creates a model $\langle \mathcal{P}, \boldsymbol{D} \rangle$ composed of the pattern set $\mathcal{P}$ and of the decomposition matrix $\boldsymbol{D}$ of size $P \times K$ (P being the number of patterns in $\mathcal{P}$), with values in the range $[-1, +1]$, and where the value of D_{pk} determines the status of class c_k with regard to pattern p, as defined previously.

The proposed learning algorithm is called **MultiClassLAD** and it is presented in Alg. 5.1 on the next page. In addition to the matrix $\boldsymbol{R}$, there is an important auxiliary data structure used in the algorithm: a matrix-like container where each component corresponds to the subset of patterns $\mathcal{P}_{ij} \in \mathcal{P}$ differentiating class c_i from class c_j. There are as many components $\mathcal{P}_{ij} \in \mathcal{P}$ as pairs of classes. Each one of these components can more easily be interpreted as a set of pointers to specific patterns in the global pattern set $\mathcal{P}$.

Algorithm 5.1 MultiClassLAD($\mathcal{Y}$)

Inputs: $\mathcal{Y}$: set of training data in Boolean format

Parameters:
$r_{\min}$: minimal differentiability rate
`maxNegCovRate` : maximal negative coverage rate
`minPosCovRate` : minimal positive coverage rate

$R_{ij} \leftarrow \begin{cases} 0.0 & \text{if } i \neq j \\ 1.0 & \text{if } i = j \end{cases}$ [*Matrix of class differentiability rates*]
$\mathcal{P}_{ij} \leftarrow \varnothing$ for all i, j [*Pattern subsets separating classes*]
$\mathcal{P} \leftarrow \varnothing$ [*Global pattern set*]
while exists some $R_{ij} < r_{\min}$ **do**
  $i, j \leftarrow \arg\min_{i,j} R_{ij}$ [*Indices of the pair of least separated classes*]
  $\mathcal{Y}_{uncovered} \leftarrow$ subset of examples from class c_i not yet covered by the pattern subset $\mathcal{P}_{ij}$
  $p \leftarrow$ **GeneratePattern**$(\mathcal{Y}_{uncovered}, \mathcal{Y}_j)$
  if $p = \varnothing$ **then** [*no pattern found*]
    $R_{ij} \leftarrow r_{\min}$ [*do not try to further separate c_i from c_j*]
  else
    $\mathcal{P} \leftarrow \mathcal{P} \cup p$
    [*Determine the status of each class with regard to the new pattern*]
    for all classes c_k **do**
      $\underline{\omega}_k^p \leftarrow$ coverage rate of p on $\mathcal{Y}_k$
      $D_{pk} \leftarrow \begin{cases} \underline{\omega}_k^p & \text{if } \underline{\omega}_k^p \geq \texttt{minPosCovRate} \\ 0 & \text{if } \texttt{maxNegCovRate} < \underline{\omega}_k^p < \texttt{minPosCovRate} \\ -1 & \text{if } \underline{\omega}_k^p \leq \texttt{maxNegCovRate} \end{cases}$
    end for
    [*Update $\boldsymbol{R}$ and the various $\mathcal{P}_{ij}$ based on p*]
    for all classes c_i such that $D_{pi} > 0$ **do**
      for all classes c_j such that $D_{pj} = -1$ **do**
        $\mathcal{P}_{ij} \leftarrow \mathcal{P}_{ij} \cup p$
        $R_{ij} \leftarrow$ coverage rate of the pattern subset $\mathcal{P}_{ij}$ on $\mathcal{Y}_i$
      end for
    end for
  end if
end while
return$(\langle \mathcal{P}, \boldsymbol{D} \rangle)$ [*Return the classification model*]

Although the goal of Alg. 5.1 is to solve problems with several classes, it may as well be applied to two-class classification tasks. Therefore, in Sect. 5.3.1 we will present results of an empirical comparison between the procedure **MultiClassLAD** and the standard LAD approach, using two-class data sets.

The principle of sequential covering

We have already cited several sequential covering algorithms for rule induction in Sect. 2.4.2 on page 28. These algorithms generate a set of rules to distinguish a set of positive instances (representing the target concept) from a set of negative ones. They start by generating a first rule to cover the maximal possible number of positive instances. These instances are discarded and another rule is generated to separate the remaining positive instances from the negative class. This process is repeated iteratively until no positive instances remain, and the classification model consists of the set of generated rules. The usual procedure for handling multi-class problems (used by algorithms like CN2 (Clark and Niblett, 1989; Clark and Boswell, 1991), TABATA (Brézellec and Soldano, 1998) or AQ (Michalski, 1969)) is to apply a one-per-class scheme, where each class in turn becomes the target concept.

The procedure **MultiClassLAD** is composed of two main parts, like every sequential covering algorithm: the high-level loop that defines which pattern to generate at each iteration, and the low-level pattern generation procedure called **GeneratePattern** inside Alg. 5.1. The major difference between **MultiClassLAD** and the mentioned algorithms is the sharing of a single pattern set among all the classes, where each class may have one of the status positive/negative/neutral with regard to each pattern. CN2 and TABATA also share a single rule set among all classes, but in that case every rule covers examples of a single class exclusively, which means that for every rule one of the classes is positive and all the others are negative.

5.2.4 Generating a single pattern

In standard LAD the patterns are generated collectively, i.e. there is a procedure that browses the search space of terms and collects all the relevant patterns, as described earlier in Sect. 2.3.5 on page 22. In the current context, we need a procedure that browses that search space in order to find a single pattern suitable to each particular task defined in the high-level loop. The next chapter is totally dedicated to this problem. It explains how an heuristic search procedure (Tabu search) is used for that matter. For now we assume the existence of a procedure **GeneratePattern**($\mathcal{Y}^+$,$\mathcal{Y}^-$) that returns a pattern covering as much of the instances in $\mathcal{Y}^+$ as possible, and covering (nearly) no instance from $\mathcal{Y}^-$, more precisely, covering at most a ratio `maxNegCovRate` of the negative instances.

It is important to remark that **GeneratePattern** may not be able to find a pattern according to the required specifications. In this case, it is possible to see in Alg. 5.1 on the facing page that the pair of classes involved is thereafter considered separated, even if the minimal required differentiability rate between them is not attained. This is justified by the limited capacity of the model, which may possibly not be able to fit the training data as required by the value of the parameter $r_{\min}$.

5.2.5 Making classification decisions

After the classification model has been created by the procedure **MultiClassLAD**, it is necessary to define how the classification decisions are made. It follows naturally that the reconstruction strategy used in the current context is similar to the standard reconstruction defined in the previous chapter in Eq. (4.4) on page 63:

$$\hat{F}(\boldsymbol{x}) = \arg\max_k \sum_i p_i(\boldsymbol{x}) \cdot D_{ik} . \tag{5.1}$$

There is, however, a difference concerning the outcome of the partial decisions. While in Eq. (4.4) each dichotomy $\hat{f}$ produces a value in $\{-1, +1\}$, in the case of Eq. (5.1) above the expression $p_i(\boldsymbol{x})$ returns 1 if the pattern p_i covers the example $\boldsymbol{x}$, and returns 0 otherwise.

5.3 Empirical evaluation

In this section we present an empirical evaluation of the **MultiClassLAD** algorithm (denoted MC-LAD). First, we compare it with the standard version of LAD, i.e. the one described by Boros et al. (1996) (Std-LAD), confined to two-class problems. Afterwards we compare MC-LAD with three other algorithms on the whole collection of data sets employed in this dissertation, including both two-class and multi-class problems. Two of these algorithms are based on sequential covering rule induction: CN2 and TABATA. The latter has the particularity of using a Tabu search approach for generating patterns, which is also the case of MC-LAD. This comparison essentially concerns the subject of the following chapter, where the approach for single pattern generation is discussed, and where TABATA is described in more detail. The CN2 algorithm has been included for being one of the most cited rule-based algorithms in the literature. Finally, we have also included the C4.5 decision tree generator, which has been used in this dissertation on various occasions, and which is one of the most cited algorithms in the Machine Learning literature in general. Despite the fact that other powerful learning algorithms exist, based on Artificial Neural Networks or Support Vector Machines, for example, C4.5 has the advantage of being fast and requiring no parameter tuning.

The tested quantities are the classification accuracy, the number of patterns/rules in the generated classification models (except for C4.5), and the execution time. Figures 5.3 to 5.5 present the results of the comparison between standard and multi-class LAD on problems with two classes. The results concerning the comparison between multi-class LAD and the other three alternative algorithms are given in Figures 5.6 to 5.8. The tables with detailed numerical results are presented at the end of the chapter on pages 109 and following.

Algorithm configuration

Concerning the particular conditions under which the algorithms have been tested, the parameter values used in MC-LAD are reported in Table 5.2. In addition, we have established a limit

PARAMETER	VALUE	MEANING
$r_{\min}$	1.0	minimal class differentiability rate
`minPosCovRate`	0.05	minimal positive coverage rate
`maxNegCovRate`	0.02	maximal negative coverage rate

Table 5.2: The parameters of the algorithm MC-LAD.

for the degree of the searched patterns ($G_{max} = 5$). This aspect is further discussed in the next chapter. The choice of the values presented in Table 5.2 does not derive from intensive experimentation, but rather from common sense. Hence, it is possible that better values could have been selected. We presume that this choice does not invalidate the presented results.

In the case of standard LAD, the same maximal pattern degree has been applied ($G_{max} = 5$) and the patterns are generated with the breadth-first search procedure described in Sect. 2.3.5 on page 22. The complementary top-down search phase described in the same section has not been applied. After the patterns have been generated and received appropriate weights, the patterns sets are filtered using the set covering approach described in Sect. 2.3.5, which finds a minimal pattern set allowing each instance to be covered by at least one pattern[3]. The transformation of the input data into Boolean format has been done using the IDEAL algorithm proposed in Chap. 3, with plain consistency.

Concerning the other algorithms (CN2, TABATA, and C4.5), they have all been run with their default parameter values. CN2 generates rules using a strict bottom-up approach, as described earlier in Sect. 2.4.2. The rule-generation engine of TABATA is described in more detail in the next chapter, in Sect. 6.4.1. Both these algorithms use a sequential rule covering for the high-level loop, as MC-LAD, but where each class is considered in turn as the concept to be learned against all the others.

In CN2 and TABATA the classification procedure is slightly different from the one used in MC-LAD (cf. Sect. 5.2.5): instead of using a decomposition matrix, they label each rule with the class and the number of instances that it covers (note that each rule always covers examples from a single class). For classifying a new example, either a single rule is matched, which immediately provides a class decision, or the class coverages are collected from the ensemble of matching rules, with the decision corresponding to the class with the highest count.

5.3.1 Standard versus multi-class LAD

Classification accuracy

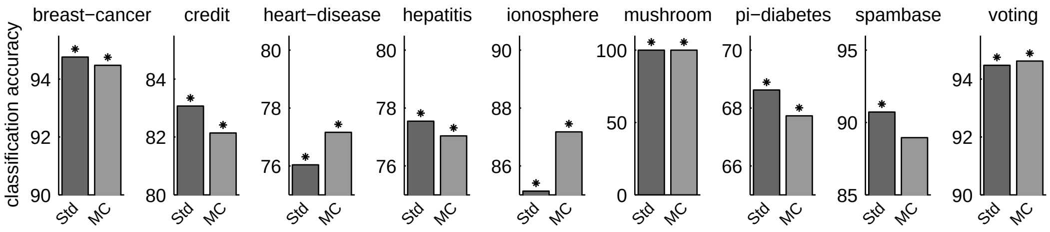


Figure 5.3: Classification accuracy of MC-LAD compared to standard LAD. Higher bars are better. Table 5.3 on page 109 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value, with statistical significance at a confidence level of 0.95.

Figure 5.3 indicates a comparable accuracy of the standard and the multi-class versions of LAD. They have alternate wins, but the standard version is significantly better on `spambase`. In any case, the maximal observed difference in accuracy is close to 2%, which is a small value.

[3] Note that if the top-down pattern search procedure is not used, it is possible that some training instances are covered by no pattern. In this case the pattern filtering phase can do no better than guarantee the coverage of those instances already covered by patterns found in the bottom-up search phase.

Number of patterns

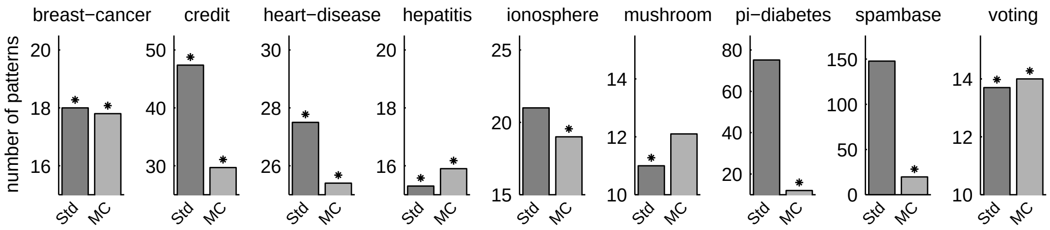


Figure 5.4: Number of patterns generated by MC-LAD and by standard LAD. Lower bars are better. Table 5.4 on page 109 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value, with statistical significance at a confidence level of 0.95.

The results of Fig. 5.4 show some large differences in the number of generated patterns. The standard version is slightly better in two of the problems, by less than one pattern of difference, on average. On the contrary, MC-LAD generates a considerably smaller number of patterns on three data sets, and a slightly smaller number on three others. We remark that in the case of Std-LAD two independent pattern sets are generated, one positive and one negative, hence the number of patterns is counted as the sum over both sets.

Execution time

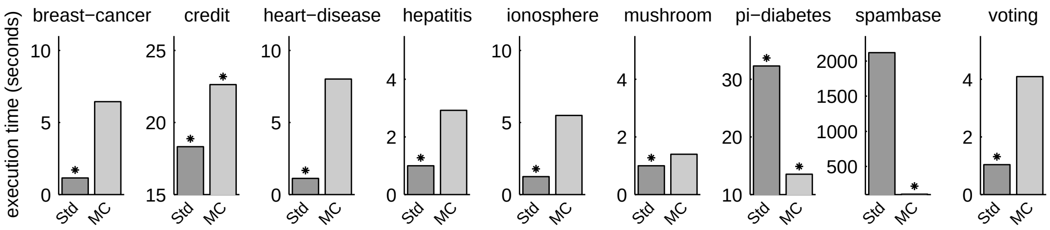


Figure 5.5: Execution time of the MC-LAD and of the standard LAD algorithms. Lower bars are better. Table 5.5 on page 109 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value, with statistical significance at a confidence level of 0.95.

The results of the execution time (Fig. 5.5) provide a definitive basis for differentiating the two algorithms. It can be seen that with the smaller data sets Std-LAD is faster by a few seconds. However, in the bigger ones (`pi-diabetes` and especially `spambase`) Std-LAD clearly shows how it badly scales with the size of the problem. Even more severe is its behavior with the `adult` data set, the largest used in this dissertation in terms of number of examples (almost 50 000), and one of the largest in terms of binary attributes (about 140, as shown in Chap. 3). No comparative results are shown here for this data set, due to the fact that Std-LAD has executed for at least four weeks without generating a single classification model. This gives experimental evidence of the exponential complexity of this algorithm, and hence of its lack of generality.

5.3.2 Multi-class LAD versus alternative approaches

The comparison reported below constitutes a critical test, in the sense that it compares MC-LAD with at least one state-of-the-art algorithm (C4.5). It allows the competitiveness of MC-LAD to be evaluated in hard conditions, also because of the quantity and diversity of the data sets employed.

We present no results for the TABATA algorithm on the `adult` data set, because in this case the algorithm has not been able to generate a model in acceptable time (one week).

Classification accuracy

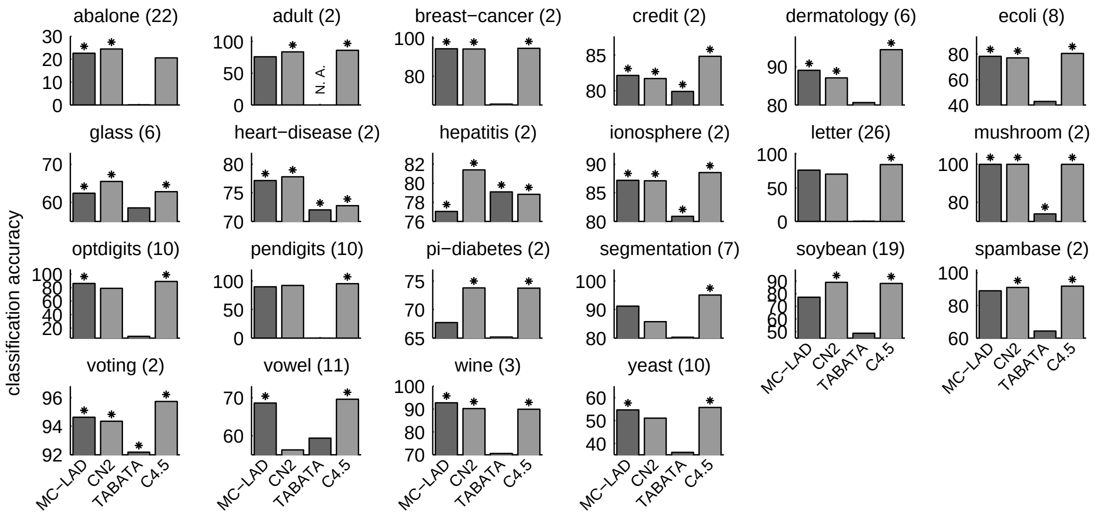


Figure 5.6: Classification accuracy of MC-LAD compared to alternative approaches. Higher bars are better. Table 5.6 on page 110 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value (or group of values), with statistical significance at a confidence level of 0.95. (N.A. means not available).

According to the significance test, MC-LAD is among the best algorithms in fifteen out of twenty-two data sets (Fig. 5.6). In four other problems, it is worse than the best method by a difference of 3%–6%, and in three others by 8%–10%. It is the best overall in one of the data sets (`wine`) but the difference is not significant.

CN2 is among the best sixteen times out of twenty-two, and it is the best overall in one of the problems (`abalone`). In two of the problems it is worse than the best method by 3%–4%, in two others by 10%, and in two others by 13%–14%.

TABATA shows a very poor performance overall. Statistically, it is no worse than the rest in six of the data sets, but it is always the method with the lowest accuracy, except in two problems (`hepatitis` and `vowel`). However, the difference from the best method attains in some cases nearly 40%, and it is useless in at least five tasks, where its performance is (near-) null. Later on page 103 we try to provide a reason for the observed phenomenon.

C4.5 looses once against MC-LAD and CN2 (in `abalone`), otherwise it is the best, or among the best, in the other twenty-one cases.

Number of patterns/rules

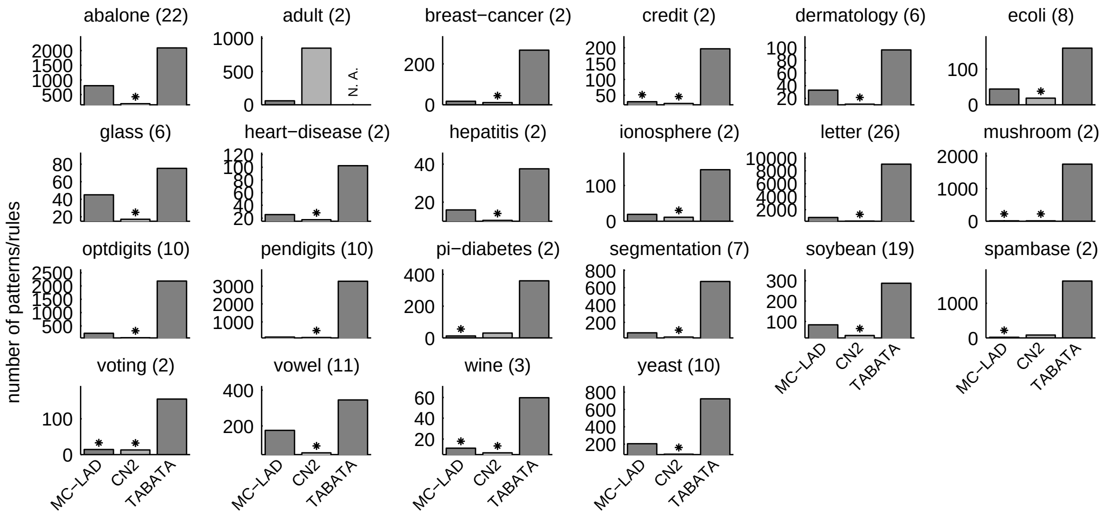


Figure 5.7: Number of patterns in the model created by MC-LAD compared to the number of rules generated by alternative approaches. Lower bars are better. Table 5.7 on page 110 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value (or group of values), with statistical significance at a confidence level of 0.95. (N.A. means not available).

According to Sect. 2.4 on page 27, we consider a rule as the equivalent of a pattern, and consequently they are counted equally. Figure 5.7 allows two main conclusions to be drawn: one of them is that CN2 and MC-LAD alternately present the lowest number of patterns, although in general CN2 is better (generates less rules). The other conclusion is that TABATA creates much larger rule sets than the other two, even by more than an order of magnitude in some cases.

Execution time

We have artificially set a lower bound (of one second) on the presented results concerning the execution time, although in some cases the effective time was below the bound. Figure 5.8 presents C4.5 as the clear winner. It is always the best or among the best, and it never takes more than half a minute to solve any of the treated problems. The execution times of the other three algorithms scale differently with the size of the problem. The execution time of CN2 scales better than that of MC-LAD, which in turn scales better than that of TABATA. Concerning our proposed procedure in particular (MC-LAD), it takes almost five hours to train a model for the `letter` data set, and this is the task that it takes longer to solve, among all.

