OPTIMIZATION IN LOGICAL ANALYSIS OF DATA

BY TIBÉRIUS OLIVEIRA BONATES

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Operations Research

Written under the direction of

Peter L. Hammer and Alexander Kogan

and approved by

New Brunswick, New Jersey

October, 2007

ABSTRACT OF THE DISSERTATION

Optimization in Logical Analysis of Data

by Tibérius Oliveira Bonates

Dissertation Directors: Peter L. Hammer and Alexander Kogan

Logical Analysis of Data (LAD) is a machine learning/data mining methodology that combines ideas from areas such as Boolean functions, optimization and logic. In this thesis, we focus on the description and the application of novel optimization models to the construction of improved and/or simplified LAD models of data. We address the construction of LAD classification models, proposing two alternative ways of generating patterns, or rules. First, we show how to construct LAD models based on patterns of maximum coverage. We show, through a series of computational experiments, that such models are as good as, if not better than those obtained with the standard LAD implementation and other machine learning methods, while requiring a much simpler calibration for optimal performance. We formulate the problem of finding the most suitable LAD model as a large linear program, and show how to solve it using column generation. For the subproblem phase, we describe a branch-and-bound algorithm, whose performance is significantly superior to that of a commercial integer programming solver. The LAD models produced by this algorithm are virtually parameter-free and practically as accurate as the calibrated models obtained with other machine learning methods. Finally, we propose a novel regression algorithm that extends the LAD methodology for the case of a numerical outcome and show that it constitutes an attractive alternative to other regression methods in terms of performance and flexibility of use.

Acknowledgements

I would like to thank my advisors Peter L. Hammer and Alexander Kogan for their patience and constant support in writing this thesis, for their kind advices, and for all that I learned from them. I will be forever grateful.

I am obliged to Gabriela Alexe, Endre Boros, Vladimir Gurvich, Nelson Maculan, and Andras Prekopa for being kind enough to participate in this thesis' committee.

I am thankful to the people at Dash Optimization Inc. for the internship experience I had in their New Jersey office, and to DIMACS for their financial support, which – over the years – made possible a major part of the research presented in this thesis. Also, the partial financial support of the Cleveland Clinic Foundation, NIH, and NSF was of great importance and was much appreciated.

I would also like to mention the names of other people who have, over the last few years, offered me so much help and kindness: David E. Axelrod, Noam Goldberg, Aritanan Gruber, Anca Hammer, Terry Hart, José Koiller, Irina Lozina and Vadim Lozin, Marcin Kamiński, Carlos and Janaina Oliveira, Fábio Oliveira, Eduardo Pinheiro and Izabel Martins, Luiz Ramos, Clare Smietana, Gabriel Tavares, Rodrigo Toso, and Cem Iyigun. Many thanks! Also, many thanks to my professors from Brazil who supported my intention of coming to RUTCOR. In particular, Nelson Maculan, Plácido Pinheiro, Marcos Negreiros and Maria Helena Jardim.

I thank my family in Brazil, for their emotional support and for bearing the distance and the limited contact that we have had over the last five years. They have always offered me so much and asked me for so little. I can only begin to express my gratitude to them.

Lastly, however mostly, I thank my wife, Mara, who has been a constant source of inspiration to me, and whose unlimited patience, kindness, and companionship were invaluable for the conclusion of this thesis. I will never be able to thank her enough.

Dedication

I dedicate this thesis to the memory of Peter L. Hammer. Peter was a brilliant and generous man, who taught me so much, and who is greatly missed. Having his guidance through my studies at RUTCOR was an honor and a pleasure. I express here my profound respect and gratitude to him.

I would also like to dedicate this thesis to the memory of my good old friend Elder Magalhães Macambira, who sadly left us on August 29, 2007. Elder was a good friend, who will also be missed a lot.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis we describe and investigate the performance of novel optimization applications in the design of classification and regression models within the framework of Logical Analysis of Data (LAD). LAD is a machine learning methodology proposed in the late 80's [41, 65], which utilizes ideas from Boolean functions, optimization and logic to construct classification models.

The original LAD algorithm deals with the machine learning task of distinguishing between observations belonging to one of two disjoint classes. A typical implementation of LAD encompasses a number of steps, including the enumeration of a collection of *patterns*, or *rules*, the selection of a relatively small set of such patterns, and the definition of a so-called *discriminant function* of theses patterns that is used for the classification of unseen observations.

In this thesis we propose alternative algorithms for each of the above mentioned steps, each of which utilizes an optimization model that accounts for the proper fitting of the given training data. The algorithm described in Chapter 3 considerably simplifies the task of constructing LAD models, since it requires the calibration of a single parameter, while achieving an accuracy as good as (if not better than) that of the standard LAD implementation. The linear optimization model of Chapter 5 allows the construction of LAD models of reasonable accuracy without the need of virtually any parameter calibration. In Chapter 6 we propose another LP formulation that permits us to extend the LAD methodology for dealing with regression problems, where each observation is assigned a numerical value, rather than being labeled as belonging to a class. In what follows, we describe in more details the contributions of each chapter.

In Chapter 2 we provide a brief review of Logical Analysis of Data, describing the main procedures involved in the construction of a LAD model for a given training dataset. We also give high-level descriptions of other machine learning algorithms used in this thesis for the purpose of assessing the relative quality of the algorithms proposed here. At the end of the chapter we discuss the cross-validation procedure utilized in the subsequent chapters.

Chapter 3 is based on a paper that has been accepted for publication in Discrete Applied Mathematics [20]. We describe a family of LAD patterns consisting of those patterns having maximum coverage in a given dataset. We present a formulation of the problem of finding a maximum coverage pattern covering a specific observation as a nonlinear set covering problem, show a natural reformulation of the problem as an integer linear program, and describe three heuristics for its solution. We also evaluate the exact and the heuristic algorithms in terms of their running times, quality of solutions, and in terms of accuracy of the LAD models built using the patterns obtained with these algorithms. We argue in favor of using two simple heuristics for building LAD models, which are computationally efficient and provide accuracies on par with those of standard machine learning algorithms.

In Chapter 4 we describe a branch-and-bound algorithm for a class of pseudo-Boolean optimization problems related to the maximum pattern problem of Chapter 3. The solution of this type of problem is utilized subsequently in this thesis as an essential part of the algorithms proposed in Chapters 5 and 6. The pseudo-Boolean optimization problems addressed in Chapter 4 have the property that all their terms have large degree (equal to $n/2$, where n is the number of binary variables in the problem). The large degree of the terms led us to propose a simple branching strategy that fixes the terms at values 0 or 1 — rather than fixing individual variables at these values — typically permitting a large number of binary variables to be simultaneous fixed, substantially simplifying the problem at every branch of the search tree. A simple node selection strategy is also described, including a variation for finding a solution of good quality early in the search. The results of a number of computational experiments are reported, comparing the running time and solution quality of our algorithm with those of the Xpress-MP integer linear programming solver [43]. We conclude that, in spite of its simplicity, our branch-and-bound algorithm is quite efficient and exhibits a clearly superior behavior to that of the Xpress solver.

In Chapter 5 we present a novel algorithm for the construction of a LAD model based on the optimization of its associated discriminant function. We first provide a variant of the linear program introduced in [28] for finding an optimal discriminant function given a set of patterns. This LP attempts to maximize the margin of separation between the two classes in the dataset, while accepting a small rate of misclassification, controlled by the use of a *soft margin* penalty factor. In the remainder of the chapter we describe the application of the column generation principle to the solution of this LP formulation on the space of *all* patterns. Starting from any nonempty set of patterns, we make use of the branch-and-bound algorithm of Chapter 4 in the subproblem phase for verifying optimality and for constructing new patterns as needed.

In addition to providing a clear optimization criterion that guides and unifies the usually distinct tasks of pattern generation and model selection, the algorithm is virtually parameter-free, having only a few control parameters that are primarily related to the underlying optimization process. In our computational experiments the accuracy of the proposed algorithm is 94.3% (on average) of the highest accuracy among the accuracies of six other (properly calibrated) machine learning algorithms.

In Chapter 6 we describe an extension of the LAD methodology for handling regression problems. Given a set of conjunctions, we propose a linear programming formulation for finding an optimal L_1-norm approximation of the dependent variable as a linear function in the so-called *conjunction-space*, i.e., the space where each observation is represented as the characteristic vector of the conjunctions satisfied by it (assuming an arbitrary order of the conjunctions). Starting from a simple set of conjunctions, we solve this LP by column generation in a similar way to the one described in Chapter 5. We compare the results of this algorithm with those of three other regression algorithms in terms of correlation and mean absolute error. We conclude that the performance of our algorithm is comparable to that of the best among the other algorithms utilized for comparison.

Finally, in Chapter 7 we highlight the main contributions of this thesis.

Chapter 2

Literature Review

2.1 Introduction

In this chapter we describe the methodology of Logical Analysis of Data (LAD) and discuss its main components. Additionally, we briefly outline the ideas behind several well-known classification and regression algorithms that are used as a reference in this thesis and present the validation procedures used to estimate accuracy in our computational experiments.

In Section 2.2 we present the main concepts of LAD, describe some variants of the main concepts and discuss several some implementation details. In Section 2.3 we briefly describe some classification and regression algorithms that are used as a reference in the remainder of the thesis. These algorithms serve mainly the purpose of carrying out empirical evaluations of the algorithms proposed in this thesis.

Finally, we outline in Section 2.4 the cross-validation procedure used throughout the text to evaluate the predictive accuracy of our algorithms.

2.2 Logical Analysis of Data

The methodology of Logical Analysis of Data (LAD) was first introduced in the 80's in [41] and [65] for dealing with classification problems involving binary data. In [27] the algorithm was extended to the case of non-binary data by means of a so-called *binarization* process, in which every numerical or categorical variable is discretized and replaced by one or more binary variables, bringing the problem to the usual binary form.