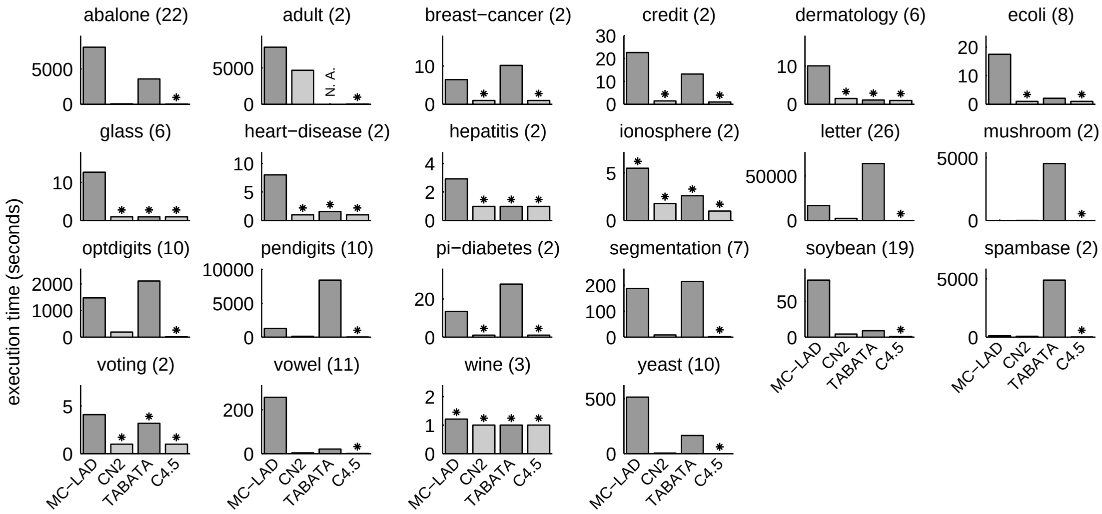


Figure 5.8: Execution time of MC-LAD compared to alternative approaches. Lower bars are better. Table 5.8 on page 111 contains equivalent results in numerical format. The $*$'s placed over the bars indicate the best value (or group of values), with statistical significance at a confidence level of 0.95. (N.A. means not available).

A note about the TABATA algorithm

It can be observed that the results of TABATA are quite deceiving in general. For the larger problems, the accuracy attains 0%, which is a rather strange result. The outer loop of TABATA is quite similar to that of CN2, which has been shown to be quite competitive. Hence, the poor results can only be due to the specific rule induction engine. We have investigated the cause of this problem, and found that the algorithm generates many rules that cover a single training instance. For the larger problems, it ends up by bluntly generating one rule for each example, covering only that example (as seen in the results of Fig. 5.7 and on Table 5.7 on page 110, which can be compared to the tables in Appendix A). This creates irrelevant classification models for the tasks at hand, which explains the very poor results.

TABATA is an example of a method which has never been developed beyond the phase of prototype. It has been tested by its authors on very small data sets, and it can be seen here that its performance scales very poorly with the size of the problem, in all respects: accuracy, model size, and execution time. As we have mentioned earlier, TABATA employs a Tabu search approach for searching patterns (rules) in the data, which is also the case of MC-LAD, and that is the reason why we have included it in the current experiments. The next chapter shows in more detail how do the two approaches differ in this respect, and what can possibly justify the observed differences.

5.4 Model interpretability

Earlier in this dissertation, we have stated that one of the interesting characteristics of logic-based methods is their ability to generate classification models that are interpretable by human beings. According to the results presented in Sect. 5.3 (we limit our analysis to the case of MC-LAD) the number of patterns contained in the generated models is in some cases very large (see Table. 5.7 on page 110). This seems to indicate that the models are too complex to be interpreted. If a domain expert is able to understand quite clearly the meaning of a single pattern, he/she may have more trouble finding meaningful information in a model containing several tens or hundreds of them.

We have decided to analyze the structure of the models created by MC-LAD for the tested tasks. For that, we have generated a set of histograms, one for each data set, showing the number of patterns in the model that cover a certain ratio of examples of one class. As the coverage rate has values in the range $[0, 1]$, we have created histograms containing ten bins of width 0.1 each (thus containing the intervals 0–0.1, 0.1–0.2, ... , 0.9–1.0). In the multi-class case, each pattern in the model may be counted several times in the histogram, since most patterns cover examples of more than one class. Given that the experiments reported in Sect. 5.3 have been done in a 5×2cv manner, the current analysis is based on the model generated for the first fold of the first replication. The results are shown in Fig. 5.9 on the facing page.

It can be seen that most of the patterns have low coverage, nevertheless some of them have very high coverage. A pattern covering more than half of the data of a class is a strong indicator of the distribution of the instances of that class, and the results of Fig. 5.9 show that certain patterns have a coverage rate of more than 90%.

From these results we may conclude that with respect to interpretability, the domain expert must do a selective usage of the model, choosing the patterns that provide the most meaningful information. To illustrate this, below we show examples of conclusions that can be drawn from some of the tested tasks, and that correspond to real situations.

The `adult` data set

The `adult` data set originates in the US Census Bureau and the aim is to predict whether an individual earns less than \$50 000 a year. The training data contains the description of 24 421 US citizens based on attributes like age, level of education, or native country, of which 18 578 are positive (they earn less than \$50 000) and 5 842 are negative. One of the patterns in the model is the following:

CLASS COVERAGE RATE		
salary $\geq$ \$50 000	salary $<$ \$50 000	PATTERN
-1	0.3148	$\overline{b}_{17} b_{107} \overline{b}_{128}$

This pattern tells us that 31.48% of the individuals earning less than \$50 000 a year are younger than 43 years old ($\overline{b}_{17}$), have never been married (b_{107}), and work less than 41 hours per week ($\overline{b}_{128}$). As this data set is large, the pattern applies to 5 848 individuals.

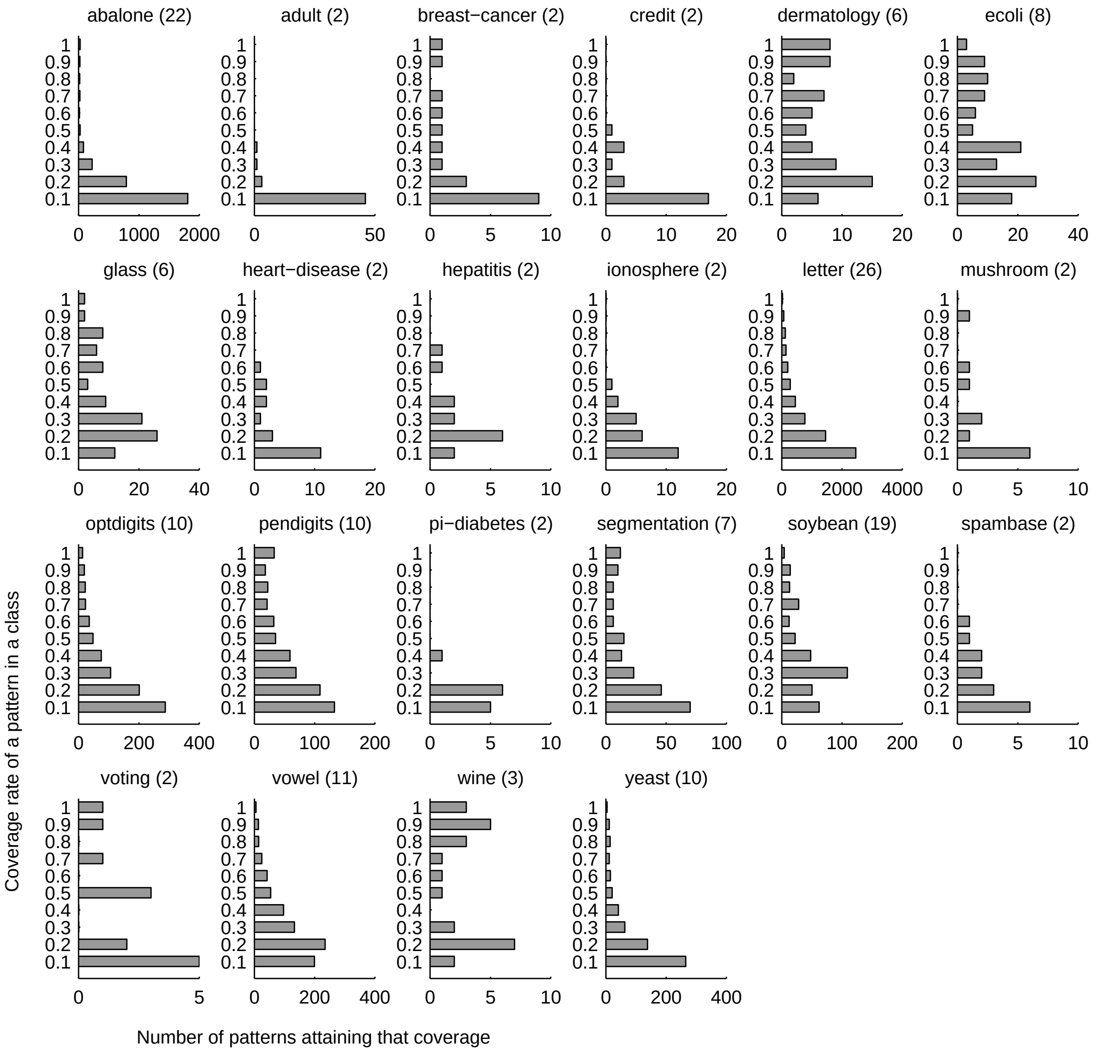


Figure 5.9: Histograms showing the distribution of pattern coverage rates.

The `spambase` data set

The `spambase` data set has been collected by a group of researchers working at the Hewlett-Packard Labs, in Palo Alto, California. It contains electronic mail messages received by the donors, which have been divided into two classes: those containing *spam* on one side (non-requested advertising), and their own regular messages on the other. The goal is to predict whether a new message has spam. We have selected the following two patterns from the model:

CLASS COVERAGE RATE		
spam	regular	PATTERN
0.2367	−1	$\overline{b}_4\overline{b}_7 b_{20}\overline{b}_{27}$
−1	0.5806	$\overline{b}_{15}\overline{b}_{21} b_{27} b_{28}$

The pattern $\overline{b}_4\overline{b}_7 b_{20}\overline{b}_{27}$ indicates that 23.67% of the spam messages contained dollar signs as well as the words "remove" and "free", but did not contain the word "re".

From the second pattern ($\overline{b}_{15}\overline{b}_{21} b_{27} b_{28}$) we know that 58.06% of the regular messages contained the words "HP" (the data set was collected at Hewlett-Packard Labs) and "edu", but did not contain dollar signs nor uninterrupted sequences of capital letters.

The `dermatology` data set

Above we have discussed two-class tasks, but a similar analysis can be done in multi-class contexts. In the `dermatology` data set the aim is to determine which one among six types of Erythemato-Squamous disease (a skin disease) does a patient suffer from. We have extracted the following three patterns from the model:

CLASS COVERAGE RATE						
psoriasis	seboreic dermatitis	lichen planus	pityriasis rosea	chronic dermatitis	pityriasis rubra pilaris	PATTERN
0.5179	-1	0.7500	0.9600	-1	-1	b_2
0.8214	-1	-1	-1	0.0769	0.9000	b_3
-1	0.9000	1.0000	1.0000	0.0375	1.0000	$\overline{b}_8$

The pattern b_2 indicates that the koebner phenomenon affects 75% of the patients suffering from lichen planus and 96% of those suffering from pityriasis rosea, but none of those suffering from seboreic dermatitis, cronic dermatitis, or pityriasis rubra pilaris.

The second pattern (b_3) indicates that 82.14% of the patients suffering from psoriasis and 90% of those suffering from pityriasis rubra pilaris present symptoms of knee and elbow involvement; moreover, theses symptoms are not observed in any of the patients suffering from seboreic dermatitis, lichen planus, or pityriasis rosea.

From the pattern $\overline{b}_8$ we learn that more than 90% of the patients suffering from seboreic dermatitis, lichen planus, pityriasis rosea, and pityriasis rubra pilaris do not show an elongation of the rete ridges.

For models containing several tens or hundreds of patterns, it is not feasible to understand the entire classification process, even for experts on the treated task. However, the above examples show that these models may provide powerful tools allowing experts to develop additional insight of the domain or to confirm their personal conclusions, under the condition that certain meaningful patterns are generated (with large coverage).

Non-intuitive domains

Obviously, the interest of this kind of analysis is higher in domains where the attributes have an intuitive interpretation. For example, the task in the `segmentation` data set collected by the Vision Group of the University of Massachusetts consists in identifying the contents of seven outdoor images, based on very small sub-images of 3×3 pixels extracted randomly therefrom. Each one of the seven images contains one of the following subjects: brick face, sky, foliage, cement, window, path, or grass. These images are themselves the classes, and the goal is to predict, given a sub-image, to which one of the seven original images it belongs to. Consider the following pattern, included in the model:

CLASS COVERAGE RATE							
brick face	sky	foliage	cement	window	path	grass	PATTERN
0.9879	−1	0.9394	0.1636	0.9818	−1	1.0000	$\bar{b}_{17}$

The binary attribute b_{17} is false when the average of the red value (in the RGB color code) over the 3×3-pixel sub-image is below 24.77. It can be observed that this holds for almost 100% of the sub-images of brick face, foliage, window, and grass, but never for the sub-images of sky and path. It is clear that this observation is less effective for humans to acquire additional knowledge of the treated domain. It relates above all to physical measurements that have a "hidden" meaning. They can, however, be very helpful for the designer of a classifier, allowing him to choose the most significant attributes. These comments apply to the domain of computer vision in general, but also to other domains like automatic speech recognition, for example.

5.5 Discussion

In the current chapter, we have assembled several principles exposed in other chapters of this dissertation in order to create a multi-class version of the LAD algorithm. We have made two types of empirical comparisons for evaluating the worthiness of the proposed approach. One of them concerned the previously existing version of LAD on which the new algorithm has been partially inspired. Naturally, the comparison has been limited to two-class problems. In this respect, we consider the approach to have succeeded, which we justify with the following facts: on the one hand, for comparable classification accuracy, MC-LAD can in some cases generate much simpler models, containing less patterns. On the other hand, we have shown that classification problems of moderate to large size cannot be handled by the standard version of LAD in acceptable computational time.

Concerning the other comparison, done in a more general context, we admit that MC-LAD presents no major improvement in terms of classification performance. It has proven fairly competitive in many problems, but it never significantly outruns the best method. Its worthiness lies perhaps in the novelty of the treatment given to the patterns, which are shared among all the classes in a "pertinent" manner (we discuss this in more detail below). This approach has been capable of generating less patterns than the alternative CN2 in some problems, but the opposite result has more often been obtained. We would have expected that the pattern-sharing effect would lead MC-LAD to create a lower number of patterns in general. However, the procedure MC-LAD relies on a duality of coverage, where each pair of classes must be separated by two pattern sets. This may justify the generation of a larger number of patterns by MC-LAD than the number of rules generated by CN2 in some cases. Concerning the execution time, MC-LAD has been much slower than CN2 or C4.5, which is an additional handicap. Another important observation that is provided by the presented results is the excellent ratio between high classification accuracy and short training time attained by C4.5, which only confirms many previous equivalent conclusions found in the Machine Learning literature.

A limitation of the proposed MC-LAD algorithm is the fact that is does not contain a flexible mechanism for controlling the capacity of the generated model. The capacity is limited by certain user-defined parameters, like the maximal degree of a pattern, or the minimal differentiability rate, which help preventing the algorithm to closely fit the training data. However, these parameters have been set uniformly, despite the fact that the chosen values may possibly be inappropriate for some classification tasks. It would be desirable to have a mechanism that adapts the algorithm to the conditions of each treated domain. One solution for this problem might consist in allowing the model to be as flexible as possible and using validation techniques (cf. Sect. 1.3) to stop the training whenever the performance on the validation set starts decreasing. This option has the drawback of demanding additional computational effort and a large amount of data.

We have also analyzed the interpretability of the generated models, which may be considered the main justification for using this type of learning algorithm. In this regard, we point out that the sharing of patterns among various classes that is done by MC-LAD generates conclusions which are different from those of the one-class-against-all-others type of rules. The latter define patterns of behavior which apply to a single class. In our case, the generated patterns apply to several classes at the same time. This has as intuitive justification and we can easily conceive it as corresponding to real-life situations, as shown by the examples of Sect. 5.4 for the `dermatology` data set. Without stating that one of these approaches is better than the other, the latter (MC-LAD) is undoubtedly more general and also more flexible. But it should be up to the user to decide which kind of analysis is more appropriate, depending on the situation.

	CLASSIFICATION ACCURACY	
DATA SET	STD-LAD	MC-LAD
breast-cancer	**94.76** ±0.92	**94.48** ±1.34
credit	**83.07** ±2.20	**82.14** ±2.34
heart-disease	**76.04** ±3.74	**77.16** ±2.93
hepatitis	**77.55** ±3.71	**77.04** ±4.12
ionosphere	**85.13** ±2.95	**87.18** ±1.89
mushroom	**100.00** ±0.00	**100.00** ±0.00
pi-diabetes	**68.62** ±2.18	**67.73** ±2.09
spambase	**90.73** ±1.03	88.95 ±0.96
voting	**94.48** ±1.91	**94.62** ±1.34

Table 5.3: Classification accuracy of MC-LAD compared to standard LAD. The numbers give the average ± standard deviation calculated over a 5×2cv set of experiments, and higher values are better. The numbers typed in bold indicate the best value of a row, with statistical significance at a confidence level of 0.95.

	NUMBER OF PATTERNS	
DATA SET	STD-LAD	MC-LAD
breast-cancer	**18.00** ± 1.10	**17.80** ± 1.94
credit	**47.40** ± 4.50	**29.70** ±13.00
heart-disease	**27.50** ± 3.35	**25.40** ± 3.23
hepatitis	**15.30** ± 2.45	**15.90** ± 2.21
ionosphere	21.00 ± 3.87	**19.00** ± 4.02
mushroom	**11.00** ± 0.00	12.10 ± 0.30
pi-diabetes	75.20 ± 4.26	**12.10** ± 2.21
spambase	147.70 ±13.80	**19.60** ± 3.72
voting	**13.70** ± 2.05	**14.00** ± 1.73

Table 5.4: Number of patterns generated by MC-LAD and by standard LAD. The numbers give the average ± standard deviation calculated over a 5×2cv set of experiments, and lower values are better. The numbers typed in bold indicate the best value of a row, with statistical significance at a confidence level of 0.95.

	EXECUTION TIME (SECONDS)	
DATA SET	STD-LAD	MC-LAD
breast-cancer	**1.16** ± 0.37	6.45 ± 0.95
credit	**18.32** ± 8.05	**22.63** ± 9.17
heart-disease	**1.12** ± 0.26	8.01 ± 1.50
hepatitis	**1.00** ± 0.00	2.92 ± 0.49
ionosphere	**1.24** ± 0.52	5.50 ± 1.91
mushroom	**1.00** ± 0.00	1.40 ± 0.04
pi-diabetes	**32.31** ± 17.12	**13.52** ± 2.42
spambase	2118.45 ±676.40	**106.62** ±20.64
voting	**1.04** ± 0.12	4.09 ± 0.98

Table 5.5: Execution time of the MC-LAD and of the standard LAD algorithms. The numbers give the average ± standard deviation calculated over a 5×2cv set of experiments, and lower values are better. The numbers typed in bold indicate the best value of a row, with statistical significance at a confidence level of 0.95.

DATA SET	MC-LAD	CN2	TABATA	C4.5
	CLASSIFICATION ACCURACY			
abalone	**22.63** ±0.58	**24.43** ±0.85	0.00 ± 0.00	20.57 ±1.19
adult	75.76 ±1.05	**83.65** ±2.08	N.A. ± —	**85.93** ±0.27
breast-cancer	**94.48** ±1.34	**94.36** ±0.82	65.52 ± 0.09	**94.71** ±0.89
credit	**82.14** ±2.34	**81.71** ±1.35	**79.91** ± 3.18	**84.81** ±1.92
dermatology	**89.07** ±2.84	**87.10** ±2.57	80.66 ± 4.57	**94.48** ±2.69
ecoli	**78.34** ±3.40	**77.03** ±3.42	42.74 ± 0.44	**80.59** ±4.14
glass	**62.41** ±5.88	**65.47** ±5.25	58.57 ± 5.64	**62.80** ±4.43
heart-disease	**77.16** ±2.93	**77.82** ±2.26	**72.01** ± 5.66	**72.74** ±1.71
hepatitis	**77.04** ±4.12	**81.41** ±3.27	**79.11** ± 2.46	**78.85** ±4.22
ionosphere	**87.18** ±1.89	**87.12** ±2.30	**80.91** ± 3.00	**88.54** ±3.28
letter	76.32 ±0.73	70.26 ±0.50	0.00 ± 0.00	**84.60** ±0.58
mushroom	**100.00** ±0.00	**100.00** ±0.00	**73.95** ±24.67	**100.00** ±0.00
optdigits	**86.10** ±1.30	79.01 ±1.22	7.79 ±23.38	**89.03** ±0.61
pendigits	89.54 ±0.91	92.16 ±0.38	0.00 ± 0.00	**95.32** ±0.45
pi-diabetes	67.73 ±2.09	**73.78** ±1.18	65.18 ± 0.17	**73.75** ±2.27
segmentation	91.21 ±1.34	85.73 ±1.05	80.32 ± 6.58	**95.09** ±0.61
soybean	77.16 ±3.04	**88.82** ±1.17	48.78 ±10.94	**87.97** ±2.01
spambase	88.95 ±0.96	**91.01** ±0.62	64.36 ± 1.27	**91.75** ±0.68
voting	**94.62** ±1.34	**94.34** ±1.21	**92.18** ± 2.09	**95.73** ±1.14
vowel	**68.57** ±3.52	56.30 ±1.91	59.37 ± 4.39	**69.56** ±2.85
wine	**92.70** ±2.54	**90.23** ±2.71	70.56 ±10.41	**89.90** ±3.11
yeast	**54.61** ±1.64	51.01 ±1.67	36.08 ± 2.52	**55.67** ±1.79

Table 5.6: Classification accuracy of MC-LAD compared to alternative approaches. The numbers give the average ± standard deviation calculated over a 5×2cv set of experiments, and higher values are better. The numbers typed in bold indicate the best value (or group of values) of a row, with statistical significance at a confidence level of 0.95. (N.A. means not available).

DATA SET	MC-LAD	CN2	TABATA
	NUMBER OF PATTERNS/RULES		
abalone	799.60 ±22.99	**190.60** ± 6.61	2087.00 ± 2.76
adult	**61.40** ±11.05	853.20 ±93.53	N.A. ± —
breast-cancer	17.80 ± 1.94	**11.40** ± 1.02	267.40 ± 18.43
credit	**29.70** ±13.00	**24.40** ± 2.80	196.00 ± 19.48
dermatology	33.00 ± 6.02	**10.90** ± 1.37	96.50 ± 12.39
ecoli	44.30 ± 3.93	**18.50** ± 1.36	158.30 ± 4.56
glass	45.10 ± 5.97	**17.20** ± 0.87	75.10 ± 10.89
heart-disease	25.40 ± 3.23	**17.60** ± 2.06	102.30 ± 5.80
hepatitis	15.90 ± 2.21	**10.50** ± 1.02	37.50 ± 3.91
ionosphere	19.00 ± 4.02	**11.30** ± 1.10	144.20 ± 25.59
letter	784.80 ±30.21	**217.20** ± 5.79	9042.10 ±229.71
mushroom	**12.10** ± 0.30	**13.40** ± 0.49	1746.50 ±582.26
optdigits	224.20 ±14.87	**61.90** ± 2.95	2184.00 ±110.25
pendigits	152.00 ± 6.99	**127.40** ± 3.95	3264.80 ±203.42
pi-diabetes	**12.10** ± 2.21	30.90 ± 3.94	359.10 ± 8.63
segmentation	79.60 ± 8.79	**31.10** ± 1.97	670.00 ±113.55
soybean	84.00 ±13.01	**32.40** ± 1.91	287.00 ± 37.05
spambase	**19.60** ± 3.72	80.10 ± 1.70	1637.70 ±111.16
voting	**14.00** ± 1.73	**12.90** ± 1.22	155.10 ± 49.40
vowel	176.00 ±12.70	**49.80** ± 1.94	345.90 ± 26.66
wine	**11.30** ± 2.65	**6.80** ± 0.75	59.70 ± 4.12
yeast	204.50 ±20.92	**84.00** ± 4.36	722.30 ± 7.68

Table 5.7: Number of patterns in the model created by MC-LAD compared to the number of rules generated by alternative approaches. The numbers give the average ± standard deviation calculated over a 5×2cv set of experiments, and lower values are better. The numbers typed in bold indicate the best value (or group of values) of a row, with statistical significance at a confidence level of 0.95. (N.A. means not available).

	EXECUTION TIME (SECONDS)			
DATA SET	MC-LAD	CN2	TABATA	C4.5
abalone	8096.84 ± 193.25	47.30 ± 1.42	3570.70 ± 111.78	**4.00** ±0.00
adult	7844.74 ±1477.50	4663.20 ±371.88	N.A. ± —	**26.20** ±0.98
breast-cancer	6.45 ± 0.95	**1.00** ± 0.00	10.20 ± 0.87	**1.00** ±0.00
credit	22.63 ± 9.17	**1.50** ± 0.50	13.20 ± 1.60	**1.00** ±0.00
dermatology	10.08 ± 2.98	**1.50** ± 0.50	**1.10** ± 0.30	**1.00** ±0.00
ecoli	17.51 ± 3.07	**1.00** ± 0.00	2.10 ± 0.30	**1.00** ±0.00
glass	12.68 ± 2.52	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ±0.00
heart-disease	8.01 ± 1.50	**1.00** ± 0.00	**1.60** ± 0.49	**1.00** ±0.00
hepatitis	2.92 ± 0.49	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ±0.00
ionosphere	**5.50** ± 1.91	**1.80** ± 0.60	**2.60** ± 0.80	**1.00** ±0.00
letter	17011.15 ± 570.80	2574.90 ± 86.63	63803.40 ±4327.22	**13.20** ±0.60
mushroom	1.40 ± 0.04	11.30 ± 0.46	4547.30 ±1395.61	**1.00** ±0.00
optdigits	1480.63 ± 100.53	188.90 ± 7.84	2111.30 ± 333.32	**6.30** ±0.46
pendigits	1271.30 ± 62.85	134.20 ± 3.06	8441.60 ± 734.62	**5.00** ±0.00
pi-diabetes	13.52 ± 2.42	**1.00** ± 0.00	27.90 ± 1.51	**1.00** ±0.00
segmentation	187.05 ± 43.06	8.80 ± 0.60	214.70 ± 39.23	**1.40** ±0.49
soybean	80.00 ± 11.40	4.40 ± 0.49	9.10 ± 1.87	**1.00** ±0.00
spambase	106.62 ± 20.64	72.20 ± 4.42	4887.00 ± 864.47	**5.20** ±0.40
voting	4.09 ± 0.98	**1.00** ± 0.00	**3.20** ± 1.40	**1.00** ±0.00
vowel	256.24 ± 75.52	3.70 ± 0.46	20.70 ± 2.19	**1.00** ±0.00
wine	**1.21** ± 0.62	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ±0.00
yeast	515.95 ± 50.79	5.80 ± 0.40	166.30 ± 10.24	**1.00** ±0.00

Table 5.8: Execution time of the MC-LAD and of the standard LAD algorithms. The numbers give the average ± standard deviation calculated over a 5×2cv set of experiments, and lower values are better. The numbers typed in bold indicate the best value (or group of values) of a row, with statistical significance at a confidence level of 0.95. (N.A. means not available).

Chapter 6

Single pattern generation

Motivation

This chapter is closely related to the previous one, and it is dedicated to the problem of generating a pattern. This is motivated by the procedure **MultiClassLAD** defined in Sect. 5.2.3, which builds a classification model based on a set of patterns in a multi-class context. We recall that it is an iterative procedure that, at each iteration, defines a particular pattern generation task and calls a procedure **GeneratePattern** to solve it. The latter is defined here, and its goal is to generate a *good* pattern given a set of binary data partitioned into two subsets (positive and negative) received by the high-level **MultiClassLAD** procedure. We will precisely define the notion of goodness of a pattern.

In the following, Sect. 6.1 presents the problem by introducing the notion of combinatorial optimization, loosely explaining how it is usually treated, and formulating the pattern generation task in that framework. Section 6.2 presents *Tabu search*, which is a common approach for combinatorial optimization, and the one that has been chosen for the current task. The details of the implementation are given in Sect. 6.3, which also contains results and their discussion. Section 6.4 presents related work.

6.1 Pattern generation as combinatorial optimization

6.1.1 Heuristics for combinatorial optimization

Solving an optimization problem involves finding the optimal values for a set of decision variables. Certain optimization problems have a set of *constraints* associated, which impose limits on the range of values that each decision variable can take. The optimization process is guided by an *objective function* that calculates a quality measure for each assignment of the variables. In constrained optimization the goal is thus to find the set of values (*solution*) that optimizes (minimizes or maximizes, depending on the problem instance) the objective function without violating the constraints.

Combinatorial optimization concerns problems in which all the decision variables are discrete. In this context, the set of solutions is also discrete, each corresponding to one among all the possible combinations of values that can be assigned to the variables. Among all the solutions in the complete *solution space*, those that satisfy the constraints are called *feasible*, and among these, those that optimize the objective function are the *optimal solutions*. There can be more than one optimal solution for certain problems. An introduction and overview of the current state-of-the-art in combinatorial optimization techniques and their applicability can be found in a book by Cook et al. (1998).

Problems of small size can be solved by *exhaustive search*, i.e. by visiting the entire solution space and choosing the optimal solution. However, the combinatorial character of this space makes that it is very large for nearly all interesting real-life problems. In situations of this kind, when exact approaches leading to an optimal solution are infeasible, the use of *heuristics* is a normal practice. Reeves (1996) defines heuristic as "a method which seeks good (i.e near-optimal) solutions at a reasonable computational cost without being able to guarantee optimality, and possibly not feasibility. Unfortunately, it may not even be possible to state how close to optimality a particular heuristic solution is."

Besides the apparent pessimism of the above definition, research in the last decades has developed powerful heuristic search methods for combinatorial optimization. One important such class of heuristics includes those based on *local search*, which can roughly be described as procedures that start from an initial solution, and iteratively visit "neighbor" solutions, thus forming a path in the quest for an optimum. In this chapter, the pattern generation task is presented and solved in a combinatorial optimization setting using *Tabu search*, a well-known local search heuristic.

6.1.2 Finding an optimal pattern

Earlier in Sect. 2.3.4 on page 21, we have analyzed a search space of terms, and how it can be structured according to a partial order relation (more-specific-than). In the current context, the same search space is used for finding a pattern. Section 2.3.5 on page 22 explained the procedure employed by LAD for searching a set of patterns, and it has been mentioned therein that the entire search space can be very large, even for problems of moderate size. We recall that for a problem containing B binary attributes, the number of terms of degree G that can be formed is equal to $2^G\binom{B}{G}$. This quantity has been plotted in Fig. 2.4 on page 23 for values of G up to five and of B up to 150. In standard LAD the effective search space is a subset of the entire space, containing only the terms up to a certain degree, which is used to limit the computational time, but which serves also as *inductive bias*, i.e. as a form of limiting the capacity of the generated classification model and avoiding it to overfit the training data (cf. Sect. 1.3).

In the procedure that will be described hereafter, the search space is also restricted, for the same reasons as above, to terms up to a degree G_{max} whose value will be specified later. However, this limitation does not suffice to allow exhaustive search to be applied, except in very specific cases. Hence, the task is approached with a heuristic search method. *Tabu search*

(TS) is one such promising method with a proven record of successful applications. With background in the 1970's, the first actual formalization of the method was made by Glover (1986), although similar ideas were developed independently by Hansen (1986) in the same period. A comprehensive, up-to-date reference on TS is given by Glover and Laguna (1997). Interesting introductory material with examples of applications can be found in articles by Glover (1989), Hertz and de Werra (1990), and Hertz et al. (1995). The following section describes the TS method in general, which aims at preparing the description of the implementation of the pattern-finding procedure.

6.2 The Tabu search meta-heuristic

6.2.1 What is Tabu search

Local search techniques, be they simple or elaborate, require the definition of a *neighborhood* which, for any solution of the problem, establishes the set of its *neighbor* solutions. The search process is driven through different solutions via *moves*, where each move allows to pass from a solution to one of its neighbors. So, given a set S of feasible solutions, a neighborhood $H(s) \subset S$ associated with every solution $s \in S$, and a real-valued objective function $O(s)$, doing local search (alternatively named *steepest descent* or *hill-climbing*) consists in choosing an initial feasible solution, moving to its best neighbor, provided that it improves the objective function, and continuing to move in this manner until no better neighbor can be found. Such an approach converges to the best local solution in a space whose topology is defined by both the chosen objective function and the neighborhood. However, the solution thus found can be largely suboptimal, unless all the local optima contained in the search space are also global optima, which is rarely or never the case in any application of interest. Contrary to the standard descent method, TS allows non-improving moves, that is, a move to the best available neighbor is made even if it deteriorates the objective function. This aims at overstepping the bounds of the local region and seeking better solutions in the remaining search space. However, moving to non-improving neighbors can result in cycles in the search process, and this is where the key feature of Tabu search plays a role — the use of *memory*, that records the recent search path, thus avoiding stepping into previously (recently) visited solutions by declaring them to be "tabu".

Despite the comparison with the steepest descent heuristic, TS is in fact a *meta-heuristic* method, in the sense that it guides a high-level exploration of the regions in the search space using a set of heuristics, and at each step applies locally a lower-level heuristic suitable to the particular problem treated. Examples of other common meta-heuristic methods are *Simulated Annealing* (SA)[1] (Kirkpatrick et al., 1983; Aarts and Korst, 1989) and *Genetic Algorithms* (GA)[2] (Holland, 1975; Goldberg, 1989).

Tabu search has been showing an exceptional solution-finding ability. A recent compilation of significant achievements in a large variety of domains can be found in Chap. 8 of the book by Glover and Laguna (1997). The authors summarize a list of publications by several

[1] A brief description of SA is given in Sect. 6.4 along with that of ATRIS, a method for rule induction (Kononenko and Kovačič, 1992).

[2] Section 2.4.4 on page 32 discusses the application of genetic algorithms for generating classification models.

researchers reporting successful applications of TS in various domains along the last decade. That summary provides evidence of the capacity of the method for finding high quality solutions within short computation times, as well as for handling higher levels of complexity than those previously judged feasible in many of the treated domains.

6.2.2 Basic features

We first provide a sketch of the basic Tabu search algorithm below. In the remainder of this section we explain in more detail each one of its most important features.

1. set the iteration counter i to 0;
2. choose an initial solution s_i;
3. generate a neighborhood H of s_i containing no tabu solutions;
4. increment the counter i ;
5. set s_i as the best neighbor in H ;
6. if s_i is better than the best solution, then the best solution becomes s_i;
7. update the tabu list with s_i;
8. if the stopping criterion is satisfied, stop;
9. go to 3.

Memory and dynamic neighborhood

In its simplest form, memory is implemented by a *tabu list* T of fixed size. The list is used in the form of a queue, that is, whenever a solution is visited, it is added to the tail of the list, while the oldest element (the head) is removed. As will be discussed below, this is not the only possibility, as in many implementations the tabu list is used to store moves between solutions, instead of the solutions themselves. The size $|T|$ of the tabu list is called the *tabu tenure*, and given that the $|T|$ most recently visited solutions are recorded, that number constitutes a lower bound on the length of the avoided search cycles. The most effective tabu tenure is problem-dependent, though normally of easy tuning with little experimentation (Glover and Laguna, 1997, Chap. 2). In certain applications, the use of variable-sized tabu lists can convey interesting improvements. Different approaches are possible for the implementation of this feature (cf. the above citation), and a specific application of it is discussed later in this section.

A neighborhood $H(s)$ consists of solutions that can be directly reached from s. In practice, solutions that are tabu are not included in $H(s)$. In addition there are good reasons to explore only a *sub-neighborhood* $H'(s)$ of each solution, instead of the complete set of its neighbors, because: (i) the latter can be computationally expensive, and (ii) the stochastic behavior induced by sub-neighborhood exploration can be very helpful to escape local optima. Selection strategies may be either based on random choice or using cleverer measures for selecting the best neighbors (Glover and Laguna, 1997, Chap. 3). Considering that the structure of the neighborhood is variable and depends on the iteration, TS can be considered a method of *dynamic neighborhood search*.

An additional consideration concerning memory: In standard local search, the best solution is always the current one, which is not the case in TS. Hence an independent memory element is used to record the best solution obtained so far.

Solutions versus moves

In any optimization problem, each solution represents a certain configuration of an entity that can assume different states. The optimal solution represents the best state, i.e. the one that optimizes a certain quality measure. The representation in computational form of each of those states can be considerably complex in many situations, which demands the use of partial representations. This can be illustrated by a simple example. Consider a graph representing a number of villages (the vertices) in an island, with road sections connecting them (the edges), as shown in Fig. 6.1 on the left. Each edge has a value associated: the distance in kilometers of the corresponding road section (not shown). Imagine a farmer living in village A (on the

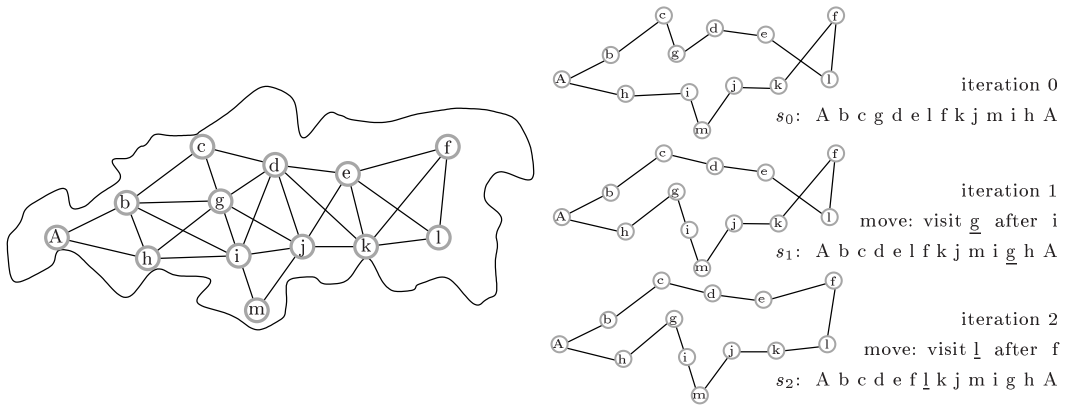


Figure 6.1: Graph representation of villages (vertices) connected by roads (edges). The goal is to find the shortest path starting and ending at A, and passing through every other village once. Using iterative local search, each move between consecutive solutions consists in displacing one of the villages along the path sequence.

extreme left) that has to deliver the product of his activity to grocery stores along the island. He has at least one client store in each village. In order to minimize his expense on fuel, he needs to calculate the shortest path allowing him to visit all the villages and return back home. If a constraint is added stating that no vertex can be visited more than once (except the initial node A), then this corresponds to a typical problem of combinatorial optimization, called the *Traveling Salesman Problem* (TSP).

In the TS framework, we can represent each solution by a sequence of vertices determining the order in which villages are visited. The objective function to be minimized is the sum of the distances of all the edges connecting the traversed vertices. In order to solve this problem, we start from an initial solution and modify it iteratively. Consider the initial solution s_0 to be the ordered set of edges represented at the top-right of the figure, with a certain value for the objective function, which we will not discuss. We determine the neighborhood $H(s_0)$ to be composed of all solutions obtained from s_0 by moving one vertex in the sequence, provided that the latter remains valid, i.e. the path visits all vertices once. In this small example, there are just a few possibles neighbors in $H(s_0)$. For example, we can visit c after g, or visit g after i, or visit l after f. Let us assume that the second (visit g after i) generates the best improvement, in which case it is adopted to form the next solution s_1, as indicated in the figure. Then we can proceed further and choose the best neighbor of s_1, and so on.

The goal of this example is to show that although each solution can be described exhaustively, there is a clear gain in restricting the representation to the variation between two consecutive solutions (the moves). In the current setting, s_1 could be simply represented in relation to s_0 by $\{\mathrm{g} \succ \mathrm{i}\}$ (which means that g succeeds i). All the other solutions in $H(s_0)$ could be represented in a similar short form. This reasoning extends to the tabu list, that must store $|T|$ different solutions. Also in this case it is a better choice to work in terms of moves. The use of this shorter form for describing neighborhoods and tabu lists has become a standard practice in TS applications. However, the full representation must still be used in order to record the best solution achieved so far.

Consider the notation $s' = s \oplus m$, which means that by applying the move m to solution s a new solution s' is obtained. In general, each move m is reversible and has an associated inverse move m^{-1}, such that $(s \oplus m) \oplus m^{-1} = s$. During the search process, whenever a solution s is reached, the applied move m, or its inverse m^{-1}, depending on the problem and the solution/move representation chosen, is inserted in the tabu list T. Likewise, the neighborhood is defined in terms of the set of moves that are applicable to a solution.

Aspiration condition

Using moves instead of solutions, as described above, implies a loss of information. As long as a move is defined relatively to a certain solution all the information is there but, taking the example above, when the move $\{$*remove* c$\}$ is stored in the tabu list, the context in which it has been defined is lost. This has two main implications: on the one hand, it can no longer be guaranteed that a solution which has recently been visited will be excluded from the current neighborhood, and on the other hand, certain non-visited adjacent solutions may be accessible only through tabu moves. The latter problem is solved through the use of *aspiration conditions*. This kind of mechanism can hold different levels of complexity, and the main principle is to establish a condition or a set of them which, if satisfied by a tabu move, makes it eligible for the current neighborhood despite its tabu status. In most cases, the aspiration condition determines that if a solution that is accessible through a tabu move is better than the best visited solution so far, then the tabu status of that move is overridden and the solution is accepted.

Stopping criteria

The stopping criterion for the standard steepest descent algorithm is quite straightforward: stopping when no further improvement is possible. In the case of TS this condition is not verifiable. There are different alternatives for establishing a stopping criterion for TS, and the choice of the most appropriate one depends on the specific problem and the formulation applied. Common approaches are: stopping when a predefined maximum allowed number of iterations is attained, either since the first iteration or since the last iteration where an improvement was verified; stopping if an acceptable solution has been found, according to some acceptability measure; stopping when evidence exists that no better improvement can ever be achieved; and also, stopping if the feasible neighborhood for the current solution is empty, which can happen if the tabu tenure exceeds the size of the neighborhood.

Intensification and diversification

This feature stands out of the normally accepted set of "basic" TS features, although mention is done here to prepare the discussion of the actual TS implementation for pattern generation in the next section. In Sect. 6.2.1, TS has been described as a meta-heuristic driving a high-level search process along regions of the solution space, and locally applying subordinate problem-dependent heuristics. One kind of high-level guidance consists in introducing alternating phases of intensification and diversification in the search process. Intensification compels the search to concentrate on a promising region where attractive solutions were found, while diversification, on the contrary, drives the search to not yet explored regions as a means of avoiding the same set of solutions already visited and thus stimulating the search process. Diversification can facilitate the access to regions that would be hardly accessible using a regular path.

One possible way of implementing this alternation is by introducing elements in the objective function that either favor or penalize solutions which are close to the recently visited ones. Another possibility is that of dynamically modifying the tabu tenure — short tabu tenures for intensification and longer ones for diversification. Restarting is also a way of diversifying the search.

6.2.3 Efficiency issues

Effective modeling

The success of TS critically depends on the modeling phase, namely in the design of the neighborhood H and the choice of the objective function, two decisive factors conditioning the topology of the search space. A suitable topology can accelerate the search process and facilitate the exploration of regions with interesting solutions, while inappropriate modelings may dramatically increase the number of iterations needed to find good solutions or even hinder the method from accessing the most promising regions. Namely, the existence of a path leading from any visited solution to the most interesting solutions must be guaranteed.

Long plateaus as well as steep ridges can be hard to traverse. For this reason, integrating the constraints of the problem as such into the formulation can be too restrictive and severely limit the mobility of the search. A better approach consists in allowing the constraints to be violated, and penalizing such violations in the objective function. In addition, by incorporating elements in the objective function that change with time, it is possible to implement a dynamic topology that renders hard-traversing regions temporarily more accessible.

As noted by Hertz et al. (1995), an exhaustive parameter-tuning effort can hardly compensate for incorrect modeling while, on the contrary, well-suited modeling can considerably reduce the importance of the parameter settings for the success of the method.

Computational efficiency

Another important factor that must be taken into account for a proper TS implementation concerns computational efficiency. Calculating the objective function is an operation executed with high frequency in the course of a TS optimization run, hence special care must be taken in its implementation in order to reduce as much as possible its execution time. This is a problem-dependent exercise, nevertheless a widely applicable advice can be given: in many problems, the objective function is best calculated incrementally, that is, when the neighborhood of a solution s is being analyzed, the value of the objective function for any of its neighbor solutions $s' \in H(s)$ can be calculated based on the differential between the two solutions, meaning the modification introduced by the move $m : s' = s \oplus m$. Depending on the type of problem, this can result in a significant reduction in the number of calculations.

The sub-neighborhood selection strategies described earlier in this section are also a means of reducing the overall computational complexity of TS implementations.

This section has provided a basic description of Tabu search. It should allow an understanding of the implementation that has been developed for pattern generation tasks, presented in the remaining sections of this chapter. At this point it should be emphasized that the research community in this domain has been developing an effort towards a deeper understanding of the method, both from the experience obtained by applying it under different conditions, as well as by trying to provide theoretical support to the success of a procedure which is fundamentally heuristic. Although there is still work to be done on the theoretical side, empirical research to date has given rise to extensions and variations of the original procedure which are continuously developing. Glover and Laguna (1997) provide detailed descriptions of additional mechanisms aimed at improving the robustness and the generality of the method.