In [28] the implementation of a LAD classification system was described, and in [4], [8], [11], [20], [51], and [66] several further developments to the original algorithm were proposed. A recent overview of LAD can be found in [70].

Let us assume we are given a dataset Ω, consisting of a number of real vectors, each belonging to one of two disjoint *classes*. We will write $\Omega = \Omega^+ \cup \Omega^-$, with $\Omega^+ \cap \Omega^- = \emptyset$, and we will refer to the set Ω^+ as the set of "positive" examples or observations, and to the set Ω^- as the set of "negative" examples or observations.

LAD can be seen as a rule-based algorithm, with some similarity to algorithms such as C4.5Rules [104], and rough sets classification [103]. LAD builds a set of rules, or "patterns" which are representative of the observations in Ω and defines a so-called "discriminant function", based on the constructed rules. The discriminant function is used for the classification of unseen observations. We show in Chapter 6 that a similar function can also be used for the prediction of a numerical outcome in the context of regression problems.

In the following subsections we describe the main LAD concepts and how they are currently implemented.

2.2.1 Binarization

In this section we present the usual binarization process of LAD, and discuss a natural way to extend it to the case of regression problems.

In most real-life applications a substantial part of the data is numerical. We describe below a simple binarization procedure for converting a numerical variable (also called feature, or predictor) into one or more binary ones, while trying to preserve the information available in the original variable. The binarization algorithm described here is based on the one introduced in [27].

Let us denote by $x_1, \ldots, x_n$ the numerical variables (or attributes) of Ω. For each variable x_j, we construct a list L_j of the values that x_j takes in Ω, in ascending order. Let us consider two consecutive values $u < v$ in L_j such that x_j takes the value u in at least one positive observation, and the value v in at least one negative observation, or vice-versa. For each such pair of values, let us introduce a cutoff value of $\frac{u+v}{2}$, called a *cutpoint*, whenever $v - u > \tau$, where τ is a predefined tolerance value that determines whether two actual values of x_j are considered to be "too close". We say that $\frac{u+v}{2}$ *separates* the points with $x_j > \frac{u+v}{2}$ from those with $x_j < \frac{u+v}{2}$. If $v - u \leq \tau$ then the two values u and v are so close that the introduction of a cutpoint between them is not justified: if the measurement of the values of x_j is subject to some small imprecision, then, in practice, we are not able to distinguish between values u and v. In our computational experiments, the value of τ is defined indirectly, as a fixed percentage p of the standard deviation of variable x_j, the same value p being used for all variables $x_1, \ldots, x_n$. We shall refer to the parameter p as the *measurement imprecision* parameter.

Let us assume that a set of cutpoints has been generated for each variable using the above definition. We associate to each cutpoint c (corresponding to a variable x_j) an indicator variable y_j^c such that

$$y_j^c = \begin{cases} 1, & \text{if } x_j > c \\ 0, & \text{otherwise.} \end{cases}$$

With this notation we now have a binary description of each point of Ω in terms of indicator variables. The rationale behind the binarization process is that points in Ω that are significantly different with respect to the original numerical variables should have significantly different representations in terms of indicator variables as well.

In general, the set of cutpoints generated in the way described above can be very large. However, it is frequently observed that only a small subset of the cutpoints suffices to map Ω to a binary representation where the differences between significantly distinct points are preserved to a reasonable extent. It will become clear in subsequent chapters that the overall computational performance of LAD algorithms is directly linked to the number of indicator variables actually used. In view of this, we show how to formulate a set covering problem whose solution provides a minimum-size set of cutpoints that still preserves the original information to a predefined level.

Let us consider the entire set $C = \{c_1, \ldots, c_{|C|}\}$ of cutpoints generated with a suitable choice of the measurement imprecision parameter p, and let $I = \{1, \ldots, |C|\}$ be the index set of those cutpoints. Let z_j be a binary decision variable associated to the inclusion or not of the j-th cutpoint in the definitive set of cutpoints to be used. For each pair of observations in Ω consisting of a positive and a negative observation, we would like to include at least one of the cutpoints (if there is any) that separates the two observations into the definitive set of cutpoints. At the same time, for the sake of improved computational performance we want the set of cutpoints chosen to be as small as possible. This gives rise to the following minimum set cover problem:

$$\begin{aligned}
\text{(SC)}\quad \text{minimize} \quad & \sum_{k=1}^{|C|} z_k \\
\text{subject to:} \quad & \\
& \sum_{k \in D(\omega^i,\omega^j)} z_k \geq 1, \;\; \forall\, \omega^i \in \Omega^+, \omega^j \in \Omega^- \qquad (2.1) \\
& z_k \in \{0,1\}, \;\; j = 1, \ldots, |C|,
\end{aligned}$$

where $D(\omega^i, \omega^j) \subseteq I$ is the index set corresponding to the cutpoints that separate observations ω^i and ω^j. To ensure a more robust separation of the observations in Ω we may require that each pair of positive and negative observations be separated by more than one cutpoint. For that purpose we replace the right hand side of constraints (2.1) by an integer larger than 1.

Several real-life datasets contain variables that are neither binary nor numerical, but instead assume one of a set of nominal, or categorical, values. Let $S = \{s_1, \ldots, s_{|S|}\}$ be the set of possible values that a certain nominal variable x_j can assume. In some cases, there is a total order between the possible values, for instance: *low*, *medium*, and *high*. In such a case, the binarization of x_j can be done by associating a sequence of $\lfloor log(|S|-1) \rfloor + 1$ binary variables to it in such a way that each value s_i of x_j is represented in binarized form simply as the binary representation of the integer $i-1$. Alternatively, or when no total order among the elements of S is available, the binarization of x_j can be accomplished by associating to it a set of $|S|$ binary variables, each one of which represents a possible value of x_j. In the resulting binarized representation of x_j exactly one of the associated indicator variables would have the value 1.

In the case of regression problems, we need to modify the procedure described above, since there is no definition of positive and negative observations. For the purpose of generating cutpoints for a variable x_j we consider all those consecutive values $u < v$ in L_j such that $v - u > \tau$. Once a set of cutpoints is generated in this manner, we solve a modified version of problem (SC) in which the constraints (2.1) are replaced by

$$\sum_{k \in D(\omega^i, \omega^j)} z_k \geq \left\lceil \frac{|r(\omega^i) - r(\omega^j)|}{\rho} \right\rceil, \ \forall\, \omega^i, \omega^j \in \Omega, \tag{2.2}$$

where $r(\omega^i)$ and $r(\omega^i)$ are the numerical outcomes associated to observations ω^i and ω^i, respectively, and ρ is a parameter determining the difference in outcome that justifies the requirement of one more cutpoint to separate a pair of observations.

It is clear that for very large datasets solving (SC) may become prohibitively expensive. In such cases one can apply a heuristic, such as the greedy one in [37], to find a relatively small set of cutpoints that is reasonably informative. Another alternative is to select a large value of the imprecision parameter p, resulting in the generation of a smaller set of cutpoints that detect only the most pronounced discrepancies in the values of the original variables. In the experiments presented in this study, we solve (SC) by the greedy heuristic [37], due to satisfactory performance of this procedure and to the fact that solving (SC) to optimality is not practically justified in data analysis problems, as noted by Dietterich in [44].

2.2.2 Patterns

In this subsection we assume that the dataset Ω is given in binary form.

In the case of a binary dataset a *pattern* is simply a homogeneous subcube of $\{0,1\}^n$, i.e., a subcube having (i) a nonempty intersection with one of the sets Ω^+ or Ω^-, and (ii) an empty intersection with the other set (Ω^- or Ω^+, respectively). We recall that a subcube consists of those points of the n-cube for which a subset of the variables is fixed to 0 or 1, while the remaining variables take all the possible 0,1 values.

A pattern P disjoint from Ω^- is called a *positive* pattern, while a pattern P' disjoint from Ω^+ is called a *negative* pattern. A point in $\{0,1\}^n$ is said to be covered by a pattern if it belongs to the subcube defined by that pattern.

Note that for a dataset with numerical variables that has been binarized according to the procedure described in the previous subsection, the definition of a pattern can be seen as a set of constraints that are simultaneously imposed on the values of one or more of the original variables. Indeed, let the j-th binary (indicator) variable be associated to the k-th original (numerical) variable and with the cutpoint value c. Then, a pattern requiring that the j-th binary variable equals 1 corresponds to the requirement that the k-th original variable exceeds c.

The *prevalence* of a positive pattern with respect to a given set of observation is the percentage of positive observations that are covered by the pattern, while the *coverage* of a pattern is simply the number of observations covered by the pattern in the given set. The *homogeneity* of a positive pattern is the percentage of positive observations among the set of observations covered by it. These concepts can be defined for negative patterns in a similar way.

Previous implementations of LAD (see [4], [11], [27] and [28]) have all relied on some type of pattern generation procedure that involves the enumeration of large collections of positive and negative patterns, each satisfying certain requirements in terms of prevalence and homogeneity.

2.2.3 Non-homogeneous Patterns and Conjunctions

The concept of patterns discussed above is based on the ideal assumption that the historical information is perfectly correct. More specifically, it is assumed that, on the one hand, all the attribute values are measured precisely, and on the other hand, all the classifications of observations are recorded correctly. In most real-life situations these ideal assumptions do not hold. This fact can be seen in the evaluation of patterns on a testing set, showing that positive patterns of large coverage (on the training set) cover on the testing set not only a significant number of positive observations but also a (usually small) number of negative observations.

In view of this fact, it is reasonable to allow patterns to cover a “small” number of observations of the opposite class. It is to be expected that such a relaxation of constraints defining positive (negative) patterns should lead to a significant increase in the coverage of positive (negative) observations. We shall refer to those patterns that do not cover any observations of the opposite class as “pure” or “homogeneous”, while the other ones shall sometimes be called “non-homogeneous”. In most cases, we will simply refer to a “pattern”, without making any distinction between homogeneous and non-homogeneous patterns.

Therefore, we can say that a pattern is a subcube of $\{0,1\}^n$ with the property that it covers "mostly" observations from one of the sets Ω^+ or Ω^-, covering only a small number of observations from the other set. A pattern is positive if the percentage of observations from Ω^+ covered by the pattern is larger than the percentage of observations from Ω^- covered by it. A negative pattern is defined in a symmetric way.

More generally, we can construct subcubes that do not necessarily satisfy the conditions for being considered patterns. Let us consider each subcube as described by its associated Boolean conjunction, where a literal is used whenever the value of the corresponding binary variable is fixed in the subcube. For instance, let $n = 5$ and $C = x_1\overline{x_3}$. The conjunction C is associated to the subcube of $\{0,1\}^5$ containing all points in which the first component takes the value 1 and the third component takes the value 0.

For the task of classification we are only interested in those conjunctions having certain values of prevalence and homogeneity. Indeed, in order to extract information that can be used to classify "unseen" observations, we are only interested in finding a conjunction with the property that the set of observations that it covers has a distribution of points from Ω^+ and Ω^- that is significantly different from that originally in Ω.

Let us consider the case of regression problems, where every observation is associated to a numerical value, rather than a class label. Thus, there is no distinction between positive and negative observations. In this setting, the concept of pattern can be replaced by that of a conjunction. Every conjunction simply identifies a subgroup of the observations. We shall see in Chapter 6 that the use of conjunctions as predictors, or independent variables, can form the basis of accurate regression models.

2.2.4 LAD Models

The basic assumption of *LAD* is that a binary point covered by some positive patterns, but not covered by any negative pattern is positive, and similarly, a binary point covered by some negative patterns, but not covered by any positive pattern is negative.

Let M^+ be a set of positive patterns and M^- be a set of negative patterns. If every observation from Ω^+ is covered by at least one pattern from M^+ and every observation from Ω^- is covered by at least one pattern from M^-, the we say that $M^+ \cup M^-$ is a *LAD model* for Ω. In many cases, when constructing a model we require that every observation in the training dataset be covered at least a number k of times by the patterns in the model, for an integer $k > 1$. This additional requirement usually has the consequence of increasing the number of patterns in the model, sacrificing its simplicity. However, it may result in more robust models for classification of unseen observations, as suggested in [8] and in other experiments with combinations of classifiers [13, 47, 82].

The construction of a LAD model for a given dataset typically involves the generation of a large set of patterns and the selection of a subset of them that satisfies the definition of a LAD model presented above, and such that each pattern in the model satisfies certain requirements in terms of prevalence and homogeneity.

2.2.5 Classification

Given a LAD model $M = M^+ \cup M^-$ and a "new" observation $\omega \notin \Omega$, the "classification" of ω is determined by the sign of a so-called *discriminant* function $\Delta : \{0,1\}^n \rightarrow \mathbb{R}$ associated to the given model. Let $M^+ = \{P_1, \ldots, P_{M^+}\}$ and $M^- = \{N_1, \ldots, N_{M^-}\}$, and let us associate to each positive pattern $P_i \in M^+$ a real weight α_i and to each negative pattern $N_j \in M^-$ a real weight β_j. The associated discriminant function on ω is given by

$$\Delta(\omega) = \sum_{P_i \in M^+} \alpha_i P_i(\omega) - \sum_{N_i \in M^-} \beta_i N_i(\omega),$$

where $P_i(\omega)$ equals 1 if ω is covered by P_i, and 0 otherwise (similarly for $N_i(\omega)$). The sign of $\Delta(\omega)$ determines the classification of ω. If $\Delta(\omega) = 0$ the observation is either left unclassified, or is classified according to the more frequent class.

In most applications of LAD, equal weights are used among the positive patterns of a LAD model, and also among its negative patterns. A more sophisticated way of selecting weights is used in [28] and consists of the solution of a linear programming model that chooses the weights in such a way so as to maximize the separation of the training set in the so-called *pattern-space.* The pattern-space representation (associated to an ordered set of patterns) of an observation ω is simply the characteristic vector of the patterns covering ω.

Note that in this approach the discriminant function is optimized over a given set of patterns that was previously selected during the pattern generation procedure. Since the number of patterns can be extremely large, the use of this approach assumes that a relatively small set of patterns has been previously identified.

2.3 Classification and Regression Literature

In this section we briefly describe some of the main classification and regression algorithms used in the machine learning literature. We concentrate our discussion on those algorithms that we use in subsequent chapters for the purpose of evaluating the quality of our computational results. Whenever possible we highlight some connections between particular features of these algorithms and some of the concepts present in LAD.

Most algorithms have a set of parameters that is dependent on the specific implementation. We discuss the main parameters of each algorithm based on their implementation in the Weka package [128].

2.3.1 Decision Trees (C4.5)

The decision tree method for classification consists of a hierarchical partition of the input space into homogeneous or almost homogeneous subspaces. For the case of numerical features, the partition is usually rectangular, i.e., parallel to the original axes. Some variants have been proposed in which more general partitions are applied, usually through the construction of artificial features based on simple operations involving the original ones. A relatively comprehensive overview of decision tree models can be found in [104].

In a decision tree model each node represents a *test* involving the values of one or more features, with each branch corresponding to one of the possible outcomes of the test. For instance, $x_1 \geq 5$ is a test that splits the input space into two groups: those observations where attribute x_1 takes values below 5, and those where x_1 takes values above or equal to 5. Since this test has only two possible outcomes, it is said to be a *binary test*. Tests with a larger number of outcomes are possible, in particular when the feature used in the test assumes nominal or discrete values. A test can also involves more than one attribute. For instance, one could use a test of the form $\alpha x_1 + \beta x_2 \leq 2$.

Given a training dataset, each test *separates* the observations into one or more smaller sets. By the successive application of such tests one can grow a tree-like structure that partitions the data into increasingly smaller regions of the input space. The construction of a decision tree for a specific training dataset is traditionally done in a greedy way. Given the entire training data, a test which *best* separates the training data is found and implemented, so that the training dataset is split into one or more subsets. Each subset is subsequently split according to the same criterion of goodness of separation, until each subset of observations is sufficiently homogeneous with respect to the classes of the observations. Therefore, the leaves of a decision tree are associated to specific regions of the input data where the class distribution of observations is significantly more homogeneous than the original distribution on the training dataset.

The criterion used for measuring the goodness of a test is based on a simple measure, such as entropy. The main parameters related to the construction of a decision tree concern the types of tests utilized, the criterion for stopping the growth of a tree, and the post-construction pruning of the decision tree model in order to avoid overfitting the training data.

Clearly, the partition of the input space into sub-regions in the decision tree context is closely related to that of patterns in Logical Analysis of Data. Indeed, the sequence of tests down a path in a decision tree corresponds to a pattern in LAD, given that it defines a subset of the training observations that satisfies certain conditions of prevalence and homogeneity. The main difference that should be noted here is that in the case of decision trees the set of "patterns" constructed is typically significantly smaller than that of LAD, since a test is only selected if it improves "locally" the separation of a set of observations that has been defined by a number of previously implemented tests. In the usual LAD enumeration scheme, all patterns (involving up to a certain number of tests) are generated and evaluated. Clearly, the greedy criterion of decision trees can neglect patterns that define highly homogeneous regions of the input space, but whose individual tests do not significantly differentiate between the classes present in the training dataset. From this point of view, the LAD classifiers are more general than decision tree models. On the other hand, because decision tree models are usually simpler classifiers, they are computationally inexpensive to generate and, if pruned, are less prone to overfitting the data.

2.3.2 Random Forests

A generalization of the decision tree model, called *random forests*, was recently proposed by Breiman [33]. In the random forests model several decision trees are constructed using small perturbations of the original training data, and the set of trees constructed is used for classification. The justification for such a procedure is to compensate for the effect of the local decisions regarding the choice of the next test to be implemented when growing a single decision tree.

Each of the trees in a random forests classifier is built using a small subset of the features in the training dataset. Typically a randomly selected set of 2, 3, or $\log_2(n)$ features (where n is the total number of features in the training dataset) is used for growing an entire tree, or for selecting the best test for each node.

The computational cost of building trees in this way is obviously very small. Also, the generalization ability of each individual tree built in this way is expected to be quite limited. However, as suggested by other studies involving the combination of simple classifiers, such as boosting [56, 113, 114], bagging [32, 105], and others [13, 47, 82], the collective use of such simple trees has been shown to result in very accurate classification models [13, 46, 105].

The number of trees used in a random forests model depends on the training dataset used. Typical values range from tens to several hundreds. The random factor in the random forests algorithm seems to effectively compensate for the limitation of growing decision trees in a greedy fashion. Given that a sufficiently large number of trees is generated, one can expect that random forests models have a practical performance very close to that of the usual LAD models based on the enumeration of complete families of patterns [4, 8, 11].

2.3.3 Support Vector Machines

The support vector machine algorithm can be interpreted as the construction of a linear classifier in a very high-dimensional space (called the *feature space*), obtained by transformation of the original input space.

The key ingredient of the algorithm is a *kernel function* that allows the training phase and the classification of new observations to be carried out in the feature space without the need to actually perform the transforming computations.

The typical support vector classifier (for two-class problems) consists of a linear discriminant function that *separates* the training data in a similar way to the LAD discriminant function defined in Section 2.2. A quadratic optimization model is used to optimize the weights, so that the *margin of separation* between the two classes if maximized. The margin of separation is simply the smallest distance from a point in one of the classes to the separating hyperplane, plus the smallest distance from a point in the other class to the separating hyperplane.

The formulation of the underlying optimization model is such that the only information required about the feature space utilized is the inner product between every pair of (transformed) observations in the training dataset. The kernel function is chosen in such a way that it provides, with low computational costs, the inner product between two observations mapped into the feature space. Clearly, one is interested in choosing a feature space in which a better separation of the two classes is possible than that obtained in the input space.