6.3 Applying Tabu search for pattern generation

In this section, we provide the details of a TS implementation for generating a pattern, given positive and negative Boolean data. As opposed to the preceding section we replace the notation s, representing a solution, by t, given that in the current framework each solution is a term. In addition, we will denote feasible solutions by p, given that every term satisfying the constraints is a pattern, as will be explained later.

As previously referred, there are several elaborate heuristics for fine-tuning TS implementations, possibly up to an extreme level (Glover and Laguna, 1997). In certain combinatorial optimization problems the cost associated to sub-optimality can be very high, hence a considerable effort in fine-tuning is worth spending. The same applies when a single problem instance exists, in which case the optimization procedure is applied to generate one high-quality solution, or a small set of them, and where the terms of the problem are completely stated and in general static. In the current setting this is not the case. Rather, TS must be applied repeatedly in a "moving" context, i.e. where the general problem setting is fixed but the data change between every two optimization runs, according to the procedure **MultiClassLAD** defined in Alg. 5.1 on page 96. This reduces somehow the possibility of extreme parameter-tuning, which would have to be done using cross-validation techniques for every generated pattern. Instead, the proposed TS implementation and its parameters have been adjusted based on a set of different conditions that constitute, hopefully, a fair representation of a sufficiently large number of real situations where the current pattern generation procedure may be applied.

To summarize, the current problem setting is the following: given a search space of terms of limited degree and a set of Boolean data $\mathcal{Y}$ partitioned into a positive and a negative subset $\mathcal{Y}^+$ and $\mathcal{Y}^-$ respectively, the goal is to find the optimal term, according to a quality function that is calculated over the sets $\mathcal{Y}^+$ and $\mathcal{Y}^-$. The constraints include bounds on the positive and negative coverages of the term and on its degree. As will be discussed below, these three constraints are all treated in a different manner.

6.3.1 The GeneratePattern algorithm

Algorithm 6.1 contains the general TS procedure for single pattern generation. The following subsections provide more detailed information on each particular feature. In the algorithm, O^* represents the *cost* of the best visited solution, O^p is the cost of the best visited feasible solution p, and i is the iteration counter. Note that TS is being employed as a minimization procedure. Also concerning notation, the positive and negative *coverages* of a term t are

Algorithm 6.1 GeneratePattern($\mathcal{Y}^+$, $\mathcal{Y}^-$)

Inputs: $\mathcal{Y}^+$: set of positive Boolean data
$\mathcal{Y}^-$: set of negative Boolean data

Parameters: i_{max} : maximal number of iterations

$t \leftarrow$ initial solution *[term containing a literal chosen at random]*
$O^* \leftarrow O(t)$
$i \leftarrow 0$
if t satisfies the constraints **then** *[the initial solution is already feasible]*
  $p \leftarrow t, \qquad O^p \leftarrow O(p)$
else
  $p \leftarrow \varnothing, \qquad O^p \leftarrow \infty$
end if
while $i \ < \ i_{max}$ **do**
  $i \leftarrow i + 1$
  $H \leftarrow \{ \ t' : t' = t \oplus m \ \text{ and } \ (\ m \notin T \ \text{ or } \ O(t') < O^* \ ) \ \}$ *[note #1]*
  $t \leftarrow \arg\min_{t' \in H'} O(t'), \ H' \subseteq H(t)$ *[H' is a sub-neighborhood (note #2)]*
  if $O(t) < O^*$ **then** $O^* \leftarrow O(t)$ *[improvement of the best solution]*
  if t satisfies the constraints and $O(t) < O^p$ **then**
    [*improvement of the best feasible solution*]
    $p \leftarrow t$
    $O^p \leftarrow O(p)$
    $O^* \leftarrow O^p$ *[note #3]*
  end if
  update T with m^{-1} *[note #4]*
end while
return(p) *[best feasible solution found]*

denoted ω_t^+ and ω_t^-, respectively. These indicate the number of positive and negative training instances that are covered by t. Alternatively, the notation $\underline{\omega}_t^+$ and $\underline{\omega}_t^-$ is used for the *coverage rate* of t, which is the number of positive (resp. negative) instances covered by t divided by the total number of positive (resp. negative) instances, according to the definition provided earlier in Sect. 5.2.3 on page 94.

Two slightly different TS approaches for the current problem are proposed, one of them being an evolution of the other. The first one (called TS1) is described in detail below, and the other (TS2) in Sect 6.3.4 on page 133. An empirical comparison of both is then presented in Sect. 6.3.5.

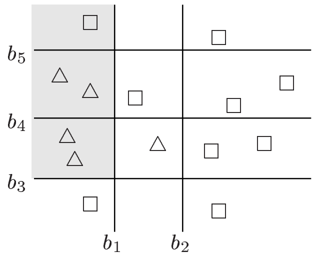
Solution and move

In a problem with B binary attributes, there are $2B$ possible literals, that is, a positive and a negative literal, respectively l_b and $\bar{l}_b$, associated with each binary attribute b. It is obvious that no interesting term contains both a literal l and its complement $\bar{l}$, from which the maximum effective term degree is B. In the current implementation, solutions are coded by a vector of B components taking values in the balanced ternary set $\{-1, 0, +1\}$, where component b has a value of $+1$ (resp. -1) if the literal l_b (resp. $\bar{l}_b$) is in the solution term, and 0 if neither of the literals associated with b are. Then, the degree of a term is equal to the number of non-null components in the vector.

With the above representation, making a move corresponds to changing one of the components in the vector, either by switching it from 0 to $+1$ or -1 (adding a literal to the term), or by switching it from $+1$ or -1 to 0 (removing a literal from the term). A move is represented by a triple $\{b, orig, dest\}$, where b is the binary attribute concerned (an index of the solution vector), and $orig$ and $dest$ are in the set $\{-1, 0, +1\}$. To make it clearer, in the following we denote the move corresponding to the triple $\{b, orig, dest\}$ by $\{b, orig \to dest\}$.

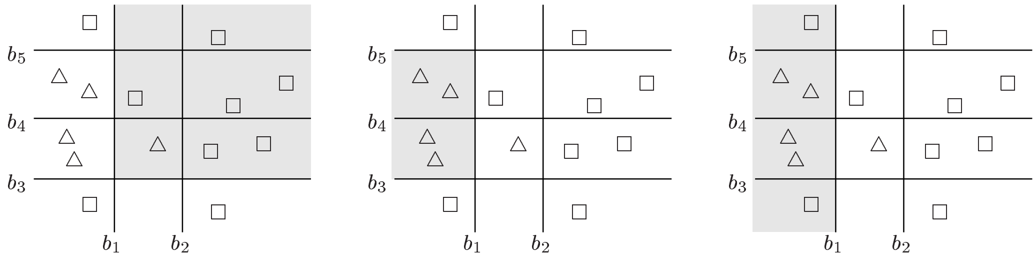
Moves of the type $\{b, -1 \to +1\}$ or $\{b, +1 \to -1\}$ are not allowed in the current implementation. Such moves correspond to replacing a literal by its complement. This operation can be interpreted geometrically with regard to the original feature space, where every term defines a sub-space delimited by the thresholds of the respective Boolean attributes. Replacing a literal in a term by its complement corresponds to transferring the sub-space defined by the term to the "other side" of one of its bounds, which is a rather radical change. In fact, there is no intersection between the two sub-spaces, the one before and the one after the move. This is shown in Fig. 6.2 on the facing page for the case of continuous attributes, where the term obtained by the move A has a very different coverage from the initial term. Binary attributes associated with discrete original attributes have a slightly different but comparable interpretation.

These extreme moves can thus be seen as a kind of diversification, though an uncontrolled one. While it might be beneficial in some situations, in others it could just drive the search away from an approaching attractive solution, thus introducing an undesired instability. In any case, if the search finds it appropriate to operate such a change, it can still do it in two consecutive iterations.



initial solution: $\bar{b}_1 b_3$

solution	term	coverage △	coverage □
initial	$\bar{b}_1 b_3$	4	1
after move A	$b_1 b_3$	1	6
after move B	$\bar{b}_1 b_3 \bar{b}_5$	4	0
after move C	$\bar{b}_1$	4	2



move A: $\{b_1, -1 \to +1\}$ new solution: $b_1 b_3$

move B: $\{b_5, 0 \to -1\}$ new solution: $\bar{b}_1 b_3 \bar{b}_5$

move C: $\{b_3, +1 \to 0\}$ new solution: $\bar{b}_1$

Figure 6.2: The move A replaces a literal in the solution by its complement, which results in a radical change of the coverage compared to the initial term. The moves B and C (adding and removing a literal, respectively) produce more moderate changes.

Neighborhood and aspiration condition

The line labeled with note #1 in Alg. 6.1 on page 121 contains the definition of the neighborhood for the current solution t : the set of solutions that can be reached from t using a move that is either not tabu or, in that case, that satisfies the aspiration condition. The aspiration condition used simply states that if a tabu move provides a solution that is better than any solution visited so far, than the move is accepted.

To determine the complete neighborhood $H(t)$ of a solution t, all the binary attributes are considered. For each binary attribute b, if the literal l_b (respectively $\bar{l}_b$) is in t, then $\{b, +1 \to 0\}$ (respectively $\{b, -1 \to 0\}$) is a possible move. Otherwise, $\{b, 0 \to -1\}$ and $\{b, 0 \to +1\}$ are possible moves. The empty term being uninteresting, moves like $\{b, -1 \to 0\}$ or $\{b, +1 \to 0\}$ are never considered when the current solution term has degree 1. Likewise, if the degree of the current term is equal to the maximal acceptable degree, then no moves of the type $\{b, 0 \to -1\}$ or $\{b, 0 \to +1\}$ are considered either.

Note #2 in the algorithm refers to the strategy of sub-neighborhood selection used. The sub-neighborhood H' is formed by selecting a subset of neighbors from $H(t)$ at random, one at a time. The effective size of H' is discussed in Sect. 6.3.3 on page 128. If one of the chosen neighbors is accessible through a move which is tabu, but that satisfies the aspiration condition, meaning that it is better than any previously obtained solution, then that neighbor is adopted immediately as the next solution, and no further neighbors are analyzed in the corresponding TS iteration.

Constraints

We distinguish three different kinds of constraints: *absolute*, *hard*, and *soft*[3]. Each one of the constraints in the current problem fits into one of these categories. We define an *absolute constraint* as one that reduces the search space from the beginning, meaning that no solution violating it will ever be visited. This is the case of the bound on the degree (G_{max}). The search space for the current problem is thus composed of all the terms of degree up to G_{max}. A *hard constraint* corresponds to the standard notion of constraint: some solutions in the search space respect it, while others do not. If a lower bound is imposed on the positive coverage of a pattern (ω^+_{min}), it is a hard constraint. Finally, a *soft constraint* is more special. It can be seen as a "moving" hard constraint. The solution obtained at the end of the optimization must satisfy it, but the bound changes in the course of the optimization procedure. That is, the constraint is alternately relaxed and hardened, according to the evolution of the search. The higher bound imposed on the negative coverage rate ($\underline{\omega}^-_{max}$) of a pattern fits into this category.

The distinction between hard and soft constraints is related to the discussion about effective modeling in Sect. 6.2.3, where the option of relaxing some of the initial constraints of the problem is defended. Of course, the search process must somehow take into account the existence of those constraints, and this is done by designing the objective function in such a way that it favors solutions respecting them. However, this creates difficult topological irregularities in the surface of the quality criterion, especially for severe constraints such as $\underline{\omega}^-_{max}$. It is intuitive that a term with a near-null negative coverage is in general difficult to find, especially one with also high positive coverage. So, to give the search some flexibility, $\underline{\omega}^-_{max}$ is treated as a soft constraint. The constraint variation scheme is described below in the framework of the objective function.

Concerning the actual values of the bounds, the maximal term degree has been set to five ($G_{max} = 5$), as mentioned in the previous chapter (Sect. 5.3 on page 99), which is a reasonable limit for a pattern to be interpretable. Conjunctions of more than five attributes have limited practical use for a domain expert. Another reason for limiting the degree of the patterns is to limit the capacity of the model, in order to prevent it from overfitting the training data, as discussed in Sect. 1.3. Besides, five is also a reasonable highest degree that can be treated by the standard breadth-first, bottom-up pattern search procedure of LAD described in Sect. 2.3.5 on page 22, in an acceptable amount of time (it is even excessive for the larger problems, as we have shown in Sect. 5.3.1). Setting the same limit for the incremental model proposed in this dissertation is a way of comparing the two approaches under similar conditions.

The lower bound for the positive coverage aims at avoiding patterns covering only a few positive instances. Also, considering that the coverage of some negative instances is allowed, it is important that more positive than negative instances are covered. In practice, two different limiting mechanisms have been defined: (i) the absolute positive coverage must exceed a minimum value, which has been set to 10 in the reported experiments (ω^+_{min} =10); and (ii) the proportion between the positive and the negative absolute coverages ($\omega_{minprop}$) must be higher than a minimal value, which has also been set to 10. For any pattern (feasible term), the

[3]To the best of our knowledge, this classification does not correspond to any standard adopted by researchers in the domain of optimization.

following condition must apply:

$$\frac{\omega_t^+}{\omega_t^-} \geq \omega_{minprop} = 10 .$$

This accounts for situations in which the cardinality of the negative class is much larger than that of the positive class.

The residual negative coverage that is allowed for the generated patterns is intended to tolerate classification noise contained in the training data. If a term covers a large amount of positive instances but also a few negative ones, it may be a bad choice to force the search to modify that term in order to remove the negative coverage. The presence of a few negative instances may not be disturbing, and forcing a change in the term in order to eliminate them will possibly eliminate also a large part of the positive coverage. In practice $\underline{\omega}_{max}^- = 0.02$ is used in all the reported experiments. Table 6.1 summarizes the values of the constraints.

CONSTRAINT	VALUE
G_{max}	5
ω_{min}^+	10
$\omega_{minprop}$	10
$\underline{\omega}_{max}^-$	0.02

Table 6.1: The constraints applied to the pattern search.

It is worth emphasizing that the values presented in Table 6.1, especially the last three, have been chosen based on common sense. The most appropriate values depend on the type of problem and on the level of noise in the data. A mechanism for automatically determining the best values for each particular situation would be a desirable evolution.

Objective function

There are three main indicators of the quality of a term t that can be measured on the training data: the positive and negative coverage (rates) $\underline{\omega}_t^+$ and $\underline{\omega}_t^-$ and the degree G_t. The first one should be maximized — the higher $\underline{\omega}_t^+$, the more positive examples are covered by t. As the general goal is to find a set of patterns discriminating certain data, the more positive examples are covered by each generated pattern, the less amount of patterns will be required, bringing compactness to the resulting classification model. Also, patterns with high positive coverage are strong indicators of the studied phenomenon, which is an interesting result that has been analyzed in Sect. 5.4 on page 104. The other two elements, $\underline{\omega}_t^-$ and G_t, should be minimized. As for G_t, a small term degree is also a way towards model compactness and interpretability. The case of $\underline{\omega}_t^-$ is more special, as it is a soft constraint, but the negative coverage should be minimized in any case. With these considerations in mind, for a given solution term t the objective function can be defined as follows:

$$\min: \ O(t) = \alpha\, G_t + \beta\, \underline{\omega}_t^- - \underline{\omega}_t^+ . \tag{6.1}$$

The role of parameters α and β is discussed below. We will refer to the value of the objective function as the *cost*, and use TS as a minimization procedure.

The parameter α determines the relative importance of the degree compared to the coverage rate or, more precisely, to the positive coverage rate because the negative coverage should be virtually null in any case. It is difficult to measure a priori the importance of each of these two factors precisely. The relation between them is difficult to disclosure and depends on the data. We consider that the most important quality of a pattern is ultimately its generalization ability, which cannot be calculated a priori, based on training data. The suitable value for α is studied in Sect. 6.3.3 on page 128 along with the other parameters of the TS method, taking this fact into consideration.

The parameter β in Eq. (6.1) is a penalizing factor. Its value determines the weight of the negative coverage rate in the objective function, which must be sufficiently high to penalize non-patterns terms. However, too high a value of β may render the search too restrictive and prevent access to interesting regions of the search space. To account for this, β is automatically adapted according to the evolution of the search. It is raised when the recently visited terms persist in showing a high negative coverage rate, and it is lowered when a series of terms with low negative coverage rate is traversed. The adaptation scheme changes the weight every i_β iterations, in the following manner: the value of β is doubled if the negative coverage rate $\underline{\omega}_t^-$ was non-null for all the terms t visited in the last i_β iterations, and it is reduced by half if $\underline{\omega}_t^-$ was always null for the same terms; no change is made otherwise. The value of the parameter i_β is also discussed in Sect. 6.3.3.

Changing the value of β, in either way, is a kind of diversification scheme, in the sense that in both cases the modification that is introduced in the objective function tries to persuade the search to give more importance to solutions which are different from those that have recently been visited. The difference lies in the fact that the value of β is raised after a series of unattractive solutions, and it is lowered after a series of attractive (feasible) ones. This balancing effect is named *strategic oscillation* by Glover and Laguna (1997, Chap. 4).

Fixed and variable objective functions

The soft constraint mechanism incorporated into the objective function creates inequalities when solutions obtained at different stages of the search are compared. The objective function, as described above, serves to guide the search towards appropriate regions, but an alternative *fixed* version is required for deciding whether the current solution is better than the best solution obtained so far without being biased by the current value of the β parameter. The adopted solution consists in discarding the weighting factor β of the negative coverage (which amounts to setting it equal to 1) from the *variable* version of Eq. (6.1), which yields:

$$\min: \; O(t) = \alpha\, G_t + \underline{\omega}_t^- - \underline{\omega}_t^+ \; . \tag{6.2}$$

Memory

As moves correspond to adding or removing literals to/from solution terms, these are the kind of actions that may become tabu. In the current implementation, the memory consists of a tabu list T of fixed size that stores the inverse of the most recent moves, noted by #4 in Alg. 6.1. This means that if a literal is removed from the solution term in some iteration, it is tabu to add it back during the next $|T|$ iterations. The opposite, i.e. to render tabu the removal of literals that have recently been added to the solution, does not apply. It has been stated previously that the degree of the searched terms is bounded by a quantity G_{max}, which is small compared to the usual number of Boolean attributes. This means that if we forbid recently added literals to be removed from the solution term, the search can quickly be blocked if $|T|$ is greater or equal to G_{max}. Then, it may happen that no literals can be added, because the maximal degree is attained, and none can be removed, because this is tabu. In fact, the search is considerably restricted even when $|T|$ is smaller than G_{max}.

To solve this problem, the memory T is split into two independent parts: a regular tabu list T_{in} storing the inverse of moves by which a literal is removed, and another memory component with a single element T_{out} storing the inverse of the last move by which a literal was added. For the rest, both are treated in the same manner. At iteration 0, both memories are empty; then, whenever a move is made, its inverse is added to the end of the corresponding memory. For T_{in}, its oldest element is discarded if its maximal length is exceeded. For T_{out}, the element contained therein is replaced by the new one. Using the adopted solution/move representation, when the b-th component of the solution vector changes from $+1$ (or -1) to 0, the triple $\{b, 0 \to +1\}$ (resp. $\{b, 0 \to -1\}$) is inserted in T_{in}. Likewise, when the b-th component is switched from 0 to $+1$ (or -1), the triple $\{b, +1 \to 0\}$ (resp. $\{b, -1 \to 0\}$) is inserted in T_{out}.

A move is tabu if it is contained in either of the two lists. Note, however, that the tabu status of a move can be overridden by the use of aspiration conditions. The choice of the tabu tenure for T_{in} is discussed in Sect 6.3.3 on the next page.

Additional features

Stopping criterion The stopping criterion used consists of a maximal number of allowed iterations (i_{max}). Section 6.3.5 on page 134 discusses the actual value that has been adopted.

Synchronization of best solutions As the set of feasible solutions is a subset of the whole solution space, the cost of the best feasible solution found at any phase of the search is probably higher (worse) than the cost of the best obtained solution overall. In the current implementation, whenever there is an improvement of the best feasible solution, the best global cost O^* is re-synchronized with the best feasible cost O^p. This is noted by #3 in Alg. 6.1.

6.3.2 Computational complexity

The most computationally demanding operation of the algorithm is the browsing of the sub-neighborhood (noted #2 in Alg. 6.1) in order to choose the best neighbor. The objective function must be calculated for each term in H', and this operation has linear complexity on the size of the training data set ($N = |\mathcal{Y}|$). The global complexity of the TS algorithm for generating a pattern is given by $\mathcal{O}(i_{max} \times |H'| \times N)$. As will be explained later, the value of $|H'|$ is itself a function of the number of binary attributes B in the problem and of the degree G_t of each solution term t (the latter is therefore bounded by G_{max}).

Behind the formal complexity, there is an important practical factor to be considered that reduces the actual computation time: the data are stored in Boolean format, and Boolean operations can be efficiently implemented by computers. In this particular case, given that all training instances are binary vectors of equal length, calculating the coverage of a term can be efficiently implemented through a series of AND operations. This does not change the formal complexity of the procedure but has the effect of reducing the actual time by a constant factor.

6.3.3 Parameter fitting

In this section, we discuss the choice of the appropriate parameter values for the TS algorithm. According to the description given, it contains five different parameters that need to be adjusted experimentally:

- the relative weight of the degree of a term in the objective function (the factor α in Eq. (6.1));
- the frequency at which the factor β in Eq. (6.1) is modified (i_β);
- the length of the tabu list T_{in} ;
- the size of the sub-neighborhood H' ; and
- the maximal number of iterations i_{max} allowed.

The number of iterations is studied in Sect. 6.3.5 on page 134. Here we perform a set of experiments aimed at defining the values of the other four parameters.

Every problem has its own characteristics, and requires a suitable set of parameter values. Nevertheless, we sustain that the proposed method can have satisfactory performance in a variety of different problems using a fixed parameter set. The current goal is to identify, for each parameter, a prominent value giving the best results overall for different problems. This will allow us to define the "default" procedure. Intensive, independent parameter tuning can, of course, provide increased performance for each particular application. Still, in the proposed formulation two of the parameters (the length of the tabu list and the size of the sub-neighborhood) are defined in a manner that takes the size of the problem into account, as will be described in the following.

Experimental setup

The actual goal of a pattern integrating a classification model is to discriminate between positive and negative data. The design of the current experiment takes this into account, but also the fact that the pattern is created with a finite amount of data, and that it should generalize, as explained in Sect. 1.1.2 on page 2. So, the current experiment consists of a set of cross-validation trials where the available training data set has been split into two equal-sized subsets, one used to generate a pattern and the other (the validation set) used to test its generalization performance. Note that the referred subsets contain no testing data, i.e. data used in other experiments of the current dissertation aimed at testing entire classification models.