In practice, the optimization model takes into account a penalty term, in order to allow some observations in the training dataset to be incorrectly classified. The so-called C-parameter dictates how much importance the model should give to the perfect separation of the training data, as opposed to the maximization of the margin of separation of "most" observations. The value of C is a critical parameter in tuning the support vector machines algorithm.

Another important parameter of the algorithm is the kernel function used, or – in other words – the feature space chosen. Many different kernel functions have been proposed for specific types of data. Among the general-purpose kernel functions frequently used we cite the *polynomial* and *radial basis function* kernels.

A very similar variant of the optimization model utilized for training allows the use of the same algorithm for regression tasks, resulting in the so-called support vector regression algorithm. For a comprehensive treatment of support vector machines for classification and regression, the reader is referred to [115].

Here, an analogy can be made with the discriminant function of LAD. Consider a LAD model for a specific dataset. If we assume a certain order among the patterns in the LAD model, and map every observation to the characteristic vector of the patterns covering it, we have effectively mapped the dataset into a certain type of feature space. Moreover, the optimization model described in Section 2.2 for optimizing the weights of the discriminant function attempts to maximize a certain type of margin of separation that resembles the one utilized in the support vector machines algorithm. In Chapter 5 we further explore this optimization model and propose an iterative algorithm that explicitly constructs the best possible such *pattern-based feature space.*

2.3.4 Logistic Regression

The logistic regression [39, 76, 122] algorithm is a regression technique for the case when the dependent variable is binary, i.e., when the problem is in fact a two-class classification problem.

The algorithm fits a linear model of the independent variables to the log odds of the dependent variable being 1. Therefore, although the model is linear in its parameters, it predicts a nonlinear function of the dependent variable. The resulting model provides information about the relative importance of the independent variables: the coefficients of the input variables represent the effect on the log odds caused by a unit change in the value of the variable. The value resulting from applying the model to an unseen observation is used to provide a prediction of its class according to the estimate obtained for its log odds: an observation with the log odds greater than or equal to 1 being classified as positive, and otherwise being classified as negative.

The training of a logistic regression model, i.e., the adjustment of an independent term and the coefficients of the input variables, is performed by maximum likelihood estimation (MLE). The MLE algorithm attempts to maximize the likelihood that the “dependent variable” (in the case of logistic regression, the log odds) values are obtained from the values of the independent variables.

2.3.5 Neural Networks (Multilayer Perceptron)

A neural network [90, 91, 106, 107, 110] consists of a set of nodes and a set of arcs, or connections, each of which has an associated weight. Each node is a simple processing unit that receives one or more inputs, and uses their sum as its output value.

A network is typically represented as a graph organized in “layers”. The first layer consists of as many nodes as the number of input variables in the data, with each node in that layer receiving a single input corresponding to the associated input variable. The last layer consists of as many nodes as the number of classes in the data, each node having no connections to other nodes. For two-class classification problems, the final layer typically contains a single node, determining a binary output. The intermediate layers, if any, are present to allow an extended flexibility of the network, so that it can learn complex functions.

During the training phase, the input values of a training observation are initially fed into the first layer of the network. Depending on the input received by a node, it may be “activated” and propagate the signal received to the set of nodes to which it connects. The propagation is determined by a so-called *activation function*, which is applied to the output of each node in order to decide if the set of inputs of that node was enough to justify the propagation of the signal. If so, the node propagates the value 1 to each of the nodes to which it connects; otherwise, no signal is propagated by that node.

After a full series of propagations reach the final layer, the only node in that layer is either activated by its inputs, or not, thus determining the classification of the observation given as input. If the classification is incorrect, a correction in the weights of the connections must be made. Typically a stochastic gradient descent algorithm is utilized to update the weights in such a way as to globally minimize the number of errors made on the set of training observations.

Slight variations in the cost function allow the adjustment of the algorithm to a classification or regression setting.

2.4 Estimating Accuracy

When analyzing the experimental results of the algorithms proposed in this thesis we utilize the classical cross-validation procedure [45, 52, 73, 80] in order to estimate the accuracy of the algorithm on unseen observations.

The k-fold cross-validation procedure first partitions the training data into k approximately equal-sized parts. (In classification, the class distribution in each part is approximately the same as that in the entire training data.) The first part is left aside and the remaining $k-1$ parts are used for training. The resulting model is evaluated on the first part and the accuracy recorded. The same process is repeated k times, every time with a different part being left out of the training process and being used for testing purposes. At the end of the procedure, k accuracy measurements have been taken and the average of these is reported as an estimate of the accuracy of a model constructed on the entire training set when presented with unseen data.

The accuracy measure utilized can be the usual percentage of incorrectly classified examples in the classification setting, or the correlation or mean absolute error for the regression case.

Chapter 3

Maximum Patterns

3.1 Introduction

Patterns are the key building blocks in Logical Analysis of Data (LAD) (e.g., see [28], [7], [4], [11] and [66]), and have been shown in numerous studies to provide important indications about the positive or negative nature of the points covered by them. As discussed in Chapter 2, the collection of patterns used in the implementation of LAD (see e.g. [28]) is generated by a combinatorial enumeration process, which can be quite expensive computationally. The number and type of patterns generated in this process are controlled by several controls parameters. The choice of the most appropriate parameter values is quite involved and time-consuming, being based on numerous computational experiments. Moreover, even with the best choice of control parameter values the size of the pattern collection produced is very large and requires in most cases the application of a "filtering" procedure, which selects small subsets of patterns to form highly accurate predictive models.

In this chapter we shall address the complexities of the pattern generation process by introducing the concept of "maximum" patterns. An important property of the approach used in this chapter is that the number of maximum patterns produced is naturally bounded by the number of observations. We propose a discrete optimization based exact algorithm, and highly efficient heuristics, for generating maximum patterns, and show that the accuracy of LAD models based on these patterns is highly competitive with that of the original LAD models, as well as with those of the best commonly used classification methods.

The vast literature on data analysis contains several approaches which resemble in certain respects the general classification methodology of Logical Analysis of Data

(LAD) proposed in 1986 and 1988 in [65, 41]. In Computational Learning Theory there is a stream of research devoted to DNF learning [35, 79] which captures certain aspects of LAD. Among related empirical machine learning approaches we have to mention those based on production or implication rules, especially those derived from decision trees, such as C4.5rules [104], those based on Rough Set theory, such as the Rough Set Exploration System [103, 17], as well as techniques based on ensemble of classifiers such as boosting [56, 114] and bagging [32, 105].

The implementation of the LAD methodology on the basis of the concept of maximum patterns proposed in this chapter bears resemblance to the concept of emerging patterns, proposed in 1999 in [49]. The emerging pattern problem of [49] and [126] is a special case of the maximum pattern problem considered in this chapter in which the only admissible patterns are monotonically non-decreasing.

Another approach we would like to mention is that of "subgroup discovery techniques", see [86]. The subgroup discovery algorithm described in [86] differs from the methods proposed in this chapter in the chosen measure of pattern quality. The algorithms in this chapter maximize the coverage of patterns while limiting their coverage of the opposite class. In contrast, the subgroup discovery algorithm of [86] maximizes a measure of the coverage of patterns, which is discounted by their coverage of the opposite class.

This chapter is organized as follows. In Section 2 we introduce the basic notation used throughout the text. In Section 3 we develop an integer programming formulation for the construction of exact maximum patterns, and in Section 4 we propose three heuristics for constructing approximately maximum patterns. Sections 5 and 6 describe natural extensions of the concept of patterns to datasets with missing attribute values and possible misclassification noise, showing how to modify the pattern generation algorithms to handle these cases. Section 7 presents computational results concerning the generation of exact and heuristic patterns. Section 8 evaluates the accuracy of LAD models built using one or more of the algorithms described in Sections 3 and 4, and shows how their accuracies compare to those of some commonly used classification algorithms. In Section 9 we discuss the comparative accuracy of the LAD models using maximum patterns, and argue in favor of the use of a combination of two of the proposed heuristic algorithms as an efficient pattern generation procedure for constructing accurate LAD models.

3.2 Notation

Following the notation defined in Chapter 2, let us consider a binary dataset $\Omega = \Omega^+ \cup \Omega^- \subset \{0,1\}^n$, with $\Omega^+ \cap \Omega^- = \emptyset$. For the purpose of developing the exact model and heuristic algorithms of Sections 3.3 and 3.4 we assume that $\Omega^+ \cap \Omega^- = \emptyset$ and consider only homogeneous (i.e., pure) patterns. In Section 3.5, we relax this requirement by introducing the concept of "fuzzy patterns".

Let us recall that for $\alpha \in \Omega^+ \subset \{0,1\}^n$ a positive α-pattern is a pattern covering α. A *maximum positive α-pattern P* is a positive α-pattern of maximum coverage in Ω^+. A maximum negative α-pattern is defined in a similar way.

Previous experience with *LAD* ([28], [66]) has shown that patterns with higher coverage provide better indication of the positive or negative character of new observations than those with lower coverage. This observation motivates the focus of this chapter on maximum patterns, their construction, and their use in classification.

3.3 Construction of Exact Maximum Patterns (EMPs)

Given the sets Ω^+, Ω^-, we shall be concerned here with ways of finding a maximum positive α-pattern, $\alpha \in \{0,1\}^n \setminus \Omega^-$. The determination of a maximum negative α-pattern can be done in a symmetric way. In view of the perfect symmetry of positive and negative α-patterns we shall describe the proposed methodology only for the positive case.

In order to formulate the maximum α-pattern problem as an integer program, we shall introduce a binary decision variable y_j which describes whether or not the value of the j-th variable of the pattern is fixed to α_j. With this notation, the condition that the α-pattern should not include any negative point requires that for every point γ of Ω^-, the variable y_j should take the value 1 for at least one of those j's for which $\gamma_j \neq \alpha_j$, i.e.,

$$\sum_{\substack{j=1 \\ \gamma_j \neq \alpha_j}}^{n} y_j \geq 1, \quad \text{for every } \gamma \in \Omega^-. \tag{3.1}$$

On the other hand, a positive point β will be covered by the α-pattern if and only if $y_j = 0$, for all those indices j for which $\beta_j \neq \alpha_j$. Therefore, the number of positive points covered by the α-pattern will be given by

$$\sum_{\beta \in \Omega^+} \prod_{\substack{j=1 \\ \beta_j \neq \alpha_j}}^{n} \overline{y}_j, \tag{3.2}$$

where $\overline{y}_j = 1 - y_j$, for every $j = 1, \ldots, n$. In conclusion, the maximum α-pattern problem can be formulated as the following nonlinear integer program

$$\begin{aligned} \textit{maximize} \quad & \sum_{\beta \in \Omega^+} \prod_{\substack{j=1 \\ \beta_j \neq \alpha_j}}^{n} \overline{y}_j \\ \textit{subject to} \quad & \sum_{\substack{j=1 \\ \gamma_j \neq \alpha_j}}^{n} y_j \geq 1, \quad \text{for every } \gamma \in \Omega^- \\ & y_j \in \{0,1\}, \quad \text{for every } j = 1, \ldots, n. \end{aligned} \tag{3.3}$$

This problem is a generalized set covering problem. Indeed, in the special case when Ω^+ consists of the n points $\beta^{(i)}$, where $\beta^{(i)}$ differs from α only in variable i, then the objective function becomes simply $n - \sum_{j=1}^{n} y_j$, which is equivalent to minimizing $\sum_{j=1}^{n} y_j$, i.e., to a standard set covering problem. In view of this remark it is clear that this problem is $\mathcal{NP}$-hard and hence no polynomial algorithm is available for its solution. Moreover, it has been shown [54] that the set covering problem is not approximable within $c \log n$, for any real c such that $0 < c < 1$.

Since numerous software packages are available for solving integer linear programs, it is useful to rewrite (3.3) in this form. This can be achieved by introducing a new binary variable z_β to replace each term of the objective function of (3.3). Clearly, in the case of 0/1 variables, the relation

$$z_\beta = \prod_{\substack{j=1 \\ \beta_j \neq \alpha_j}}^{n} \overline{y}_j$$

is equivalent with the system of inequalities (3.4) and (3.5):

$$\begin{aligned} w(\beta)z_\beta &\leq \sum_{\substack{j=1\\ \beta_j\neq\alpha_j}}^{n} \overline{y}_j, \text{ for every } \beta\in\Omega^+, &(3.4)\\ \sum_{\substack{j=1\\ \beta_j\neq\alpha_j}}^{n} \overline{y}_j &\leq z_\beta + w(\beta) - 1, \text{ for every } \beta \in\Omega^+, &(3.5)\end{aligned}$$

where

$$w(\beta) = |\{j : \beta_j \neq \alpha_j\}|. \tag{3.6}$$

Note that on the one hand, (3.4) forces $z_\beta = 0$ whenever at least one $y_j = 1$, and on the other hand, (3.5) forces $z_\beta = 1$ whenever all $y_j = 0$.

Therefore, taking into account that $\sum_{\substack{j=1\\ \beta_j\neq\alpha_j}}^{n} \overline{y}_j = w(\beta) - \sum_{\substack{j=1\\ \beta_j\neq\alpha_j}}^{n} y_j$, the nonlinear integer program (3.3) can be seen to be equivalent to the linear integer program (EMP):

$$\begin{aligned} &\textit{maximize} && \textstyle\sum_{\beta\in\Omega^+\setminus\{\alpha\}} z_\beta \\ &\textit{subject to} \\ &&& \textstyle\sum^{n}_{\substack{j=1\\ \gamma_j\neq\alpha_j}} y_j \geq 1, \text{ for every } \gamma\in\Omega^- &(3.7)\\ &&& w(\beta)z_\beta + \textstyle\sum^{n}_{\substack{j=1\\ \beta_j\neq\alpha_j}} y_j \leq w(\beta), \text{ for every } \beta \in \Omega^+\setminus\{\alpha\}, \\ &&& z_\beta + \textstyle\sum^{n}_{\substack{j=1\\ \beta_j\neq\alpha_j}} y_j \geq 1, \text{ for every } \beta\in\Omega^+\setminus\{\alpha\}, \\ &&& y_j \in\{0,1\}, \text{ for every } j = 1,\dots,n, \\ &&& z_\beta\in\{0,1\}, \text{ for every } \beta\in\Omega^+\setminus\{\alpha\}. \end{aligned}$$

While the integer linear program (3.7) has the same feasible solutions as (3.3), the omission of the constraints (3.5) from (3.7) results in a new integer linear program, which may have a larger set of feasible solutions, but has exactly the same set of optimal solutions as (3.7), and therefore as (3.3). However, from the computational efficiency point of view this simplification is not beneficial and was not used in our experiments.

3.4 Heuristics

The number of binary variables appearing in problem (3.7) is $n + |\Omega^+| - 1$, which in case of large datasets results in very large integer linear programs. In view of the computational difficulty of handling such large integer linear programs, it is important to develop appropriate heuristics to deal with instances for which current integer linear programming software packages fail to find exact solutions to problem (3.7). In order to achieve this objective, two heuristic approaches will be presented below.

Section 3.4.1 will describe an approach based on replacing the original objective function of (3.3) by its best linear approximation in L_2, while Sections 3.4.2 and 3.4.3 will present two implementations of a greedy combinatorial heuristic in which the coverage of a positive α-pattern is successively increased in order to find a maximal (rather than maximum) positive α-pattern. Computational experiments with the proposed heuristics will be presented in Section 3.7.

3.4.1 Best Linear Approximation (BLA)

We shall describe in this section a model based on the nonlinear set covering model (3.3) in which we shall replace the objective function by its best linear approximation in L_2 in order to reduce it to the usual format of a weighted (linear) set covering problem. The determination of the L_2-best linear approximation of the objective function of (3.3) is based on the direct application of the results of [68].

The objective function of (3.3) is a real-valued function in binary variables, i.e., a *pseudo-Boolean function*. If we represent a pseudo-Boolean function $f(u_1, u_2, \ldots, u_m)$ as

$$f(u_1, u_2, \ldots, u_m) = \sum_{j=1}^{s} \left(c_j \prod_{i \in S_j} u_i \right),$$

where c_j are real numbers, and S_j are subsets of $\{1, 2, \ldots, m\}$, then the L_2-best linear approximation $L(f)$ of f is known ([68]) to be given by

$$L(f(u_1, u_2, \ldots, u_m)) = \sum_{j=1}^{s} c_j \; L(\prod_{i \in S_j} u_i).$$

Further, it was shown in [68] that

$$L(\prod_{i\in S_j} u_i) = -\frac{|S_j|-1}{2^{|S_j|}} + \frac{1}{2^{|S_j|-1}}\left(\sum_{i\in S_j} u_i\right).$$

Therefore, the L_2-best linear approximation of f is given by

$$L(f(u_1,u_2,\ldots,u_m)) = \sum_{j=1}^{s} c_j\left(-\frac{|S_j|-1}{2^{|S_j|}} + \frac{1}{2^{|S_j|-1}}\left(\sum_{i\in S_j} u_i\right)\right).$$

The computation of this formula should utilize standard numerical techniques to avoid the loss of precision.

Applying the above formula to the objective function of (3.3) we find that its L_2-best linear approximation is given by

$$\sum_{\beta\in\Omega^+}\frac{w(\beta)+1}{2^{w(\beta)}} - \sum_{j=1}^{n}\left(\sum_{\substack{\beta\in\Omega^+\\ \beta_j\neq\alpha_j}}\frac{1}{2^{w(\beta)-1}}\right)y_j.$$

Since the coefficient of every variable y_j is non-positive, we shall approximate problem (3.3) by the following weighted (linear) set covering problem:

$$\begin{aligned}
&\textit{minimize} \\
&\qquad \sum_{j=1}^{n}\left(\sum_{\substack{\beta\in\Omega^+\\ \beta_j\neq\alpha_j}}\frac{1}{2^{w(\beta)-1}}\right)y_j \\
&\textit{subject to} \\
&\qquad \sum_{\substack{j=1\\ \gamma_j\neq\alpha_j}}^{n} y_j \geq 1, \quad \text{for every } \gamma\in\Omega^- \\
&\qquad y_j\in\{0,1\}, \quad \text{for every } j=1,\ldots,n.
\end{aligned} \tag{3.8}$$

Clearly, the optimal solution of (3.8) will define a positive α-pattern whose size will provide a lower bound to the size of a maximum positive α-pattern. The computational experiments to be presented in Section 3.7 will show that this bound provides a good approximation of the maximum size of a positive α-pattern.

We recall that problem (3.8) is NP-complete and, therefore, in principle, its solution can become computationally intractable. However, many commercially available integer linear programming solvers are capable of solving fairly large size weighted set covering problems in an acceptable amount of time. Since in many machine learning datasets the number of relevant variables is relatively small [19], one can utilize a feature selection procedure (e.g. the binarization step of LAD [28]) to select such a small subset of relevant binary variables and then use this subset to construct an instance of problem (3.8) which can be solved fairly quickly.

3.4.2 Enlarging Patterns to Maximized Prime Patterns (MPP)

We will present in this and the next subsection combinatorial heuristics for solving (3.3). Since (3.3) is a generalized set covering problem, the combinatorial heuristics presented here are built on an idea which is similar to one used in the standard greedy heuristic for the common weighted set covering problem [37]. The important distinction of the heuristics proposed here is that they work "backwards", i.e., they start with a "cover" (minterm) and then proceed in a greedy fashion to improve the objective function while still satisfying all the covering constraints.