Considering the standard procedure used in this dissertation of always using half of the available data for training and the other half for testing, the additional sub-partitioning of the training data into training/validation sets makes that the quantity of data available for generating a pattern is only a quarter of the total number of examples contained in each data set, and a similar amount is available for validation. This restricts the usable data sets, which have been chosen among all those described in Appendix A in such a way that the number of training (validation) examples is at least 100. The used data are taken from two classes chosen from each data set (a trivial choice for 2-class data sets). One of them has been assigned positive status and the other negative. Table 6.2 shows the chosen data sets, classes, and respective cardinalities, ordered by the number of Boolean variables (larger first), which means ordered by the size of the search space. The transformation of data into Boolean format has been made using the IDEAL algorithm proposed in Chap. 3, with plain consistency (cf. Sect. 3.1.3).

	NUMBER OF			NUMBER OF EXAMPLES			
	BOOLEAN	CLASS INDEX		TRAINING		VALIDATION	
DATA SET	ATTRIBUTES	POS.	NEG.	POS.	NEG.	POS.	NEG.
abalone	147	9	10	172	158	173	159
adult	138	1	2	2921	9288	2922	9289
letter	49	1	2	197	191	197	192
pendigits	35	1	2	285	285	286	286
spambase	33	1	2	697	453	697	453
yeast	31	1	2	115	107	116	108
optdigits	24	1	2	138	142	139	143

Table 6.2: Data used for determining the values of the parameters for the Tabu search procedure.

Each particular combination (parameter/tested value/data set) has been repeated five times with different initial random seeds. The bars in Figures 6.3 to 6.6 plot the average over these five trials. The plots contain four different results:

- the positive coverage of the obtained pattern on the training data (to be maximized);
- the same as above, but on the validation data (the most relevant result, to be maximized);
- the degree of the pattern (to be minimized); and
- the iteration in which the best solution has been found (to be minimized);

Concerning this last result, we precise that in every trial the search made 1 000 iterations (this was the value assigned to i_{max}).

For adjusting each parameter, all the other parameters have been assigned a fixed standard value chosen a priori as the average or reasonable value among those that are tested below. In this case we assume that the influence of each one of the parameters is independent of the others, which is probably not the case. This option is justified by the difficulty of presenting the results of all the combinations of values in an effective and clear manner. However, we expect the obtained conclusions to be reasonable even in case the independence assumption is not valid. The standard values that have been used are the following: $\alpha = 0.005$, $i_\beta = 25$ iterations, $|T_{in}| = \sqrt{2B}$, and $|H'| = \sqrt{2B - G_t}$. They are given convenient justification in the following descriptions.

Choosing α The value of α determines the relative weight of the degree for the quality of a term. Considering feasible solutions only, in which case the expression $\beta\,\underline{\omega}_t^-$ is nearly irrelevant in the objective function of Eq. (6.1), the weight α means that a solution t of degree G is better than a solution t' of degree $G-1$ only if t covers an additional proportion α of positive instances comparing to t'. If $\alpha = 1$ the degree becomes the first quality factor of a term, and the degree of the optimal solution is guaranteed to be equal to that of the feasible solution of lowest degree found. This means that a solution of degree G can never be better than a solution of degree $G-1$, independently of their coverages. On the other end of the spectrum, if $\alpha = 0$ the degree has no importance, only the coverage matters. Other values in this range correspond to different levels of compromise. Five different values have been tested: 0, 0.005, 0.05, 0.5, and 1 (Fig. 6.3).

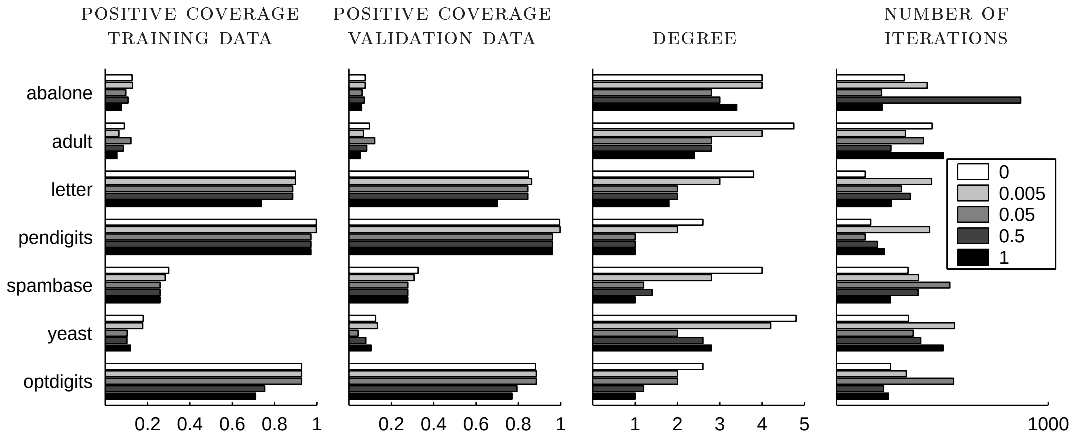


Figure 6.3: Cross-validation results for different values of α.

The plots indicate that the resulting degree decreases with increasing α in general, as expected. But the other outcome of according more importance to the degree is a smaller positive coverage in general, which conditions our decision towards a small α. The more interesting values are 0 and 0.005, which provide comparable coverage. However, 0 provides solutions of higher degree than 0.005, so the value 0.005 is the one selected. In fact, such a small value has a natural justification: the degree should be used just to decide among two solutions of different degree but with comparable coverage on the ensemble of problems. Finally, we remark that α has no clear influence on the number of iterations.

Choosing i_β A change in β implies a change in the search process. It is a diversification mechanism, as referred in Sect. 6.3.1 about the objective function. The adopted TS implementation makes a change in β whenever a sequence of i_β iterations provided only infeasible solutions or only feasible ones. Hence, small values of i_β correspond to a very flexible adaptation scheme, and large values to the opposite. Apart from that, no further element provides indication for reasonable values of i_β. The following have been tested: 5, 25, and 50 iterations (Fig. 6.4 on the next page).

We can draw two main conclusions: (i) there are only slight differences in the results, and (ii) these differences do not show any clear trend, and hence do not allow to conclude whether the adaptation process needs flexibility or not. The value 5 was chosen (more flexible adaptation).

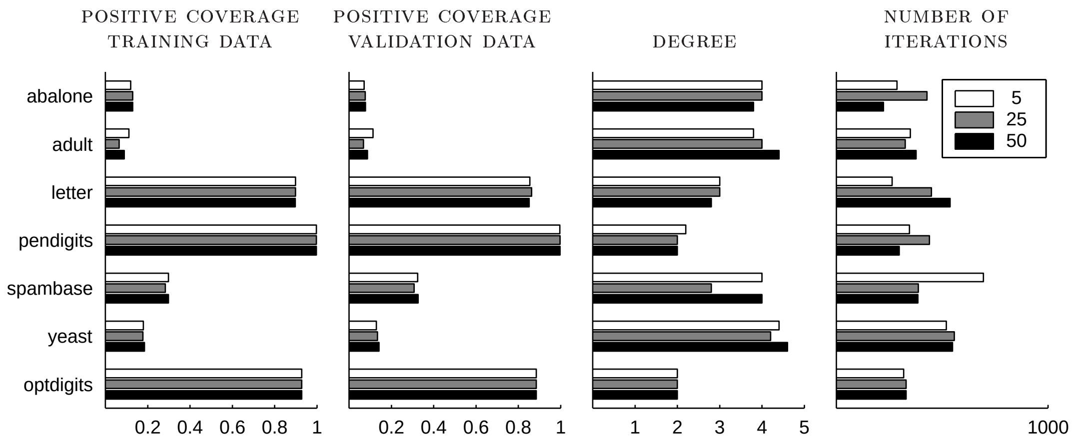


Figure 6.4: Cross-validation results for different values of i_β.

Choosing $|T_{in}|$ The length of the tabu list has intuitively a large importance for the performance of the method. To choose the appropriate value, we have taken a baseline value, and tested it against its half and its double values. The baseline is denoted by Δ and it is equal to the square root of the total number of elements that can possibly enter the list in each iteration. This choice was suggested by Prof. Alain Hertz, and is motivated by his practical experience in applying TS to different domains. In our case, all the literals are candidates to enter the tabu list T_{in}, except those that are in the solution. The latter have low importance, though, given that the bound on the solution degree is small comparing to the usual total number of literals. In addition, the degree of the solution changes at each iteration, and if this was taken into account, the tabu length would also vary at each iteration. So, this factor is neglected and Δ is set equal to $\sqrt{2B}$. The results of the test are shown in Fig. 6.5.

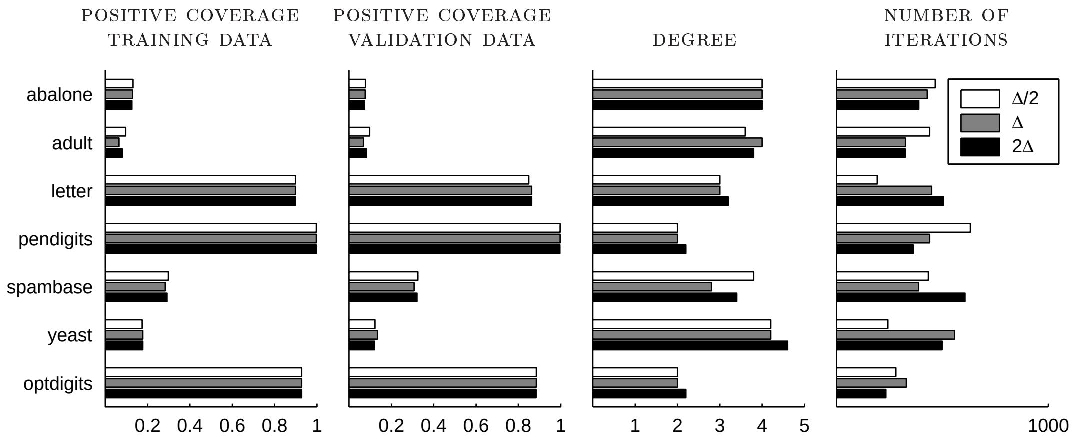


Figure 6.5: Cross-validation results for different values of $|T_{in}|$. The quantity Δ represents a baseline value, in the order of the square root of the total number of elements that can enter the tabu list ($\Delta = \sqrt{2B}$).

Surprisingly, the method does not produce very different results with different values of the parameter. We have chosen the lowest value, $\Delta/2$.

Choosing $|H'|$ For choosing $|H'|$ the same reasoning has been applied as for $|T_{in}|$, which means that the baseline consists in choosing a sub-neighborhood equal to the square root of the total neighborhood. The size $|H|$ of the total neighborhood is equal to the number of literals minus those that belong to the solution. The fact that the degree of the solution changes at each iteration, with a consequent change in the neighborhood size, can be more naturally accommodated in this case than for $|T_{in}|$. So, the value of Δ is here equal to $\sqrt{2B - G_t}$, where t is the current solution.

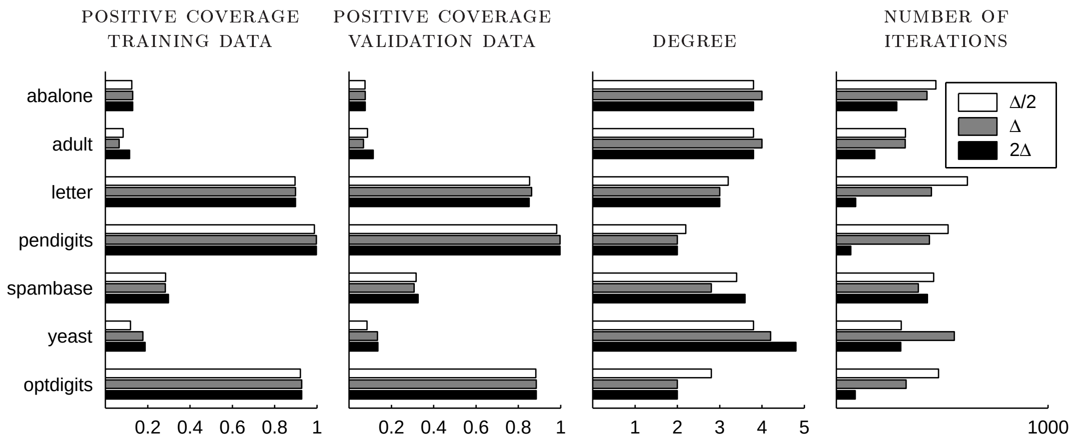


Figure 6.6: Cross-validation results for different values of $|H'|$. The quantity Δ represents a baseline value, in the order of the square root of the total neighborhood size ($\Delta = \sqrt{|H|}$).

In this case also, the obtained results are fairly insensitive to changes in the parameter, with the exception of the number of iterations. It is clear from the plot that a larger sub-neighborhood allows the search to reach the best solution faster. Hence 2Δ is the value chosen.

Discussion of the various results As Fig. 6.3 on page 130 shows, the value of α has a significant influence on the obtained solution. This is a natural result if we consider that this parameter has a very direct influence on the objective function. The best value found with the tests has also a natural interpretation, and so we can consider that α is appropriately determined.

Concerning the other three parameters, we observe that they have a modest influence on the performance of the proposed approach. The different tested values result in comparable behavior for the used sets of data. This allows a very important conclusion: the problem has been appropriately modeled, according to the discussion in Sect. 6.2.3, and is relatively insensible to the parameter choice, at least where it would be reasonably expected to. Table 6.3 on the facing page summarizes the adopted parameter values.

PARAMETER	VALUE
α	0.005
i_β	5 iterations
$\lvert T_{in}\rvert$	$\sqrt{2B}\,/\,2$
$\lvert H'\rvert$	$2\sqrt{2B-G_t}$

Table 6.3: Definitive parameter values for the Tabu search procedure.

6.3.4 Avoiding multiple criteria — alternative TS implementation

The algorithm described in Sect. 6.3.1 performs a pattern search guided by two independent criteria: the coverage and the degree of the pattern. It is not easy to reduce this duality to a single global criterion in a natural manner. This has been done so far through the use of the parameter α that defines the relative importance of each of the criteria. To avoid this situation, an alternative implementation of the algorithm is proposed, which searches patterns based on the coverage only, but does it independently for each degree. The idea is to distribute the total number of iterations by the sub-searches performed for the different term degrees. The two implementations will be referred to as TS1 (the earlier) and TS2 (the new one).

TS2 starts by executing the TS procedure defined in Alg. 6.1 on page 121 once to search only terms of degree 1, then executes it again for terms of degree 2, and so on, up to degree G_{max}. There are three main implications of this: (i) the objective function must be changed to match the new criterion; (ii) the definitions of move and neighborhood must be adjusted, since for every solution t, all its neighbor solutions must be of the same degree as t ; and (iii) the number of iterations for each search run must be appropriately defined.

Concerning the objective function, it is actually the variable version that must be changed, since the fixed one must still be used to compare solutions found in different search levels. The change is quite simple: comparing to Eq. (6.1) on page 125, it implies dropping the factor related to the degree:

$$\min:\ O(t) = \beta\,\underline{\omega}_t^- - \underline{\omega}_t^+ \ .$$

In order to keep the degree unchanged when moving between solutions, the number of non-null components in the ternary vector that is used to code solutions must not change. So, a move applied to a solution term should inevitably consist of an exchange of literals. Comparing to TS1, a move is composed of two sub-moves, one switching a component of the solution vector from -1 or $+1$ to 0, and another one switching some other component from 0 to -1 or $+1$. As before, the removed literal goes into the tabu list T_{in} while the inserted one enters the tabu list T_{out}.

As the stopping criterion is based on a maximal number of iterations i_{max}, the strategy employed here consists in splitting the number i_{max} into slots that are allocated to the different search levels according to their respective search space size. Considering again the plot of Fig. 2.4 on page 23, it can be seen that the number of possible solutions grows exponentially with the degree. If the iterations were distributed proportionally to that number, the highest degree would receive nearly the totality of them. So, the distribution is based on the logarithm of the search space size instead, according to the following formula:

$$i_{G_{max}} = \frac{\log(\text{ number of solutions of degree } G \text{ })}{\sum_{G'=1}^{G_{max}} \log(\text{ number of solutions of degree } G' \text{ })} \times i_{max} \tag{6.3}$$

During the search for terms of degree G, the total number of evaluations of the objective function is equal to the size of the sub-neighborhood $|H'|$ multiplied by the number of iterations $i_{G_{max}}$. For certain problems it may happen that the number of possible solutions for the smaller degrees is lower than $|H'| \times i_{G_{max}}$. In these cases, it is preferable to search the respective space of solutions exhaustively, since this demands a single evaluation of the objective function per solution. More specifically, in TS2 the space of terms of degree G is searched exhaustively whenever:

$$2^G \binom{B}{G} \leq i_{G_{max}}(2\sqrt{2B-G})$$

The next section compares the two approaches empirically.

6.3.5 Comparison between the two TS approaches

Performance

To compare TS1 with TS2, a set of experiments has been performed under the same conditions as those reported in Sect. 6.3.3 on page 128 for the choice of the TS parameters, with three different stopping criteria: $i_{G_{max}} = 100$, $i_{G_{max}} = 1\,000$, and $i_{G_{max}} = 10\,000$. Figure 6.7 shows the performance achieved by the two TS approaches with regard to the positive coverage of the obtained solution on the validation data, while Fig. 6.8 on page 136 shows the degree of the solution pattern.

Concerning the positive coverage on the validation data (the most important result), there are two main conclusions that can be drawn from the plot: (i) the methods have comparable performance, except in the `adult` problem, which is one of the largest (see Table 6.2), and where TS2 is better; and (ii) the coverage rate of the obtained patterns does not scale with the raise in the number of iterations, although a small increase is detected in some cases. The plot of Fig. 6.8 adds no significant information as the results are also comparable, except that using only 100 iterations produces less good patterns in some cases, because they have higher degree for an equivalent positive coverage. However, for `spambase` it is the opposite. Also, for this data set the patterns are on average almost one degree lower with TS1 than with TS2.

These results determine the choice of TS2 as the adopted TS method for pattern finding, not only due to its generally better performance, but also because it appears to be the more stable and reasonable approach. At the same time, $i_{max} = 100$ is the adopted stopping criterion, given that ten or a hundred times more iterations do not provide important improvements of the obtained solutions. In addition, using more than 100 iterations would cause the **MultiClassLAD** algorithm to be unreasonably slow for large problems (see the results of Sect. 5.3). In the following, some additional insight is given into the difference between the two methods and two of the stopping criteria.

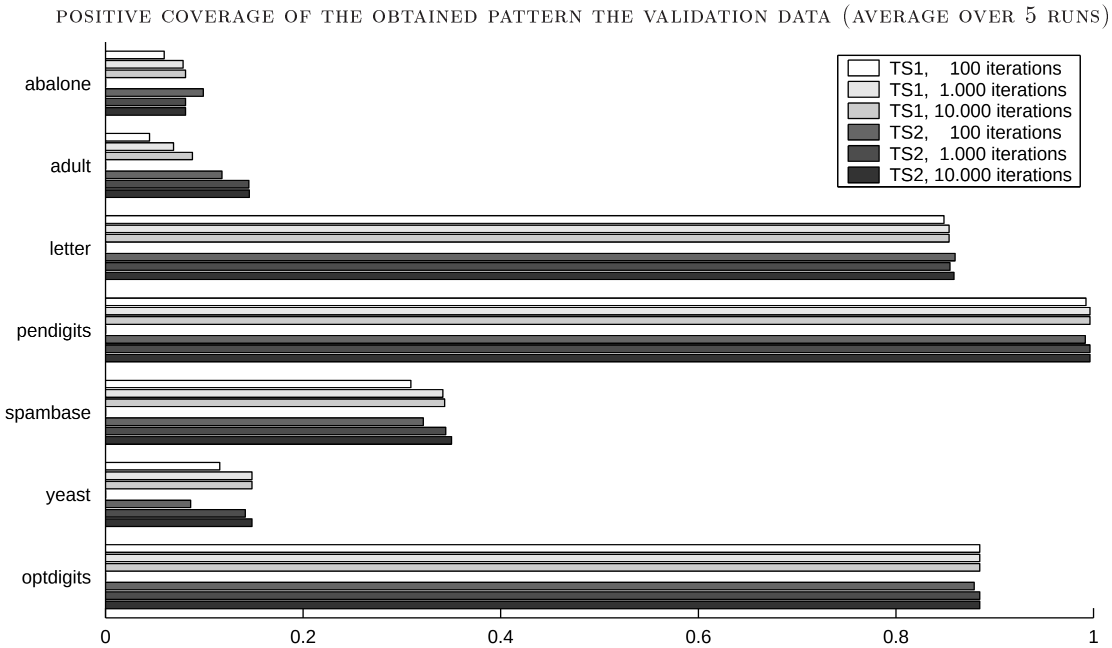


Figure 6.7: Comparison between the two TS approaches (TS1 and TS2) with regard to the positive coverage of the obtained pattern on the validation data, using three different stopping criteria: 100, 1 000, and 10 000 iterations.

Search evolution

Here, we make a more detailed analysis of some typical and interesting cases extracted from the experiments described above. A series of plots, presented in Figures 6.9 to 6.16 at the end of this chapter, on pages 142 to 145, show the evolution of two different search processes with the `adult` data set, each with a different random seed, plus one search process with `spambase`, and another with `yeast`.

Each figure contains two parts: the upper one shows the evolution of the cost of the best feasible solution, while the lower part shows the evolution of the cost for all the visited solutions. The horizontal axes contains the number of iterations, and the two plots in each figure are synchronized, i.e they correspond to exactly the same search process. On each page the evolution for one of the problems can be compared between TS1 (above) and TS2 (below).

As can be seen in the figures, the plots of TS2 are horizontally divided into sections, each one corresponding to a degree. As mentioned in Sect. 6.3.4, these sections correspond to independent search processes. Every complete section represents the search process when $i_{max} = 10\,000$ iterations, while the vertical dotted lines mark the limit of the search for the case $i_{max} = 1\,000$. This means that, inside each section, the search evolution beyond the dotted lines (to the right) is not executed when only 1 000 iterations are allowed. Also, in this case the values of the best feasible solution for each degree slot are independent. When a continuous, horizontal line appears at the left of a TS2 plot, it means that those degrees have been subject to exhaustive search, and the line shows the value of the best solution found for all of them.