Given a point α, the associated *minterm* $\mathcal{P}_\alpha$ is defined as $\bigwedge_{i=1}^{n} x_i^{\alpha_i}$, where

$$x_i^{\alpha_i} = \begin{cases} x_i, & \text{if } \alpha_i = 1; \\ \overline{x}_i, & \text{if } \alpha_i = 0. \end{cases}$$

Clearly, if $\alpha \in \Omega^+$ then P_α is a positive α-pattern, covering only the point α. Each pattern can be represented as a Boolean conjunction $\bigwedge_{j\in S} x_j^{\beta_j}$, $S \subseteq \{1, \ldots, n\}$, of a subset of literals (i.e., complemented and non-complemented variables). Minterms are those positive patterns for which $S = \{1, \ldots, n\}$.

A positive pattern $\bigwedge_{j\in S} x_j^{\beta_j}$ is called *prime* if the conjunction $\bigwedge_{j\in S'} x_j^{\beta_j}$ is not a positive pattern for any proper subset $S' \subset S$. It is known from experience that many of the prime patterns have very large coverages. Because of this it makes sense to transform a minterm $\mathcal{P}_\alpha$ into a positive prime α-pattern.

If a positive pattern $\mathcal{P}$ is not a prime pattern then we can apply to it the iterative algorithm described in [66], which transforms a given pattern into a prime pattern. Starting from $\mathcal{P}$ we shall obtain a prime pattern $P = \bigwedge_{j\in S} x_j^{\alpha_j}$, for some $S \subset \{1, \ldots, n\}$, by successively removing literals from $\mathcal{P}$ so as to maximize the coverage of the resulting pattern.

In order for the algorithm to construct a prime pattern of large coverage a heuristic criterion is used to choose the literal to be removed at each iteration. The removal of a literal is considered to be advantageous if the resulting pattern is "closer" to the set of positive points not covered by it than to the set of negative points.

In order to specify the heuristic, let us define the *disagreement* between a point β and a pattern P to be the number of literals of P whose values are zero on β. The disagreement between a set of points and a pattern is simply the sum of the disagreements between the pattern and every point in the set. Let us denote by $d_+(P)$ the disagreement between P and the set of positive points not covered by it. Similarly, let us denote by $d_-(P)$ the disagreement between the pattern and the negative points. Our computational experiments suggest that the ratio $\frac{d_+(P)}{d_-(P)}$ provides a good criterion for choosing the literal to be removed at each step. We describe in Figure 3.1 a pseudo-code for this algorithm. Obviously this algorithm can be restated for the construction of negative α-patterns.

```
1. Input: $\Omega^+, \Omega^-, \mathcal{P}(S) = \bigwedge_{i \in S} x_i^{\alpha_i}$– positive pattern.
2. For every $k \in S$ let $\mathcal{P}_k(S) = \bigwedge_{i \in S \setminus \{k\}} x_i^{\alpha_i}$.
3. If there is no $k \in S$ such that $\mathcal{P}_k(S)$ is a positive pattern
       then stop and output $\mathcal{P}(S)$.
4. Choose a $k \in S$ such that $\mathcal{P}_k(S)$ is a positive pattern and
   $\frac{d_+(\mathcal{P}_k(S))}{d_-(\mathcal{P}_k(S))}$ is minimized.
5. Set $S := S \setminus \{k\}$, $\mathcal{P}(S) = \bigwedge_{i \in S} x_i^{\alpha_i}$, and go to Step 2.
```

Figure 3.1: Algorithm for Enlarging Patterns to Maximized Prime Patterns (MPP).

This algorithm can be used to enlarge a minterm $\mathcal{P}_\alpha$ to a prime α-pattern, or more generally, to enlarge patterns generated by other algorithms to prime ones. Here we use the term "enlargement of a pattern" in the sense of increase of its coverage with respect to the dataset Ω. While the successive removal of elements from S shortens the pattern description, it is accompanied by a potential increase in the number of points of Ω that are covered by $\mathcal{P}(S)$.

One can easily see that the time complexity of this algorithm is $O(|\Omega|n^2)$. Therefore, constructing a heuristic maximum prime pattern for every point in the dataset takes $O(|\Omega|^2n^2)$ time.

3.4.3 Enlarging Patterns to Maximized Strong Patterns (MSP)

Let $\alpha \in \Omega^+$ and $\mathcal{P}$ be a positive α-pattern. We denote by $Cov_\Omega(\mathcal{P})$ the set of points of Ω covered by $\mathcal{P}$, and denote by $Lit(\mathcal{P})$ the index set of literals defining $\mathcal{P}$, i.e., $\mathcal{P} = \bigwedge_{i \in Lit(\mathcal{P})} x_i^{\alpha_i}$.

A positive pattern $\mathcal{P}$ is called *strong* if there is no positive pattern $\mathcal{P}'$ such that $Cov_\Omega(\mathcal{P}') \supset Cov_\Omega(\mathcal{P})$. It is known (see e.g. [66]) from experience that many strong patterns have very large coverage and their use in LAD leads to a superior performance. If $\mathcal{P}$ is not a strong pattern, we can apply the iterative algorithm described in [66] to transform it to a strong pattern $\mathcal{P}'$, such that $Cov_\Omega(\mathcal{P}') \supset Cov_\Omega(\mathcal{P})$.

As shown in [66] a strong pattern is not necessarily prime, even though one can transform it to a prime and strong pattern having the same coverage. On the other hand, a prime pattern is not necessarily strong and, as above, such a prime pattern $\mathcal{P}$ can be transformed to a strong pattern $\mathcal{P}'$, such that $Cov_\Omega(\mathcal{P}') \supset Cov_\Omega(\mathcal{P})$.

Let S be a non-empty subset of Ω^+, and let $[S]$ be the *Hamming convex hull* of the points in S, i.e., the smallest subcube containing S. A pattern $\mathcal{P}$ is called *spanned* if $\mathcal{P} = [Cov_\Omega(\mathcal{P})]$. As shown in [66] this definition is equivalent to saying that if I is the set of those indices i for which the corresponding components of all points of $Cov_\Omega(\mathcal{P})$ have the same value, say β_i, then

$$\mathcal{P} = \bigwedge_{i \in I} x_i^{\beta_i}.$$

The general algorithm described in [66] can be adapted so as to produce patterns of large coverage. For this purpose we use a heuristic criterion to choose the next point to be included in the coverage of the current pattern $\mathcal{P}$. The criterion selects a point $\beta \in \Omega^+ \backslash Cov_\Omega(\mathcal{P})$ such that $[Cov_\Omega(\mathcal{P}) \cup \{\beta\}]$ is a positive pattern, and $|Lit([Cov_\Omega(\mathcal{P}) \cup \{\beta\}])|$ is maximized. We describe in Figure 3.2 a pseudo-code for this algorithm.

```
1. Input: $\Omega^+, \Omega^-, \alpha \in \Omega^+, \mathcal{P}$ – positive $\alpha$-pattern, $S = Cov_\Omega(\mathcal{P})$.
2. $\mathcal{P}(S) := [S]$.
3. For every $\beta \in \Omega^+ \setminus S$ let $\mathcal{P}_\beta(S) = [S \cup \{\beta\}]$.
4. If there is no point $\beta \in \Omega^+ \setminus S$ such that
   $\mathcal{P}_\beta(S)$ is a positive pattern
       then stop and output $\mathcal{P}(S)$.
5. Choose $\beta \in \Omega^+ \setminus S$ such that $\mathcal{P}_\beta(S)$ is a positive
   pattern and $|Lit(\mathcal{P}_\beta(S))|$ is maximized.
6. Set $S := Cov_\Omega(\mathcal{P}_\beta(S))$ and go to Step 2.
```

Figure 3.2: Algorithm for Enlarging Patterns to Maximized Strong Patterns (MSP).

It is obvious from the definition that the pattern generated by this algorithm is not only strong, but also spanned. Furthermore, note that, since $\mathcal{P} = [Cov_\Omega(\mathcal{P})]$ (according to Theorem 4.5 in [66]), after $\mathcal{P}_\beta(S)$ is computed in Step 3, no additional computation is required in Step 2 of the following iteration.

A straightforward way to generate a strong α-pattern is to apply this algorithm to the minterm $\mathcal{P}_\alpha$. Another way of using this algorithm is to apply it to the patterns generated by any other pattern generating algorithms, e.g. the two heuristics described above, and thus possibly achieving an increase in the coverage of the patterns produced by them.

It is clear that the algorithm can be restated to construct negative α-patterns. One can easily see that the time complexity of this algorithm for constructing a heuristic maximum strong positive α-pattern is $O(|\Omega^+|^2|\Omega^-|n)$. Therefore, constructing a heuristic maximum strong pattern for every point in the dataset takes $O(|\Omega|^2|\Omega^+||\Omega^-|n)$ time.

3.4.4 Combined Heuristic Algorithms (CHAs)

Computational experiments show that in the case of large problems it takes a substantially shorter time to run *all* the heuristics described above than to solve problem (3.7) using a standard integer programming package. Thus, whenever the running time of solving problem (3.7) becomes prohibitively long, a computationally affordable alternative is to run *all* the heuristics, and combine their results, as described below.

A so-called "combined heuristic algorithm" (CHA) consists in: (i) choosing a subset of heuristics to use, (ii) running the chosen heuristics for every point in the dataset, and (iii) selecting for each $\alpha \in \Omega^+$ (respectively, Ω^-) a positive (respectively, negative) α-pattern of largest coverage from the collection of all patterns constructed in step (ii).

3.5 Fuzzy Patterns

As discussed in Chapter 2, when dealing with real-life datasets it is frequently necessary to relax the definition of a pattern, allowing the coverage of observations from its opposite class. In this chapter we introduce the concept of "fuzziness" of a pattern, which is closely related to that of homogeneity (see Chapter 2). The reason for utilizing a slightly different definition in this chapter is that some of the algorithms described here are more easily adapted to this definition than to the usual definition of homogeneity.

The fuzziness of a pattern is measured by a parameter which determines how many observations of the opposite class can be covered by that pattern. A family of positive (negative) patterns is said to have *fuzziness* φ if the percentage of negative (positive) observations covered by each pattern in the family does not exceed φ. A pattern with nonzero fuzziness is said to be a "fuzzy pattern."

The construction of fuzzy (positive) patterns can be accomplished by a simple modification of the constraints of model (3.7). The only constraints that have to be modified are those which have to hold for every $\gamma \in \Omega^-$; more precisely, for these constraints we shall require that

$$\sum_{\substack{j=1 \\ \gamma_j \neq \alpha_j}}^{n} y_j \geq 1 - s_\gamma, \quad \text{for every } \gamma \in \Omega^-$$

$$s_\gamma \in \{0,1\}, \text{ for every } \gamma \in \Omega^-,$$

and

$$\sum_{\gamma \in \Omega^-} s_\gamma \leq \varphi |\Omega^-|.$$

Exactly the same modification carries over to the best linear approximation-based heuristic, formulated as problem (3.8).

It is fairly straightforward to generalize the two combinatorial heuristic algorithms – Algorithm for Enlarging Patterns to Prime Patterns, and Algorithm for Enlarging Patterns to Strong Patterns – to the case of fuzzy patterns. The only modification to be made in the formulation of the algorithms is to replace everywhere "positive pattern" by "positive pattern with fuzziness at most φ".

Note that the utilization of fuzzy patterns allows to relax the assumption that Ω^+ and Ω^- are disjoint, by the weaker assumption that their intersection is "relatively small".

3.6 Robust Patterns

A common occurrence in real world datasets is the absence of the values of some of the attributes in certain observation points. This can be due either to missing information, or to measurement imprecisions leading to missing values when numerical data are binarized. Some of the standard approaches to dealing with such datasets consist either in the removal from the dataset of the observations or of the attributes with missing values, or in the filling in of the missing values with estimates obtained in a variety of ways (e.g. by using average values). While both approaches transform the original dataset with missing values into a fully specified one, the former approach discards possibly valuable information from the dataset, while the latter one introduces poorly justified modifications in the data. Since the reduction of datasets with missing values to completely specified ones is not satisfactory, we propose instead to work directly with datasets with missing values by appropriately extending the concept of patterns.

The concept of *robust patterns* (see [28]) extends that of patterns to the case of datasets with missing attribute values, always assuming a worst-case scenario. The worst-case assumption concerns the way of defining whether a pattern covers an observation with missing attribute values. More specifically, on the one hand, a positive observation with a missing attribute value in any of the variables appearing in a robust positive pattern will be considered not to be covered by that pattern (since the actual attribute value may conflict with the literal in the pattern). On the other hand, a negative observation with missing attribute values will be considered covered by a robust positive pattern if there is a combination of missing attribute values for which the corresponding completed observation is covered by that pattern. For example, the coverage of the positive pattern $x_1\overline{x}_2$ does not include the positive point $(1, -, 0)$, but does include the negative point $(-, 0, 1)$.

To generalize the algorithms presented in Sections 3.3 and 3.4 for the case of missing values and robust patterns we have to specify our algorithms as follows. In the case of problems (3.3) and (3.8), if the value of α_j is missing then the variable y_j will not appear in the formulation of the problems at all. Otherwise, if the value of β_j is missing then the condition $\beta_j \neq \alpha_j$ is considered to be satisfied. Additionally, if the value of γ_j is missing then the condition $\gamma_j \neq \alpha_j$ is considered to be not satisfied.

In the combinatorial heuristics described in subsections 3.4.2 and 3.4.3, the starting minterm $\mathcal{P}_\alpha$ includes only those literals for which the value of α_i is not missing. As a matter of fact, if any α_j is missing, then this conjunction is not a minterm anymore since it covers other Boolean vectors in addition to α. In the execution of the combinatorial heuristics the positive and negative coverage of prospective patterns is calculated using the worst-case approach described above.

3.7 Computational Evaluation of Maximum Pattern Generation Algorithms

We evaluated the performance of the heuristics described in the previous section in a series of computational experiments, first with artificially generated datasets, and then on ten publicly available datasets from the UC Irvine repository [97].

We generated three families of artificial datasets with binary attributes, each family being characterized by the type of target function defining the class to which each observation belongs. Each family is parameterized by the number of variables (n). The three families used in our experiments are:

(i) DNF(n)_i: is a randomly generated Boolean function in DNF form with the number of clauses randomly chosen between $0.5n$ and n, each clause having degree randomly chosen between $2\log_2(0.1n)$ and $4\log_2(0.1n)$. Observations on which this function evaluates to 1 are assigned to the positive class.

(ii) Linear(n)_i: is a linear function of the variables with randomly chosen coefficients in the range $[-1, 1]$ and a zero constant term. Observations on which this function evaluates to a positive value are assigned to the positive class, while those where it evaluates to a negative value are assigned to the negative class.

(iii) PB(n)_i: is a randomly generated pseudo-Boolean function with the number of terms randomly chosen between $0.5n$ and n, each term having degree randomly chosen between $2\log_2(0.1n)$ and $4\log_2(0.1n)$ and randomly generated coefficients in the range $[-1, 1]$. Observations on which this function evaluates to a positive value are assigned to the positive class, while those where it evaluates to a negative value are assigned to the negative class.

For each target function in the three families above we randomly generated a dataset consisting of $15n$ points in such a way that the number of points in the smaller class was at least $5n$. This was achieved by simply discarding the randomly generated points of the larger class after the cardinality of the larger class reached $10n$.

We generated 10 random problems of each family with $n = 20$, and 3 problems of each family with values of n equal to 50 and 100. We refer to the datasets with 20 attributes as "small" datasets, while the others are referred to as "large" ones. Since the number of positive observations and the number of negative observations in each dataset are random numbers in the range $[5n, 10n]$, the last two columns in the tables below specify the numbers of positive and negative observations in the dataset.

The computer used to run the maximum pattern generation algorithms was an *Intel Pentium* 4, 3.4GHz, with 2GB of RAM. The maximum pattern generation algorithms were implemented using the *MS Visual C++ .NET 1.1* compiler. Problems (3.7) and (3.8) were solved using version 16.10.02 of the Xpress-MP solver [43].

We report in Tables 3.1 and 3.2 the quality of the patterns obtained, as well as and the relative running time required, by applying the three heuristics described in Section 3.4, as well as the combinations CHA of MPP, MSP and BLA (referred to simply as CHA), and of MPP and MSP (referred to as MPSP). For the small datasets the quality of heuristically generated patterns is expressed in Table 3.1 [1] as the average percentage of the number of points covered by these patterns compared to the number of points covered by the optimum patterns constructed by solving exactly the corresponding integer programming problem (3.7). The relative running time is shown as a percentage of the running time required to solve the exact model (3.7), shown in the first column.

The solution of the exact model (3.7) becomes computationally prohibitive for the large datasets, and our attempts to solve this problem to optimality failed in the allocated maximum computing time of 24 hours. Therefore, we compare in Table 3.2 [2] the sizes of the patterns and the relative running time required by the heuristics MPP, MSP, BLA, and the combination MPSP to the size of the best pattern obtained with the CHA combination and the time required by CHA, presented in the first column.

As shown in Tables 3.1 and 3.2, the running time required for the solution of the exact model (3.7) can be orders of magnitude higher than that required to run a combination of the heuristics. As the number of variables and observations in the dataset increases (and, consequently, the number of binary variables in (3.7) also increases), solving (3.7) tends to become significantly more expensive than running one or more of the heuristics described.

It can be seen that the heuristic algorithms produced patterns whose average coverages ranged from 95.6% to 99.8% of those of the optimum patterns, whenever an optimal solution of problem (3.7) was found. For the large datasets this range is from 68.3% to 98.8% of the sizes of the patterns produced by CHA. The average time needed by the heuristics ranged from 1% to 15.4% of the time needed by the exact algorithm in the

[1] EMP: exact maximum patterns; BLA: best linear approximation heuristic; MPP: maximized prime patterns; MSP: maximized strong patterns; CHA: combination of three heuristics; MPSP: combination of prime and strong heuristics.

[2] CHA: combination of three heuristics; BLA: best linear approximation heuristic; MPP: maximized prime patterns; MSP: maximized strong patterns; MPSP: combination of prime and strong heuristics.