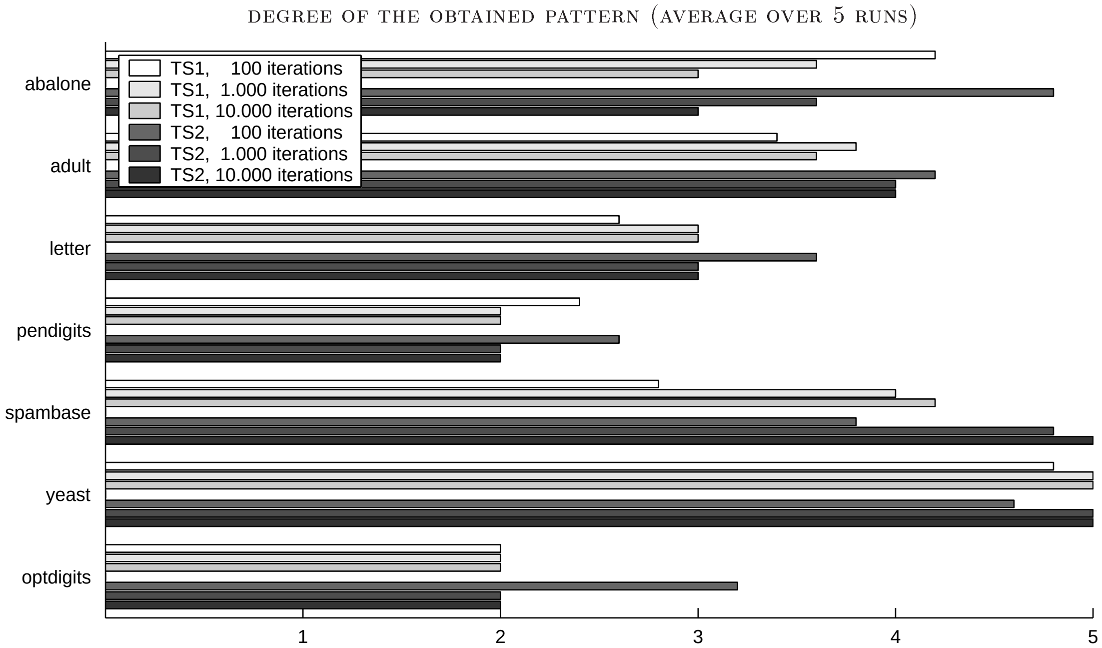


Figure 6.8: Comparison between the two TS approaches (TS1 and TS2) with regard to the degree of the obtained pattern, using three different stopping criteria: 100, 1 000, and 10 000 iterations.

The plotted values are all calculated using the same objective function (given in Eq. (6.2) on page 126), both for TS1 and TS2. In addition, for each data set all the plots fit exactly in the same y-axis range, so all the values on a page can be directly compared.

In general, it can be verified that the search process is highly irregular, and that TS2 reaches low cost solutions much more frequently than TS1. Note that only few of them are feasible, in which case they can eventually bring down the value plotted above. This situation appears quite clearly in the plots of Fig. 6.9 for example, where certain extreme cost reductions (lower plot) visibly improve the best feasible solution (upper plot).

The most significant improvements of the best feasible solution are mostly made inside the 1 000-iteration slot (to the left of the dotted vertical lines). Moreover, whenever there are additional improvements beyond that region, the graph of Fig. 6.7 inevitably shows a difference between the performance with $i_{max} = 1\,000$ and $i_{max} = 10\,000$. Another interesting observation is that in most cases 1 000 iterations are easily enough to find the best solution, and although the image resolution of Figures 6.9 to 6.16 does not allow to see it clearly, Figures 6.7 and 6.8 prove that 100 iterations already provide good solutions, in general.

6.3.6 Summary of the Tabu search features

fixed objective function	• min : $\alpha\, G_p \;+\; \underline{\omega}_p^- \;-\; \underline{\omega}_p^+$
variable objective function	• min : $\beta\, \underline{\omega}_p^- \;-\; \underline{\omega}_p^+$
	• weight of degree of a term in the objective function: $\alpha = 0.005$
absolute constraint	• maximal term degree $G_{max} = 5$
hard constraints	• minimal positive coverage $\omega_{min}^+ = 10$
	• $\frac{\omega_t^+}{\omega_t^-} \geq \omega_{minprop} = 10$
soft constraint	• maximal negative coverage rate $\underline{\omega}_{max}^- = 0.02$
	• the factor β in the objective function is variable: $\beta = \beta \times 2$ after i_β iterations with $\beta > 0$ $\beta = \beta\,/\,2$ after i_β iterations with $\beta = 0$ $i_\beta = 5$
solution	• vector of B ternary components $(b_1\ b_2\ \ldots\ b_B)\ \in\ \{-1, 0, +1\}^B$
move	• flip two different components of the solution vector replacing a literal by another. Possible moves: $\{b_i, 0 \rightarrow +1\}$ and $\{b_j, +1 \rightarrow 0\}$ $\{b_i, 0 \rightarrow +1\}$ and $\{b_j, -1 \rightarrow 0\}$ $\{b_i, 0 \rightarrow -1\}$ and $\{b_j, +1 \rightarrow 0\}$ $\{b_i, 0 \rightarrow -1\}$ and $\{b_j, -1 \rightarrow 0\}$
sub-neighborhood	• random selection of $H' \subset H(t)$, the complete set of neighbors of t ($\lvert H' \rvert = 2\sqrt{2B - G_t}$)
tabu lists	• store the inverse of made moves
	• T_{out} prevents the last added literal to be removed from the solution ($\lvert T_{out} \rvert = 1$)
	• T_{in} prevents the last $\lvert T_{in} \rvert$ removed literals to be added again to the solution ($\lvert T_{in} \rvert = \sqrt{2B}\,/\,2$)
aspiration condition	• if a tabu move improves the best solution, it is accepted
stopping criterion	• maximal number of iterations i_{max} $i_{max} = 100$, distributed along independent search runs for terms of different degree (cf. Eq. (6.3))

6.4 Related work

6.4.1 Rule induction

As already mentioned in Sect. 2.4 on page 27, many rule induction systems proposed in the Machine Learning community construct classification models of a type similar to LAD, based on a set of *rules*. A rule is equivalent to a term in LAD. It is composed of an antecedent, which is normally a conjunction of attribute values, and a consequent, which is a class label, of the form: *IF condition THEN class label.* In particular, rule induction methods based on *sequential covering* (Sect. 2.4.2) are of particular interest here, since they also employ a rule generation engine comparable to the single pattern generation described in this chapter.

During the description of the standard LAD method, in Sect. 2.3.4 on page 21, a structured organization of the search space of terms has been presented, based on a *lattice* and the associated *Hasse diagram*, where terms of different degrees are connected by the *more-specific-than* relation. Then, subsequent descriptions of various term/rule search methods, either proposed in the LAD framework or in other algorithms, have been based on that organization of the search space. Concerning the direction of the search specifically (bottom-up by specialization of terms or top-down by generalization), it has been mentioned that some methods are not restricted to a single direction, but allow moves to be done in both directions (both generalizations and specializations are made in the same search process). As this is also the case for our TS procedure, a brief description of alternative methods employing this same strategy is given below.

In a method called ATRIS, Kononenko and Kovačič (1992) apply a meta-heuristic method for rule discovery which is different from TS: Simulated Annealing (SA) (Kirkpatrick et al., 1983; Aarts and Korst, 1989). The main principle of this method is to allow non-improving moves with a probability that is initially high but that decreases with time. The probability of acceptance of non-improving moves is controlled by a parameter called the *temperature*. Considering that we want to maximize the objective function, the probability of accepting a new solution t' is given by:

$$P(\text{acceptance of } t') \;=\; \exp\left(\frac{O(t) - O(t')}{\text{temperature}}\right)$$

where t is the current solution. The move to t' is taken if the probability of acceptance exceeds a random number in the range $[0, 1]$. This mechanism is intended to escape local optima. The temperature is initially high and decreases gradually, which means that the search is largely stochastic in the beginning, but when the temperature decreases it becomes deterministic. The authors provide no details about the temperature reduction scheme.

The condition part of the rules considered in ATRIS contains no negated literals. Conjunctions are based only on positive values of Boolean variables. In this case, moves are made by adding or removing a variable to/from the conjunction. The search process is governed by the *J-measure*:

$$\max:\; O(t) \;=\; \omega_t \left(-\,\frac{\omega_t^+}{\omega_t} \log \frac{\frac{\omega_t^+}{\omega_t}}{\frac{|\mathcal{Y}^+|}{|\mathcal{Y}|}} \;-\; \frac{\omega_t^-}{\omega_t} \log \frac{\frac{\omega_t^-}{\omega_t}}{\frac{|\mathcal{Y}^-|}{|\mathcal{Y}|}} \right) .$$

The main body of the formula represents a *cross entropy* measure, or average expected mutual information between the positive and negative classes and the rule. The factor ω_t at the beginning is intended to provide a measure of simplicity of the rule, represented here by its coverage. The intuition is that a rule covering many instances (either positive or negative) is simpler (shorter), and so a direct relation is assumed between the coverage of a rule and its length. Altogether, the formula is assumed to provide the average information content of the rule.

Also in the ATRIS framework, Mladenić (1993) proposes a common heuristic for deterministic search, called *k-opt*, normally employed for solving the traveling salesman problem (Cook et al., 1998, Chap. 7). In this case, a move consists in flipping up to k bits in the solution, which contains values in $\{0, 1\}$. In standard k-opt, it is always the best neighbor that is chosen. Mladenič also uses a "faster" variant which accepts the first tested neighbor that is better than the current solution. The search is guided by the simple *relative frequency* measure, which has been used in the AQ method (Michalski, 1969; Michalski et al., 1986):

$$\max: \; O(t) = \frac{\omega_t^+}{\omega_t} \; .$$

Fensel and Wiese (1993) proposed a rule finding procedure named JoJo. Starting from a solution chosen either at random, based on certain heuristics, or equivalent to some previously obtained rule, JoJo moves in the search space by adding or deleting one literal at each iteration, which respectively specializes or generalizes the current rule. The search is guided by a maximization of the *g-measure*:

$$\max: \; O(t) = 1 - \frac{\omega_t^+ + 0.5}{\omega_t + 0.5} \; ,$$

and generalizations are preferred over specializations. The latter are done only if the current rule has an *error rate* above a pre-defined threshold, otherwise the best generalization is chosen. If no better generalization exists, the search stops.

Later on, Fensel and Wiese (1994) developed Frog, an evolution of JoJo that allows also literal replacement, as well as "jumps", which are moves causing more than just one literal modification. These aim at accelerating and/or diversifying the search, but the length of the allowed jumps is reduced as the quality of the solution improves. Another mechanism used in this algorithm is a memory that keeps trace of all the visited solutions to avoid search loops, something like a "complete" Tabu list.

The TABATA algorithm

The TABATA algorithm by Brézellec and Soldano (1998), that we have tested empirically in the previous chapter, uses a simple Tabu search formulation for finding rules in a sequential covering manner. However, the search for each individual rule operates in a restricted search space, defined by a *seed* positive example chosen at random from the not yet covered positive data, as in the AQ method (see Sect. 2.4.2). This means that only terms (rules) covering the seed are considered in the search process. As in ATRIS (cf. above), conjunctions are based only on positive values of the Boolean variables, no negations are considered, and so moves are made by either adding or removing a variable to/from the current term (solution). The value of the objective function for a term consists of the number of positive examples that it covers plus the number of negative examples that it does not cover:

$$\max: \; O(t) = \omega_t^+ + (|\mathcal{Y}^-| - \omega_t^-) \; .$$

In TABATA, infeasible solutions are never considered, since only terms with null negative coverage are allowed to be visited. Additional TS features include a tabu list of visited terms, of fixed, user-defined length (7 is the default), and a stopping criterion based on a maximal number of iterations without improvement of the best solution (10 is the default), also chosen by the user. The stopping criterion seems quite restrictive. However, the empirical tests of Sect. 5.3.2 in the previous chapter have shown that TABATA was the method that took the longest time to execute in general, among all the tested methods. Hence, we did not consider the possibility of raising this number.

6.4.2 Genetic algorithms for rule induction

The search for classification rules using Genetic Algorithms (GA) has already been described in Sect. 2.4.4 on page 32. GA and Tabu search are usually considered as concurrent heuristic procedures for solving combinatorial optimization problems. From the two traditional GA procedures for rule induction, the Michigan (Holland, 1986; Wilson, 1987) is the one more directly related to the approach described in this chapter, given that each one of the solutions in the population represents a rule (a term). However, a distinguishing feature between both is that while in the Michigan approach the population of solutions defines directly the classification model, which is established in a single optimization run, our TS procedure generates one single independent pattern at each time, and several runs are needed to create the model, as in standard sequential covering (cf. Sect. 2.4.2). The fitness function used by GA in this case must take into account the quality of the entire population (the model), and not just of each solution individually.

6.5 Discussion

From the description made, it is clear that the TS algorithm does not guarantee that a solution of good quality is obtained, not even a feasible solution (cf. the definition of heuristic mentioned in Sect. 6.1.1). In fact, it is not possible to make a safe statement regarding the difference between the obtained patterns and the globally optimal solution. However, there are some elements indicating that the proposed method has a good performance. The plots of Sect. 6.3.5 containing the evolution of the search show that even if a very large number of iterations is allowed, the best solutions are mostly found in the first iterations. Thus, the fact that a long search process continues after the first iterations without significantly improving the previously obtained solutions may be an indicator that these are close to the optimum, but this is not more than a conjecture.

The success of the procedure probably depends on the particular constraints imposed and their pertinence with regard to each treated problem. For example, problems of larger size require, in principle, more iterations to find good solutions, because the search space is much larger. As we have already mentioned on several occasions, fine-tuning each parameter value for a particular learning task can lead to improved results compared to those obtained here with the standard configuration described in Sect. 6.3.6.

We have provided a rather lengthy and elaborated description of the proposed pattern generation engine. The question arises of whether it is worth defining this procedure with such detail. In Sect. 5.3.2 we have presented an empirical comparison with an alternative, much simpler, procedure: TABATA (described on page 139). This method also employs a Tabu search approach for finding rules, and the results it produces are much poorer than those obtained by our procedure **GeneratePattern**. Besides the Tabu search implementation, the two methods differ also on the search space that is explored. We might thus wonder which of these two factors justifies the different results. However, the other rule-based method tested in Sect. 5.3.2 (CN2) also searches a limited space (as is the case with TABATA), and it has proven to be quite competitive in the generality of problems. There is thus strong evidence that the modeling effort that is recommended in Sect. 6.2.3 for Tabu search implementations, and that we have followed for defining the **GeneratePattern** procedure, has effectively been rewarded.

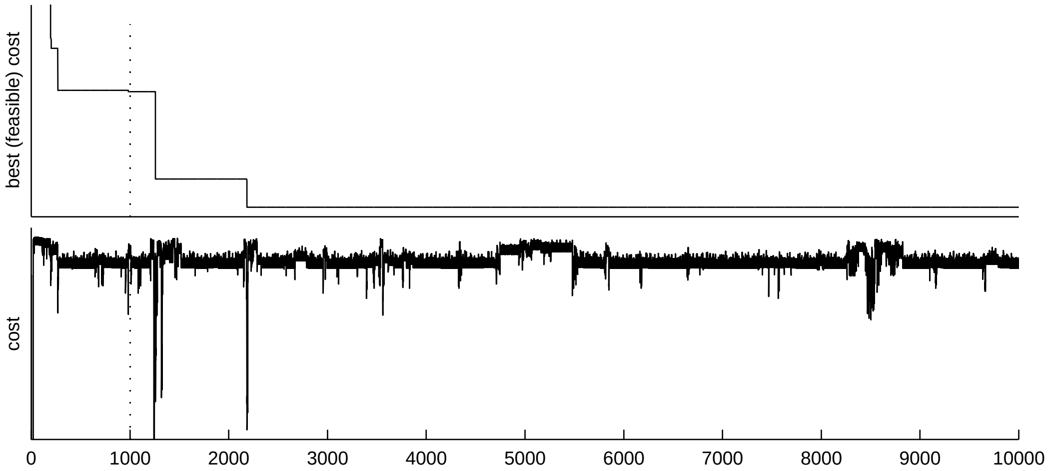


Figure 6.9: Search evolution for `adult`, with TS1 and random seed 2 (compare with Fig. 6.10).

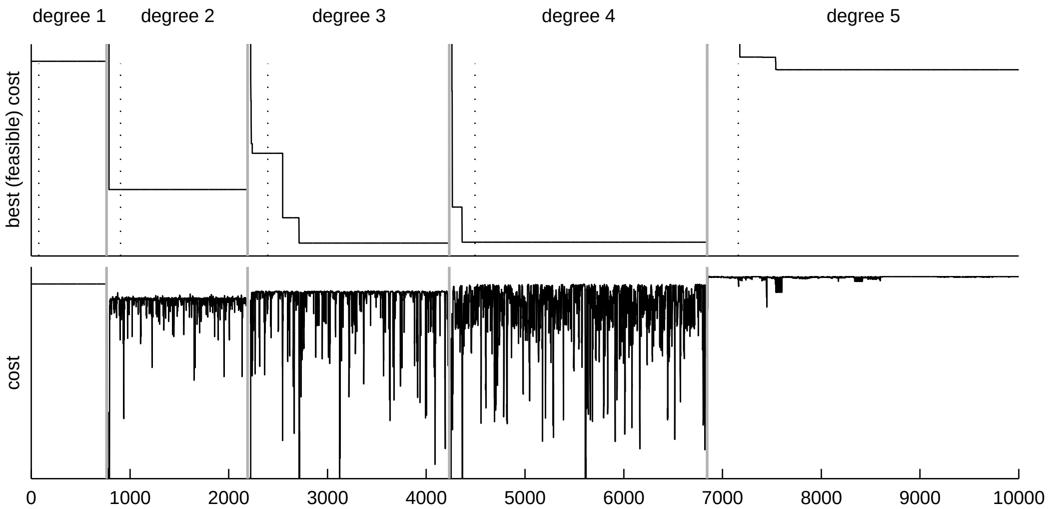


Figure 6.10: Search evolution for `adult`, with TS2 and random seed 2 (compare with Fig. 6.9).

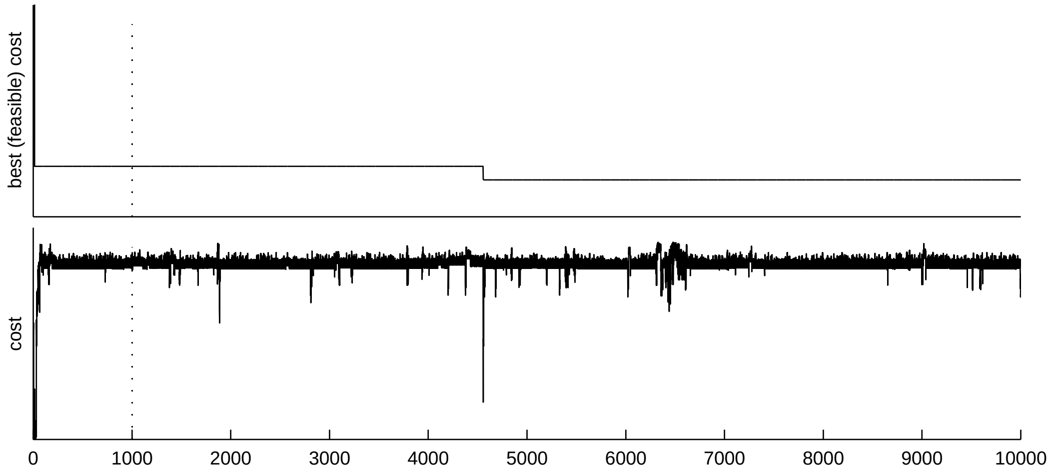


Figure 6.11: Search evolution for `adult`, with TS1 and random seed 5 (compare with Fig. 6.12).

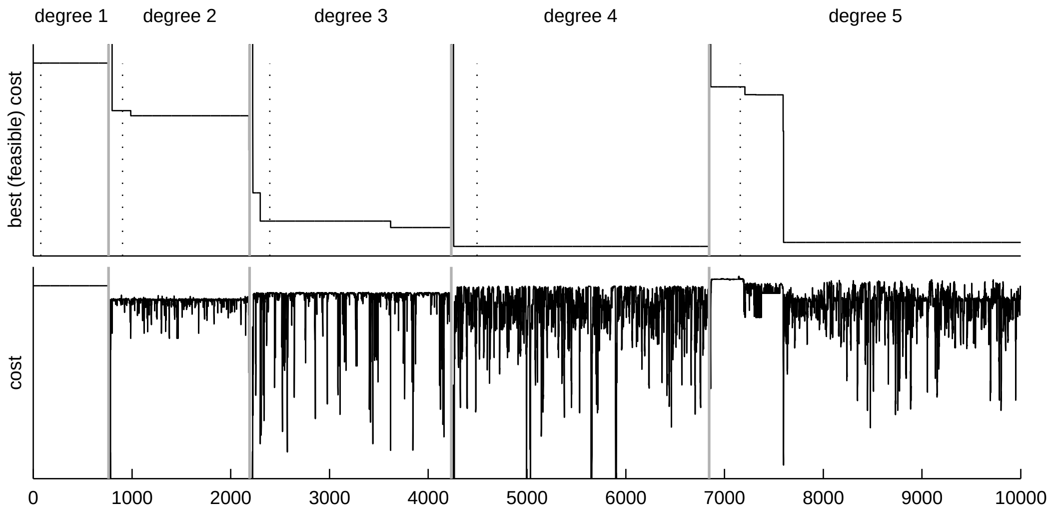


Figure 6.12: Search evolution for `adult`, with TS2 and random seed 5 (compare with Fig. 6.11).

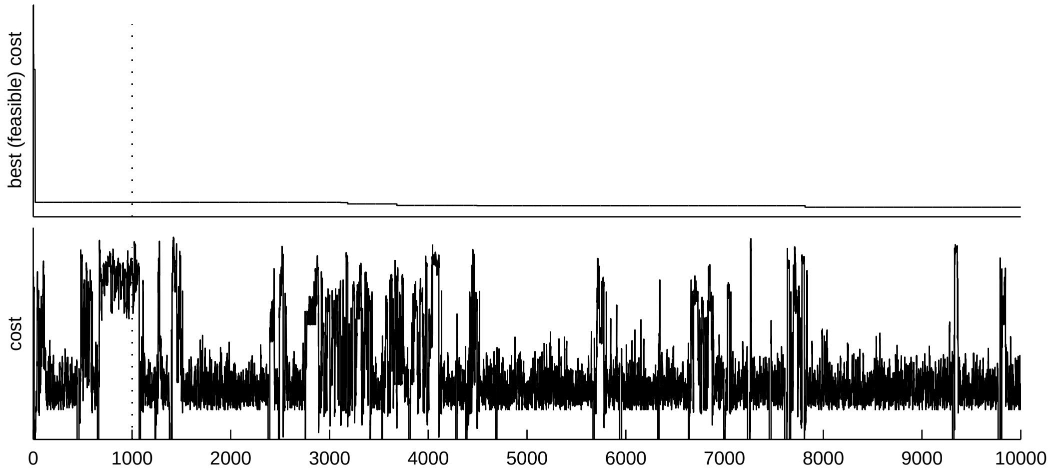


Figure 6.13: Search evolution for `spambase`, with TS1 and random seed 4 (compare with Fig. 6.14).

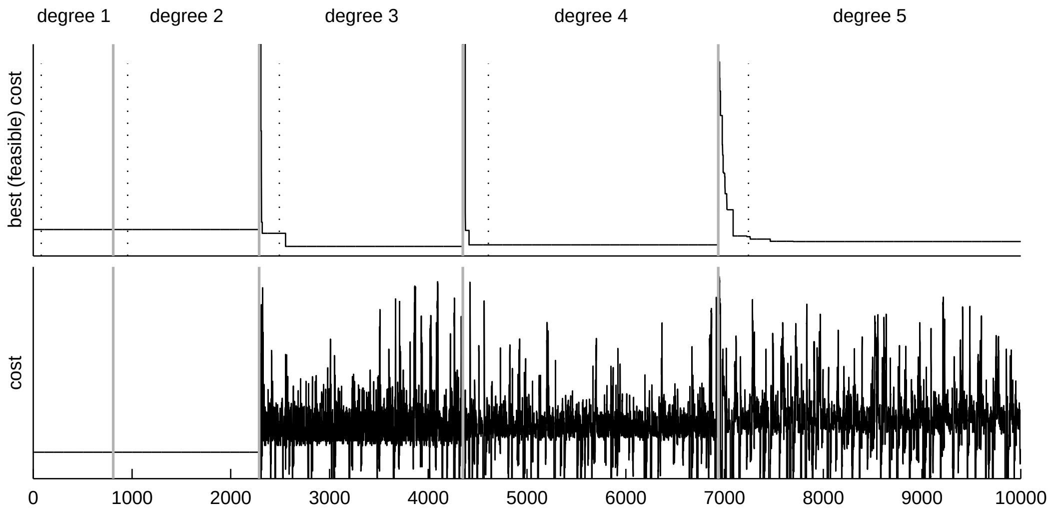


Figure 6.14: Search evolution for `spambase`, with TS2 and random seed 4 (compare with Fig. 6.13).

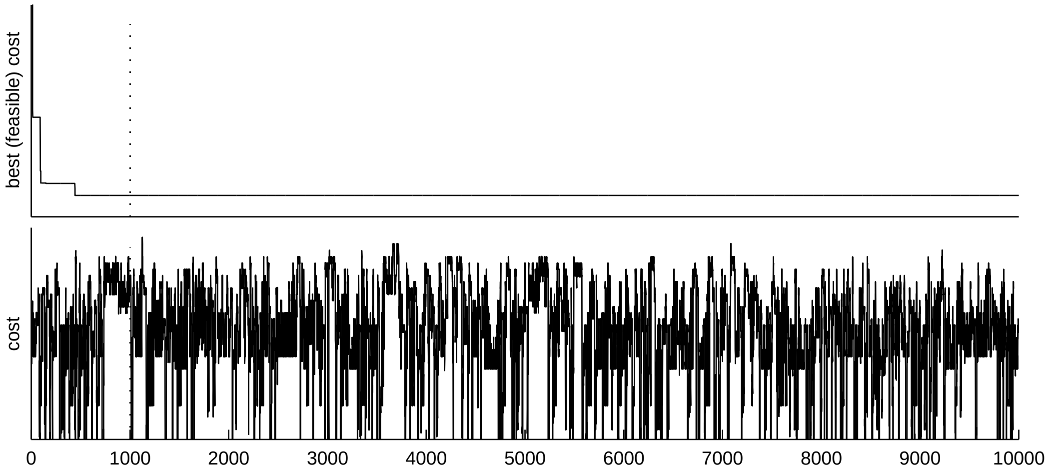


Figure 6.15: Search evolution for `yeast`, with TS1 and random seed 2 (compare with Fig. 6.16).

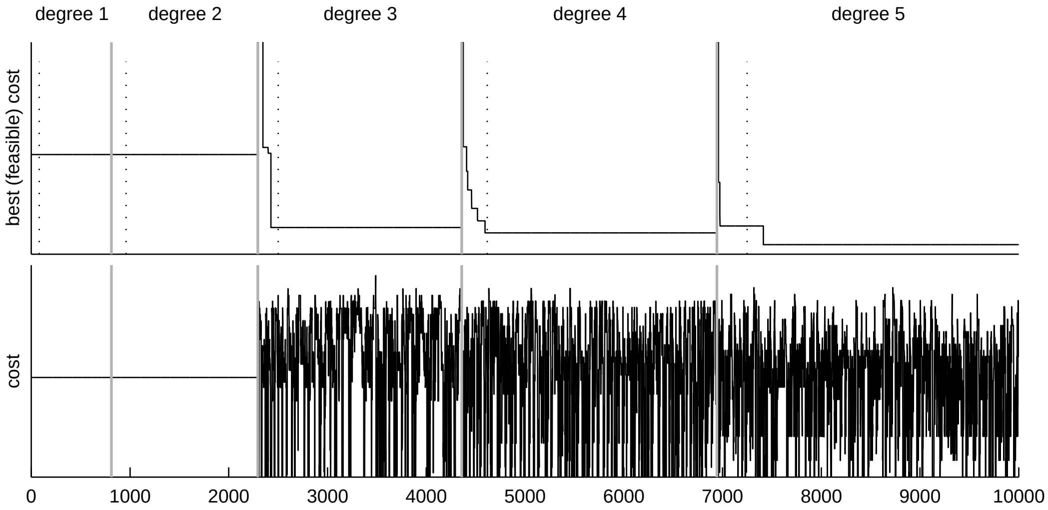


Figure 6.16: Search evolution for `yeast`, with TS2 and random seed 2 (compare with Fig. 6.15).

CHAPTER 7

Conclusions

We recall the overview of this thesis given in Sect. 1.5, in which we presented the initial goals of this work and briefly described the main results obtained. Therein we enumerated three different results. For each one of them we present in the following the global conclusions that we have derived and suggestions for future work.

Boolean mapping

Concerning the transformation of input data into Boolean format and the goals that have been defined in this respect, we consider the proposed procedure to be successful. Our empirical tests have shown that the new procedure is able to generate a mapping of quality equivalent to alternative approaches, but with an execution time that scales very well with the size of the problem. Even for the larger tested problems containing tens of thousands of examples, the mappings have been obtained in a few minutes.

The worthiness of this result is confirmed by the fact that we were able to test developments introduced in other parts of the LAD methodology using the larger data sets, which would have been infeasible otherwise, i.e. if we had not had the possibility of transforming the input data within acceptable computational time.

Future work

In this dissertation, the creation of the Boolean mapping has been carried out under the condition that instances with different class labels are associated with different Boolean images. This is a necessary condition for guaranteeing the coherence of the model. However, this condition may be too strong in the presence of noise in the training data. The procedure that we propose does not take this into account. One way of solving this problem might be to use an alternative way of constructing the mapping that does not necessarily take the class label into account, but rather tries to associate the same Boolean image to examples that lie close to each other in the input space, independently of their class. It might be interesting to implement such a procedure through the use of clustering techniques like k-means (Bottou and Bengio, 1995), for example.

Decomposition of classification tasks

The interestingness of methods allowing two-class classifiers to be applied to multi-class problems is undeniable. With the advent of Support Vector Machines (SVM), which generate two-class classifiers, and the outburst of research activity that they have instigated, this type of method, that decomposes the original problem into sub-problems, has become crucial. By putting side-by-side in the same framework various such decomposition schemes we have been able to analyze their strengths and weaknesses. With an evaluation based on several axes: classification accuracy, training time, sub-model complexity, global model complexity, and number of sub-problems, we have seen that there is a tradeoff between the first of these measures and the rest of them, which is solved differently by each one of the studied methods. The generalization ability (accuracy) is normally ahead of the goals considered in classification tasks. We have observed that a method like ECOC is able to achieve high accuracy, but normally at the cost of long training times and complex models. Similar observations have been made before by other researchers concerning this and other types of methods that employ large numbers of sub-models, like boosting and bagging.

The main result obtained in this framework relates to the proposed decomposition scheme based on pertinent dichotomies. Among those that require the lowest training effort, this method has been shown to provide the best results in accuracy. Hence, we consider this approach to be one of the most interesting ones, as it has proven superior to the standard one-per-class approach that is the default choice whenever two-class classifiers must be applied to multi-class problems. In addition, the number of sub-problems generated by both procedures is comparable, approximately equal to the number of classes.

Future work

The proposed decomposition scheme based on pertinent dichotomies has been tested using decision trees as base classifiers. Due to the promising results, it is naturally of interest to test it with other types of classifiers. One of the advantages of using pertinent dichotomies is the fact that they try to decompose the classification problem into sub-problems that are as simple to solve as possible. Hence, there are some questions that could be asked, such as: to which extent is this scheme better than the others when the capacity of the classifiers is very limited? And what about the opposite case, using very powerful functions, like SVM for example?

Multi-class Boolean-based models

According to our experiments, the accuracy of logic methods in inherently numerical domains (computer vision and speech recognition) does not rival with that of other state-of-the-art classifiers (we base our analysis on decision trees), even if they have comparable accuracy in most of the symbolic domains, i.e. those in which the used attributes have a clear and intuitive meaning.

In certain classification tasks, we are interested in attaining above all a high level of accuracy, and the interpretability of the model brings no significant added value. In other tasks, of which certain medical domains are clear examples, interpretability is a necessity. We can say that, at least in the near future, doctors will always hesitate in allowing diagnostics to be decided by automated learning systems, however accurate they may be, due to the risk usually involved in those decisions. On the contrary, we have shown that in such contexts the interpretability of the model can be a precious asset, even if this implies less good classification results. These models generate independent cause-effect conclusions that can either confirm and support the expert knowledge and convictions, or open new perspectives to the understanding of the studied phenomena, provided that the system is correctly defined and trained. In this respect, the approach followed here, i.e. the proposed multi-class version of LAD which defines patterns that are shared by several classes, provides a perspective that is different from the usual one-class-against-the-others provided by traditional rule-based methods.

Future work

As we have shown, the major advantage of LAD lies in the possibility of using the created models to gain a better understanding of the domain itself. In this perspective, it would be interesting to investigate further possibilities. For example, the domain expert may be interested in studying the relation between two particular classes in the problem, or analyzing the distinguishing features that apply to two (or more) particular classes and not to the others. For this, a more flexible tool could be created, based on the same learning algorithm, but which concentrates only on the disclosure of the most significant patterns that apply to a particular situation, rather than trying to create complete classification models for that matter.

One aspect that should deserve some attention is the comparison of the models created by the multi-class version of LAD and those obtained by using the method in a two-class manner associated with decomposition schemes. Depending on the particular decomposition scheme used, it would be interesting to analyze the obtained patterns, especially the most significant ones, compared to those of the direct multi-class case. For example, using a pairwise-coupling scheme for decomposing the problem we obtain only patterns that differentiate among two of the classes in the problem. This comparison might be particularly interesting in the case where the algorithm of pertinent dichotomies, which directly inspired the multi-class version of LAD, is used for the decomposition. The interest lies not only in the comparison between their most significant patterns, but also between their classification accuracy and the overall size of their models.

Appendix A

Data Sets

The data sets used in the reported experiments can be found in the Repository of Machine Learning Databases of the University of California, at Irvine (Blake and Merz, 1998), whose contents are available through `ftp` access. This repository constitutes the source of experimental data used in the large majority of international scientific publications in the domain of Machine Learning during the last decade. Other repositories include the Machine Learning network (MLnet) Online Information Service (`http://www.mlnet.org`) and (Bay, 1999).

All the twenty-two data sets used in the reported experiments relate to real situations, none of them is synthetic. They originate in various domains, like medicine (five data sets), image recognition (four data sets), socioeconomics, politics, biology, chemistry, and others.

Below there are two summary tables, one containing the data sets with two classes, and the other with the sets of more than two classes. The column "Missing Values?" indicates whether the data misses some attribute values for certain instances. A brief description of each set is provided afterwards.

The `abalone` data set has been slightly modified. Originally, it contains twenty-eight classes which are very unbalanced. While the most frequent class contains 689 examples, there are six classes with only one or two examples. The latter have been removed from the data set, since otherwise the train/test partitioning would result in certain learning tasks where one of the classes would be empty. The modified data set contains twenty-two classes and 4168 examples, instead of the 4177 ones in the original set.

Two-class data sets

Code name	Number of classes	Number of instances	Number of attributes	Archive size	Missing values?
adult	2	48842	14	5193Kb	yes
breast-cancer	2	699	9	20Kb	yes
credit	2	690	15	33Kb	yes
heart-disease	2	303	13	15Kb	yes
hepatitis	2	155	19	7Kb	yes
ionosphere	2	351	34	75Kb	
mushroom	2	8124	22	373Kb	yes
pi-diabetes	2	768	8	23Kb	
spambase	2	4601	57	696Kb	
voting	2	435	16	18Kb	yes

More-than-two-class data sets

Code name	Number of classes	Number of instances	Number of attributes	Archive size	Missing values?
abalone	22	4177	9	193Kb	
dermatology	6	366	34	36Kb	yes
ecoli	8	336	7	10Kb	
glass	6	214	9	16Kb	
letter	26	20000	16	700Kb	
optdigits	10	5620	64	824Kb	
pendigits	10	10992	16	728Kb	
segmentation	7	2310	19	374Kb	
soybean	19	683	35	65Kb	yes
vowel	11	990	10	88Kb	
wine	3	178	13	10Kb	
yeast	10	1484	8	84Kb	

Data set descriptions

The “donation” field in the following descriptions correspond to the donor and date at which the data set has been contributed to the repository.

`abalone` [Abalone]

Task: Predicting the age of abalone from physical measurements.
Source: Marine Research Laboratories, Taroona, Australia.
Donation: Sam Waugh, December 1995.

adult [Adult]

Task: Predicting whether a given north-american adult earns more than $50 000 a year based on attributes such as age, working class, education, hours of works per week, etc.

Source: US Census Bureau.

Donation: Ronny Kohavi and Barry Becker, May 1996.

breast-cancer [Wisconsin Breast Cancer]

Task: Predicting breast cancer diagnostic (benign/malignant) from medical observations.

Source: O. L. Mangasarian and W. H. Wolberg: "Cancer diagnosis via linear programming", SIAM News, Volume 23, Number 5, September 1990, pp. 1–18.

Donation: Olvi Mangasarian, July 1992.

credit [Credit Approval]

Task: Predicting the approval/denial of credit card applications based on (undisclosed) attributes.

Source: confidential.

Donation: J. R. Quinlan, late 1980's.

dermatology [Dermatology]

Task: Determining the type of erythemato-squamous disease from medical observations.

Source: Nilsel Ilter, Gazi University, Ankara, Turkey and H. Altay Guvenir, Bilkent University, Ankara, Turkey.

Donation: H. Altay Guvenir, January 1998

ecoli [Protein Localization Sites]

Task: Predicting the Cellular Localization Sites of Proteins.

Source: Kenta Nakai, Osaka University, Japan.

Donation: Paul Horton, September 1996.

glass [Glass Identification]

Task: Identifying types of glass from physical measurements.

Source: B. German, Central Research Establishment, Berkshire.

Donation: Vina Spiehler, September 1987.

heart-disease [Cleveland Heart Disease]

Task: Predicting the presence or absence of a heart disease from medical observations.

Source: V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano.

Donation: David W. Aha, July 1988.

`hepatitis` [Hepatitis]

TASK: Predicting hepatitis patient survival from medical observations.

SOURCE: unknown.

DONATION: G.Gong, November 1988.

`ionosphere` [Ionosphere]

TASK: Predicting the presence/absence of structures in the ionosphere based on radar signals.

SOURCE: Space Physics Group, Johns Hopkins University.

DONATION: Vince Sigillito, 1989.

`letter` [Letter Image Recognition]

TASK: Identifying the 26 capital letters in the English alphabet, using pre-processed image data. Pre-processing is based on statistical moments and edge counts.

SOURCE: David J. Slate, Odesta Corporation, IL.

DONATION: David J. Slate, January 1991.

`mushroom` [Mushroom]

TASK: Predicting whether mushrooms in the Agaricus and Lepiota Family are edible or poisonous, based on physical descriptions.

SOURCE: Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf.

DONATION: Jeff Schlimmer, April 1987.

`optdigits` [Optical Recognition of Handwritten Digits]

TASK: Recognizing handwritten digits, using normalized bitmaps of scanned images.

SOURCE: E. Alpaydin and C. Kaynak, Bogazici University, Turkey.

DONATION: E. Alpaydin, July 1998.

`pendigits` [Pen-Based Recognition of Handwritten Digits]

TASK: Recognizing handwritten digits created with a pressure sensitive tablet. The attributes are based on the x-y coordinate position of the pen over time.

SOURCE: E. Alpaydin and F. Alimoglu.

DONATION: E. Alpaydin, July 1998.

`pi-diabetes` [Pima Indians Diabetes]

TASK: Identifying the presence/absence of diabetes in patients, from medical observations.

SOURCE: National Institute of Diabetes and Digestive and Kidney Diseases.

DONATION: Vincent Sigillito, May 1990.

segmentation [Image Segmentation]

TASK: Identifying image contents (brickface, sky, foliage, etc.).

SOURCE: Vision Group, University of Massachusetts.

DONATION: Carla Brodley, November 1990.

soybean [Large Soybean]

TASK: Identifying different types of soybean diseases.

SOURCE: R.S. Michalski and R.L. Chilausky, "Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis", International Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, 1980.

DONATION: Ming Tan and Jeff Schlimmer, July 1988.

spambase [Spam E-mail]

TASK: Identifying e-mail messages as containing spam (unsolicited commercial advertising) or not, based on statistical parameters of the text and selected keyword frequency.

SOURCE: Mark Hopkins, Erik Reeber, George Forman, Jaap Suermondt, Hewlett-Packard Labs, Palo Alto, CA.

DONATION: George Forman, July 1999.

voting [Congressional Voting]

TASK: Predicting the political affiliation (democrat/republican) of U.S. House of Representatives Congressmen based on 16 key votes.

SOURCE: Congressional Quarterly Almanac, 98th Congress, 2nd session 1984, Volume XL: Congressional Quarterly Inc. Washington, D.C., 1985.

DONATION: Jeff Schlimmer, April 1987.

vowel [Vowel Recognition]

TASK: Speaker independent recognition of the eleven steady state vowels of British English using a specified training set of lpc derived log area ratios.

SOURCE: David Deterding, Mahesan Niranjan, and Tony Robinson.

DONATION: unknown donor, late 1980's.

wine [Wine Recognition]

TASK: The data contains the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The task is to identify the cultivar.

SOURCE: M. Forina et al., "PARVUS - An Extendible Package for Data Exploration, Classification and Correlation". Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno, 16147 Genoa, Italy.

DONATION: Stefan Aeberhard, July 1991.

`yeast` [Protein Localization Sites]

TASK:	Predicting the Cellular Localization Sites of Proteins.
SOURCE:	Kenta Nakai, Osaka University, Japan.
DONATION:	Paul Horton, September 1996.

Appendix B

Statistical tests

This appendix describes the general procedure used throughout this thesis in order to compare different algorithms empirically, and evaluating the statistical significance of the results.

The goal of statistical tests

For a given classification task, deciding whether two learning algorithms A and B have different performance is a procedure that requires some care. A simple test might consist in taking two non-intersecting sets of data representative of the classification task treated, training a classifier on one set with each learning algorithm, testing the obtained classifiers on the other set, and observe which one provides better accuracy. In practice, there are several factors that may render such a procedure not conclusive enough about the intrinsic performance of each algorithm.

Dietterich (1998) cites four different sources of variation that may cause different accuracies for A and B and that are not directly ascribable to the algorithms themselves:

- the particular choice of the testing data;
- the particular choice of the training data;
- internal randomness of the algorithm itself, in case it is not deterministic;
- random classification error (noise) in the testing data.

This means that if the set of testing data or the set of training data would have been chosen differently, or if the initial random seed for the algorithms would have been different, or if the testing data would have contained less noise, the performances of each one of the algorithms would perhaps have been different. So, an effective test must allow these sources of variation to be filtered out and it must be able to isolate the actual performance of the algorithms. This can be achieved by a series of repeated runs of the algorithms using different training and testing data sets and different random seeds, together with a statistical test for significant differences[1].

Note, however, that if a large quantity of training and testing examples are available, and if the tested algorithms are deterministic, a single run may be sufficient to obtain an accurate comparison. In this case, there are other tests, alternative to the one described

We consider that *under the null hypothesis, there is no difference between the algorithms.* There are two types of error that can be produced by a test:

Type I error: the test rejects the null hypothesis when it is true, i.e. it considers that the algorithms have different performance when in reality they have not;

Type II error: the test accepts the null hypothesis when it is false, i.e. it is not capable of detecting existing differences. A test with low type II error is normally said to be powerful, because it is able to detect differences when they exist.

Dietterich (1998) analyzes five different statistical tests and recommends the use of the 5×2*cv test*, which has acceptable type I error and reasonable power. In this dissertation, we have adopted this test (an evolution of it, to be more precise), which is briefly described below.

The 5×2cv test

The 5×2cv test consists in doing, for each algorithm, five replications of a *two-fold cross-validation test* with a given data set, and then applying a statistical test on the obtained results in order to accept or reject the null hypothesis.

A two-fold cross validation test involves splitting the data into two equal-sized parts (folds), training the algorithm on one fold, testing it on the other, and then repeating the procedure with the reversed folds. In each replication, a different random seed is used to split the data into two folds, which should respect the class distribution of the original, entire set (i.e. each class subset is split into halves, each sent to a different fold). This procedure generates two sequences of accuracy values, one for each algorithm, and each containing ten values. The goal of a statistical test is to verify whether the values in these two sequences have been drawn from the same distribution with a certain level of confidence, in which case the null hypothesis is accepted. In all the experiments that we made, we used the confidence level 0.95.

Let $p_i^{(j)}$ be the difference between the accuracy values of the classifiers generated by algorithms A and B, where $i = 1, \ldots, 5$ represents a replication and $j = 1, 2$ represents a fold. $\overline{p}_i$ and s_i^2 are respectively the estimated average and variance on replication i, computed as follows:

$$\begin{aligned}\overline{p}_i &= (p_i^{(1)} + p_i^{(2)})/2 \\ s_i^2 &= (p_i^{(1)} - \overline{p}_i)^2 + (p_i^{(2)} - \overline{p}_i)^2\end{aligned}$$

[1] In certain particular situations, a single run of each algorithm may be sufficient to obtain an accurate comparison. For example, the Hoeffding's inequality (Hoeffding, 1963) can be used to establish a minimal number of testing examples allowing a certain confidence interval to be defined around the error produced by a classifier.

Using well-known approximations of statistical distributions and various (unrealistic) independence assumptions, Dietterich shows that, under the null hypothesis, the statistic:

$$\tilde{t} = \frac{p_1^{(1)}}{\sqrt{\frac{1}{5}\sum_{i=1}^{5} s_i^2}} \tag{B.1}$$

has approximately a t-distribution with 5 degrees of freedom (we omit the demonstration, which can be found in the publication). This means that we can reject the null hypothesis that the algorithms are equivalent with 95% of confidence if $\tilde{t}$ is greater than 2.571.

Alpaydin (1999) has observed that the choice of the value $p_1^{(1)}$ for the numerator of Eq. (B.1) is arbitrary, and that choosing one of the nine remaining possible values of $p_i^{(j)}$ may lead to different results. So, he has proposed a different version of the test, which makes an average over the ten possibilities for the numerator. Using additional well-known approximations of statistical distributions and (unrealistic) independence assumptions, Alpaydin derives the following expression from Eq. (B.1):

$$f = \frac{\sum_{i=1}^{5}\sum_{j=1}^{2}\left(p_i^{(j)}\right)^2}{2\sum_{i=1}^{5} s_i^2} \tag{B.2}$$

which approximately follows a F-distribution with 10 and 5 degrees of freedom (we also omit this derivation, which can be found in the mentioned publication). Hence, with a confidence level of 0.95, we reject the hypothesis that the two algorithms are equivalent if f is greater than 4.74. This is the test that is used throughout the current dissertation. Moreover, in every situation where algorithms must be compared, we do it on several classification tasks (described in Appendix A). The statistical test is done independently for each task.

Applying the test to several algorithms

Above we have described how to compare two algorithms, but in many cases we need to compare several algorithms simultaneously. For this, we have adopted the following procedure:

1. use the 5×2cv approach to test each algorithm;
2. select the best algorithm, which is the one with the highest mean over the sequence of ten accuracy values;
3. for each one of the other algorithms, apply the statistical test of Eq. (B.2) in order to compare it with the best; if the null hypothesis is accepted, we consider it to belong to the group of the best.

This means that the statistical test divides the results of various algorithms into at most two groups: the group of the best (not significantly different from the best) and the group of the others (significantly different from the best).

Testing other quantities

The given description assumed the test to be applicable to measure the classification accuracy. In several situations of this thesis, different algorithms were also compared according to other quantities, like the execution time or the complexity of the generated classification models, for example. In these cases, the 5×2cv test was applied in the same manner as above.

Bibliography

Aarts, E. and Korst, J. (1989). *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons, Chichester.

Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–66.

Almuallim, H. and Dietterich, T. G. (1991). Learning with many irrelevant features. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, pages 547–552.

Almuallim, H. and Dietterich, T. G. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1–2):279–306.

Alpaydin, E. (1999). Combined 5x2cv F test for comparing supervised classification learning algorithms. *Neural Computation*, 11(8):1885–1892.

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1/2):105–139.

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

Bergadano, F., Matwin, S., Michalski, R. S., and Zhang, J. (1992). Learning two-tiered descriptions of flexible contexts: The POSEIDON system. *Machine Learning*, 8(1):5–43.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford.

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

Bloedorn, E. and Michalski, R. S. (1991). Constructive induction from data in AQ17-DCI: Further experiments. MLI 91-12, Artificial Intelligence Center, George Mason University.

Boros, E., Hammer, P. L., Ibaraki, T., Kogan, A., Mayoraz, E., and Muchnik, I. (1996). An implementation of Logical Analysis of Data. RRR 22-96, RUTCOR–Rutgers University Center For Operations Research, `ftp://rutcor.rutgers.edu/pub/rrr/reports96/22.ps.gz`. Appeared as (Boros et al., 2000).

Boros, E., Hammer, P. L., Ibaraki, T., Kogan, A., Mayoraz, E., and Muchnik, I. (2000). An implementation of Logical Analysis of Data. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):292–306.

Boros, E., Hammer, P. L., Kogan, A., Mayoraz, E., and Muchnik, I. (1994). Logical Analysis of Data — Overview. RTR 1-94, RUTCOR–Rutgers University Center For Operations Research, `ftp://rutcor.rutgers.edu/pub/LAD/LAD-ov.ps`.

Boros, E., Ibaraki, T., and Makino, K. (1999). Logical analysis of binary data with missing bits. *Artificial Intelligence*, 107:219–263.

Bottou, L. and Bengio, Y. (1995). Convergence properties of the k-means algorithms. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems 7*, pages 585–592. The MIT Press.

Breiman, L. (1994). Bagging predictors. TR 421, Department of Statistics, University of California, Berkeley, CA.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth and Brooks/Cole, Monterey, CA.

Brézellec, P. and Soldano, H. (1998). Tabata: a learning algorithm performing a bidirectional search in a reduced search space using a Tabu strategy. In Prade, H., editor, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 420–424. John Wiley & Sons.

Brown, F. M. (1990). *Boolean Reasoning: The Logic of Boolean Equations*. Kluwer Academic Publishers, Boston.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):955–974.

Castellano, P. J., Slomka, S., and Sridharan, S. (1997). Telephone based speaker recognition using multiple binary classifier and gaussian mixture models. In *Proceedings of ICASSP'97*, volume 2, pages 1075–1078. IEEE Computer Society Press.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370.

Cestnik, B., Kononenko, I., and Bratko, I. (1987). Assistant 86: a knowledge-elicitation tool for sophisticated users. In Bratko, I. and Lavrac, N., editors, *Progress in Machine Learning: Proceedings of the 2nd European Working Session on Learning*, pages 31–45.

Cheeseman, P. and Stutz, J. (1996). Bayesian classification (autoclass): Theory and results. In Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors, *Advances in Knowledge Discovery and Data Mining*, chapter 6, pages 153–180. AAAI Press / The MIT Press.

Chvátal, V. (1979). A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235.

Clark, P. and Boswell, R. (1991). Rule induction with CN2: Some recent improvements. In Kodratoff, Y., editor, *Proceedings of the 5th European Working Session on Learning (EWSL-91)*, volume 482 of *Lecture Notes in Computer Science*, pages 151–163. Springer.

Clark, P. and Niblett, T. (1989). The CN2 algorithm. *Machine Learning*, 3(4):261–284.

Cohen, W. W. (1995). Fast effective rule induction. In *Machine Learning: Proceedings of the 12th International Conference*, pages 115–123,.

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Schrijver, A. (1998). *Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley & Sons, New York.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Crama, Y., Hammer, P. L., and Ibaraki, T. (1988). Cause-effect relationships and partially defined Boolean functions. *Annals of Operations Research*, 16:299–326.

De Jong, K. A. and Spears, W. M. (1991). Learning concept classification rules using genetic algorithms. In *Proceedings of IJCAI-91*, pages 651–656. Morgan Kaufmann.

De Jong, K. A., Spears, W. M., and Gordon, D. F. (1993). Using genetic algorithms for concept learning. *Machine Learning*, 13(2/3):161–188.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):1–22.

Dietterich, T. G. and Bakiri, G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, pages 572–577. AAAI Press / MIT Press.

Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286.

Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In Prieditis, A. and Russel, S., editors, *Machine Learning: Proceedings of the Twelfth International Conference*, San Francisco, CA. Morgan Kaufman Publishers.

Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis*. John Wiley & Sons.

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors (1996). *Advances in Knowledge Discovery and Data Mining*. AAAI Press / The MIT Press.

Fensel, D. and Wiese, M. (1993). Refinement of rule sets with JoJo. In Brazdil, P., editor, *Proceedings of the 6th European Conference on Machine Learning (ECML-93)*, Lecture Notes in Artificial Intelligence, pages 378–383, Berlin. Springer.

Fensel, D. and Wiese, M. (1994). From JoJo to Frog: Extending a bi-directional search strategy to a more flexible three-directional search. In Globig, C. and Althoff, K.-D., editors, *Beiträge zum 7. Fachgruppentreffen Maschinelles Lernen*, pages 37–44.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–286.

Friedman, J. H. (1996). Another approach to polychotomous classification. Technical report, Stanford University, Department of Statistics, `http://www-stat.stanford.edu/reports/friedman/poly.ps.Z`.

Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, New York.

Genoud, D., Moreira, M., and Mayoraz, E. (1998). Text dependent speaker verification using binary classifiers. In *Proceedings of ICASSP'98*, volume 1, pages 129–132. IEEE Computer Society Press.

Gersho, A. and Gray, R. M. (1992). *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Boston.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549.

Glover, F. (1989). Tabu search — part I. *ORSA Journal on Computing*, 1(3):190–206.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

Glover, F., Laguna, M., Taillard, E., and de Werra, D., editors (1993). *Annals of Operations Research*, volume 41. J. C. Baltzer. Journal issue about Tabu Search.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts.

Grefenstette, J. J. (1989). A system for learning control strategies with genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 183–190. Morgan Kaufman.

Hammer, P. L. (1986). Partially defined Boolean functions and cause-effect relationships. Presented at the International Conference on Multi-Attribute Decision Making Via OR-Based Expert Systems, University of Passau, Germany. Appeared as (Crama et al., 1988).

Hansen, P. (1986). The steepest ascent, mildest descent heuristic for combinatorial programming. Presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy.

Hastie, T. and Tibshirani, R. (1996). Classification by pairwise coupling. Technical report, Stanford University and University of Toronto, `ftp://utstat.toronto.edu/pub/tibs/coupling.ps`.

Haykin, S. (1994). *Neural Networks. A Comprehensive Foundation*. Macmillan College Publishing, New York.

Hertz, A. and de Werra, D. (1990). The tabu search metaheuristic: how we used it. *Annals of Mathematics and Artificial Intelligence*, 1:111–121.

Hertz, A., Taillard, E., and de Werra, D. (1995). A tutorial on tabu search. In *Proc. of Giornate di Lavoro AIRO'95, (Entreprise Systems: Management of Technological and Organizational Changes)*, pages 13–24.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An artificial intelligence approach*, volume II, chapter 20, pages 593–623. Morgan Kaufmann.

Hunt, E. B., Marin, J., and Stone, P. T. (1966). *Experiments in Induction*. Academic Press, New York.

Hussain, F., Liu, H., Tan, C. L., and Dash, M. (1999). Discretization: An enabling technique. Technical Report TRC6/99, National University of Singapore.

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice Hall, Engelwood Cliffs.

James, G. and Hastie, T. (1997). The error coding method and PiCTs. *Journal of Computational and Graphical Statistics*, 7(3):377–387.

Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13(2/3):189–228.

John, G. H., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In Cohen, W. W. and Hirsh, H., editors, *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129, San Francisco, CA. Morgan Kaufman.

Kira, K. and Rendell, L. A. (1992). A practical approach to feature selection. In *Proceedings of the 9th International Conference on Machine Learning*, pages 249–256. Morgan Kaufmann.

Kirkpatrick, S., Gelatt, D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

Kohonen, T. (1995). *Self-Organizing Maps*. Springer-Verlag, Berlin.

Kong, E. B. and Dietterich, T. G. (1995). Error-correcting output coding corrects bias and variance. In *Proceedings of the 12th International Conference on Machine Learning*, pages 313–321, San Francisco, CA. Morgan Kaufmann.

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of RELIEF. In *Proceedings of the European Conference on Machine Learning*, pages 171–182. Springer.

Kononenko, I., Bratko, I., and Roskar, E. (1984). Experiments in automatic learning of medical diagnostic rules. Technical report, Faculty of Electircal Engineering, E. Kardelj University.

Kononenko, I. and Kovačič, M. (1992). Learning as optimization: Stochastic generation of multiple knowledge. In *Proceedings of the 9th International Conference on Machine Learning*, pages 257–262. Morgan Kaufmann.

Lavrač, N. and Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

Lim, T.-S., Loh, W.-Y., and Shih, Y.-S. (2000). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40(3):203–228.

Mayoraz, E. (1995). C++ tools for Logical Analysis of Data. RTR 1-95, RUTCOR–Rutgers University's Center For Operations Research.

Mayoraz, E. and Moreira, M. (1997). On the decomposition of polychotomies into dichotomies. In Fisher, D. H., editor, *Proceedings of the 14th International Conference on Machine Learning*, pages 219–226. Morgan Kaufmann.

Mayoraz, E. and Moreira, M. (1999). Combinatorial approach for data binarization. In Zytkow, J. and Rauch, J., editors, *Principles of Data Mining and Knowledge Discovery: third european conference; proceedings / PKDD'99*, volume 1704 of *Lecture Notes in Artificial Intelligence*, pages 442–447. Springer.

Mendelson, E. (1970). *Theory and Problems of Boolean Algebra and Switching Circuits*. Schaum's Outline Series. McGraw-Hill, New York.

Michalski, R. S. (1969). On the quasi-minimal solution of the general covering problem. In *Proceedings of the Fifth International Symposium on Information Processing*, volume A3 (Switching circuits), pages 125–128.

Michalski, R. S. (1975). Variable-valued logic and its applications to pattern recognition and machine learning. In Rine, D. C., editor, *Computer Science and Multiple-Valued Logic Theory and Applications*, pages 506–534. North-Holland.

Michalski, R. S., Mozetic, I., Hong, J., and Lavrac, N. (1986). The multipurpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, pages 1041–1045.

Minsky, M. L. and Papert, S. A. (1988). *Perceptrons. An Introduction to Computational Geometry*. The MIT Press, Cambridge, MA, expanded edition.

Mitchell, T. (1997). *Machine Learning*. McGraw Hill.

Mitchell, T. M. (1978). *Version spaces: An approach to concept learning*. PhD thesis, Stanford University.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18(2):203–226.

Mitchell, T. M. (1991). The need for bias in learning generalizations. In Shavlik, J. and Dietterich, T., editors, *Readings in Machine Learning*, pages 184–191. Morgan Kaufmann.

Mladenić, D. (1993). Combinatorial optimization in inductive concept learning. In *Proceedings of the 10th International Conference on Machine Learning*, pages 205–211. Morgan Kaufmann.

Moreira, M., Hertz, A., and Mayoraz, E. (1999). Data binarization by discriminant elimination. In Bruha, I. and Bohanec, M., editors, *Proceedings of the ICML-99 Workshop: From Machine Learning to Knowledge Discovery in Databases*, pages 51–60.

Moreira, M. and Mayoraz, E. (1998). Improved pairwise coupling classification with correcting classifiers. In Nédellec, C. and Rouveirol, C., editors, *Machine Learning: ECML-98*, volume 1398 of *Lecture Notes in Artificial Intelligence*, pages 160–171. Springer.

Pagallo, G. and Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5(1):71–99.

Peterson, W. W. and Weldon, E. J. (1972). *Error-correcting codes.* The MIT Press, Cambridge, Massachussetts, second edition.

Pfahringer, B. (1995). Compression-based discretization of continuous attributes. In Prieditis, A. and Russell, S., editors, *Proceedings of the 12th International Conference on Machine Learning (ICML'95)*, Los Altos/Palo Alto/San Francisco. Morgan Kaufmann.

Pijls, W. and Bioch, J. C. (1999). Mining frequent itemsets in memory-resident databases. In Postma, E. and Gyssens, M., editors, *Proceedings of the 11th Belgium/Netherlands Conference on Artificial Intelligence (BNAIC 99)*, pages 75–82.

Pijls, W. and Potharst, R. (2000). Classification and target group selection based upon frequent patterns. `http://www.few.eur.nl/few/people/pijls/classtarget.ps`.

Platt, J. C., Cristianini, N., and Shawe-Taylor, J. (2000). Large margin DAGs for multiclass classification. In Solla, S. A., Leen, T. K., and Muller, K.-R., editors, *Advances in Neural Information Processing Systems 12*, pages 547–553. MIT Press.

Plotkin, G. D. (1971). A further note on inductive generalisation. In Meltzer, B. and Michie, D., editors, *Machine Intelligence*, volume 6, pages 101–124. Elsevier North-Holland, New York.

Price, D., Knerr, S., Personnaz, L., and Dreyfus, G. (1995). Pairwise neural network classifiers with probabilistic outputs. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems 7*, pages 1109–1116. The MIT Press.

Quinlan, J. R. (1979). Discovering rules by induction from large collections of examples. In Michie, D., editor, *Expert Systems in the Micro Electronic Age*, pages 168–201. Edinburgh University Press, Edinburgh.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine learning: An Artificial Intelligence Approach*, pages 463–482. Morgan Kauffman.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5:239–266.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

Quinlan, J. R. (1996). Bagging, boosting, and C4.5. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'96)*, pages 725–730. Morgan Kaufmann.

Reeves, C. R. (1996). Modern heuristic techniques. In Rayward-Smith, V. J., Osman, I. H., Reeves, C. R., and Smith, G. D., editors, *Modern Heuristic Search Methods*, chapter 1, pages 1–25. John Wiley & Sons, Chichester.

Ripley, B. D. (1996). *Pattern recognition and Neural networks*. Cambridge University Press, Cambridge, UK.

Rivest, R. L. (1987). Learning decision lists. *Machine Learning*, 2(3):229–246.

Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan Books, Washington, DC.

Rudasi, L. and Zahorian, S. A. (1991). Text-independent talker indentification with neural networks. In *Proceedings of ICASSP'91*, volume 1, pages 389–392. IEEE Computer Society Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., McClelland, J. L., and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–364. MIT Press, Cambridge, MA.

Rymon, R. (1993). An SE-tree based characterization of the induction problem. In Utgoff, P. E., editor, *Proceedings of the 10th International Conference on Machine Learning*, pages 268–275. Morgan Kaufmann.

Sahami, M. (1995). Learning classification rules using lattices. In Lavrac, N. and Wrobel, S., editors, *Proceedings of the 8th European Conference on Machine Learning (ECML-95)*, pages 343–346. Springer-Verlag.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

Schölkopf, B., Burges, C. J. C., and Smola, A. J., editors (1999). *Advances in Kernel Methods — Support Vector Learning*. MIT Press, Cambridge, Massachusetts.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423 and 623–656.

Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. In Bundy, A., editor, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, volume 1, pages 422–425. Morgan Kaufman.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer, New York.

Venturini, G. (1993). SIA: A supervised inductive algorithm with genetic search for learning attribute based concepts. In *Proceedings of the 7th European Conference on Machine Learning*, pages 280–296.

Vere, S. (1975). Induction of concepts in the predicate calculus. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 351–356.

Webb, G. I. (1994). Recent progress in learning decision lists by prepending inferred rules. In *Proceedings of the Second Singapore International Conference on Intelligent Systems*, pages B280–B285.

Webb, G. I. and Agar, J. W. M. (1992). Inducing diagnostic rules for glomerular disease with the DLG machine learning algorithm. *Artificial Intelligence in Medicine*, 4:3–14.

Weisstein, E. W. (1996). Eric Weisstein's world of mathematics. `http://mathworld.wolfram.com`.

Wilson, S. W. (1987). Quasi-darwinian learning in a classifier system. In *Proceedings of the 4th International Workshop on Machine Learning*. Morgan Kaufman.

Zahorian, S. A., Silsbee, P., and Wang, X. (1997). Phone classification with segmental features and a binary-pair partitioned neural network classifier. In *Proceedings of ICASSP'97*, volume 2, pages 1011–1014. IEEE Computer Society Press.

Index

Curriculum Vitae

Luis Miguel Moreira

Date of birth: August 31, 1971
Nationality: Portuguese

Education

1989, Colégio Internato dos Carvalhos, Portugal: three-year technical education in Management Information Science.

1995, Universidade do Minho, Portugal: five-year degree on Systems and Computer Science Engineering. Specialized in Management Information Science.

Professional experience

Dalle Molle Institute for Perceptual Artificial Intelligence (IDIAP), Martigny, Switzerland, 1996–to date:

> Preparation of a PhD. dissertation in the domain of Machine Learning.

IDIAP, 1996–1998:

> Participation in an project in partnership with other European institutions and companies, and under contract of the European Space Agency, which aimed at preparing a hands- and eyes-free computer terminal using commercially available products for Automatic Speech Recognition and Synthesis.

IDIAP, 1995:

> Study and experimentation of several methods for training Artificial Neural Networks.
>
> Development of the software of an industrial prototype for quality control of wrist watch production, using Artificial Neural Networks and Automatic Image Recognition techniques.

Universidade do Minho, Department of Computer Science, Portugal, Spring 1994:

> Training period involving the modeling of a Groupware application for decision-making, using a methodology of Object-Oriented Analysis and Design.

Publications

- Eddy Mayoraz and Miguel Moreira. Combinatorial approach for data binarization. In J. Zytkow and J. Rauch, editors, *Principles of Data Mining and Knowledge Discovery: third european conference; proceedings / PKDD'99*, volume 1704 of *Lecture Notes in Artificial Intelligence*, pages 442–447. Springer, 1999.

- Miguel Moreira, Alain Hertz, and Eddy Mayoraz. Data binarization by discriminant elimination. In Ivan Bruha and Marco Bohanec, editors, *Proceedings of the ICML-99 Workshop: From Machine Learning to Knowledge Discovery in Databases*, pages 51–60, 1999.

- Miguel Moreira and Eddy Mayoraz. Improved pairwise coupling classification with correcting classifiers. In Claire Nédellec and Céline Rouveirol, editors, *Machine Learning: ECML-98*, volume 1398 of *Lecture Notes in Artificial Intelligence*, pages 160–171. Springer, 1998.

- Dominique Genoud, Miguel Moreira, and Eddy Mayoraz. Text dependent speaker verification using binary classifiers. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing — ICASSP'98*, volume I, pages 129–132. IEEE, 1998.

- Eddy Mayoraz and Miguel Moreira. On the decomposition of polychotomies into dichotomies. In Douglas H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 219–226. Morgan Kaufmann, 1997.

- Miguel Moreira, Emile Fiesler, and Gianni Pante. Image classification by neural networks for the quality control of watches. In R. Soto, J. M. Sanchez, M. Campbell, and F. J. Cantu, editors, *Proceedings ISAI /IFIS 1996*, pages 141–149. ITESM, 1996.

- Miguel Moreira and Emile Fiesler. Neural networks with adaptive learning rate and momentum terms. IDIAP-RR 4, IDIAP, Martigny, Switzerland, October 1995.