	EMP	**BLA**		**MPP**		**MSP**		**CHA**		**MPSP**		**Observations**	
Dataset	Time (s)	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Pos.	Neg.
DNF(20)_1	1436.0	0.9%	99.3%	2.5%	93.4%	4.6%	93.0%	8.0%	99.5%	7.1%	97.8%	200	100
DNF(20)_2	1418.3	1.1%	97.8%	2.6%	81.5%	3.9%	92.8%	7.5%	99.2%	6.5%	96.9%	200	100
DNF(20)_3	1538.9	0.8%	99.4%	2.3%	91.9%	4.0%	95.6%	7.1%	99.8%	6.3%	98.3%	200	100
DNF(20)_4	1453.2	0.9%	98.8%	2.4%	93.3%	4.3%	95.9%	7.6%	99.6%	6.7%	99.0%	200	100
DNF(20)_5	780.2	0.9%	99.7%	4.7%	97.5%	11.0%	97.6%	16.6%	99.9%	15.7%	99.6%	200	100
DNF(20)_6	963.8	0.9%	99.5%	3.6%	94.0%	7.2%	96.1%	11.7%	99.8%	10.8%	98.5%	200	100
DNF(20)_7	920.0	0.9%	99.7%	4.0%	96.8%	7.6%	97.9%	12.5%	99.9%	11.6%	99.2%	200	100
DNF(20)_8	926.3	0.9%	99.1%	3.8%	95.3%	7.5%	94.2%	12.3%	99.3%	11.4%	98.4%	200	100
DNF(20)_9	1619.6	1.3%	97.7%	2.2%	77.4%	3.1%	89.6%	6.6%	99.1%	5.3%	94.7%	200	100
DNF(20)_10	845.3	1.1%	99.6%	4.2%	95.4%	8.9%	94.4%	14.2%	99.8%	13.1%	98.6%	200	100
Linear(20)_1	2411.5	0.8%	99.4%	1.5%	93.9%	4.2%	96.3%	6.5%	99.8%	5.7%	98.0%	152	148
Linear(20)_2	2656.6	0.5%	99.6%	1.4%	97.6%	4.0%	95.6%	5.9%	99.9%	5.3%	99.1%	186	114
Linear(20)_3	2785.4	0.5%	99.7%	1.3%	97.7%	4.1%	94.8%	5.9%	99.9%	5.3%	98.7%	100	200
Linear(20)_4	2088.7	0.7%	99.8%	1.8%	98.6%	5.1%	97.6%	7.6%	99.9%	6.9%	99.5%	126	174
Linear(20)_5	2165.3	0.5%	100.0%	1.7%	97.7%	5.4%	95.4%	7.6%	100.0%	7.1%	99.9%	100	200
Linear(20)_6	1060.9	0.7%	99.9%	3.3%	99.1%	10.8%	99.9%	14.8%	100.0%	14.1%	100.0%	200	100
Linear(20)_7	2874.1	0.6%	99.9%	1.3%	96.9%	3.1%	96.3%	5.0%	99.9%	4.4%	98.8%	188	112
Linear(20)_8	1665.9	0.6%	100.0%	2.2%	99.7%	6.9%	99.9%	9.7%	100.0%	9.1%	100.0%	100	200
Linear(20)_9	2508.5	0.6%	99.8%	1.4%	99.0%	4.2%	97.6%	6.2%	99.9%	5.6%	99.3%	100	200
Linear(20)_10	2562.4	0.6%	100.0%	1.5%	96.6%	3.8%	98.2%	5.9%	100.0%	5.3%	98.9%	181	119
PB(20)_1	497.8	1.7%	100.0%	7.2%	98.4%	29.1%	96.8%	38.0%	100.0%	36.3%	99.5%	100	200
PB(20)_2	1056.6	0.7%	99.9%	3.4%	99.2%	10.9%	100.0%	15.0%	100.0%	14.3%	100.0%	200	100
PB(20)_3	911.9	0.9%	99.9%	4.0%	98.4%	13.3%	99.6%	18.2%	99.9%	17.3%	99.7%	100	200
PB(20)_4	301.4	2.5%	99.9%	12.0%	99.4%	43.9%	99.7%	58.3%	99.9%	55.9%	99.9%	184	116
PB(20)_5	1016.3	0.9%	99.6%	3.6%	98.0%	10.8%	99.3%	15.2%	100.0%	14.4%	100.0%	200	100
PB(20)_6	297.2	2.3%	100.0%	12.2%	88.3%	46.1%	100.0%	60.5%	100.0%	58.2%	100.0%	200	100
PB(20)_7	1291.7	0.8%	99.7%	2.8%	96.8%	8.6%	99.7%	12.2%	100.0%	11.5%	99.9%	132	168
PB(20)_8	472.3	1.5%	100.0%	7.7%	99.8%	23.6%	99.8%	32.7%	100.0%	31.3%	100.0%	100	200
PB(20)_9	663.5	1.0%	100.0%	5.4%	100.0%	17.6%	100.0%	24.1%	100.0%	23.1%	100.0%	100	200
PB(20)_10	1445.6	0.7%	99.9%	2.5%	96.4%	6.3%	99.7%	9.5%	100.0%	8.7%	99.9%	100	200
Average		**1.0%**	**99.6%**	**3.7%**	**95.6%**	**10.8%**	**97.1%**	**15.4%**	**99.8%**	**14.5%**	**99.1%**		

Table 3.1: Relative time and size of heuristically generated patterns for small datasets (as percentage of size and time of the exact model).

case of small datasets, and from 22.3% to 51.5% of the time spent by the CHA procedure in the case of large datasets. In view of the very high coverages of the patterns produced by CHA, the computational expense of solving the exact model is not justified.

In Table 3.3 [3] we report the quality of patterns produced, and the running time required by the heuristics when applied to a set of ten problems from the UCI machine learning repository [97]. Table 3.3 shows the relative running time and quality of patterns produced by the heuristics, as compared to those of the exact algorithm EMP, whenever the solution of (3.7) finished within the limit of 24 hours. In the case of problems "krkp" and "sick" we compared the running time and quality of patterns produced

[3]EMP: exact maximum patterns; MPP: maximized prime patterns; MSP: maximized strong patterns; BLA: best linear approximation heuristic; CHA: combination of three heuristics; MPSP: combination of prime and strong heuristics. Whenever EMP did not terminate after the allocated 24h limit, we report the running time of CHA and present the relative sizes and running times of the other algorithms as a percentage of CHA's size and running time.

	CHA	BLA		MPP		MSP		MPSP		Observations	
Dataset	Time (s)	Time	Size	Time	Size	Time	Size	Time	Size	Pos.	Neg.
DNF(50)_1	3,051.9	53.0%	99.3%	22.8%	59.9%	24.2%	64.0%	47.0%	73.2%	500	250
DNF(50)_2	2,499.8	40.4%	99.1%	27.7%	70.1%	31.9%	80.2%	59.6%	85.3%	500	250
DNF(50)_3	2,963.5	51.3%	99.3%	23.8%	64.6%	24.9%	70.4%	48.7%	78.6%	487	263
Linear(50)_1	2,734.6	27.0%	99.8%	25.7%	82.0%	47.3%	69.5%	73.0%	85.6%	365	385
Linear(50)_2	2,177.7	23.5%	99.9%	31.5%	85.4%	44.9%	78.8%	76.5%	91.0%	250	500
Linear(50)_3	2,167.8	20.1%	99.9%	31.5%	90.7%	48.4%	87.5%	79.9%	94.5%	250	500
PB(50)_1	3,205.3	40.4%	98.9%	21.9%	65.7%	37.7%	69.6%	59.6%	83.1%	399	351
PB(50)_2	2,586.3	42.9%	98.9%	26.7%	64.0%	30.4%	65.5%	57.1%	73.8%	500	250
PB(50)_3	2,642.8	42.7%	99.0%	26.5%	71.7%	30.8%	71.6%	57.3%	81.9%	338	412
DNF(100)_1	126,212.3	80.7%	98.8%	10.8%	53.8%	8.5%	68.1%	19.3%	76.2%	615	885
DNF(100)_2	116,342.9	80.8%	98.8%	9.5%	55.2%	9.7%	74.4%	19.2%	77.2%	558	942
DNF(100)_3	112,230.5	73.3%	98.2%	10.0%	52.9%	16.7%	73.6%	26.7%	76.8%	500	1000
Linear(100)_1	42,990.3	49.6%	99.5%	25.2%	74.7%	25.2%	63.4%	50.4%	79.0%	1000	500
Linear(100)_2	43,568.0	48.6%	99.6%	25.0%	70.4%	26.3%	59.3%	51.4%	73.4%	500	1000
Linear(100)_3	42,759.3	46.9%	98.6%	26.6%	83.2%	26.4%	66.5%	53.1%	86.7%	965	535
PB(100)_1	84,151.5	74.4%	98.3%	13.2%	62.3%	12.4%	67.3%	25.6%	79.0%	815	685
PB(100)_2	63,301.1	65.8%	95.5%	17.5%	65.4%	16.7%	79.4%	34.2%	84.9%	916	584
PB(100)_3	103,915.6	64.9%	96.8%	25.5%	57.0%	9.6%	71.4%	35.1%	76.7%	578	922
Average		**51.5%**	**98.8%**	**22.3%**	**68.3%**	**26.2%**	**71.1%**	**48.5%**	**80.9%**		

Table 3.2: Relative time and size of heuristically generated patterns for large datasets (as percentage of size and time of the CHA algorithm).

	EMP	BLA		MPP		MSP		CHA		MPSP		Observations		Attribu
Dataset	Time (s)	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Pos.	Neg.	Attr.
breast-w	54,601.4	0.1%	98.6%	2.9%	98.4%	3.4%	97.5%	6.5%	99.5%	6.4%	99.1%	241	458	9
credit-a	160.8	15.7%	100.0%	31.9%	99.6%	29.9%	99.9%	77.5%	100.0%	61.7%	100.0%	307	383	14
hepatitis	588.5	0.9%	91.5%	13.6%	86.5%	4.5%	91.8%	18.9%	96.7%	18.0%	95.1%	32	123	19
krkp	132,996.1*	0.6%	99.6%	4.6%	99.5%	94.8%	99.7%	100.0%	100.0%	99.4%	100.0%	1,669	1,527	36
boston	109.3	18.8%	99.1%	89.9%	97.0%	13.2%	99.8%	122.0%	99.8%	103.1%	99.8%	250	256	13
bupa	40.7	19.2%	98.6%	76.6%	97.2%	10.1%	100.0%	105.9%	100.0%	86.7%	100.0%	200	125	6
heart	2,113.4	0.6%	92.8%	9.2%	87.0%	4.9%	88.5%	14.8%	97.1%	14.1%	95.0%	139	164	13
pima	2,113.4	11.1%	94.7%	137.1%	90.1%	7.0%	99.0%	155.3%	99.4%	144.1%	99.2%	268	500	8
sick	41,784.9*	5.6%	99.8%	23.9%	94.0%	70.5%	99.0%	100.0%	100.0%	94.4%	99.3%	231	3,541	26
voting	55.8	21.6%	99.9%	14.7%	93.5%	58.7%	93.1%	95.0%	100.0%	73.4%	93.5%	267	168	16

Table 3.3: Relative time and size of heuristically generated patterns for UCI datasets (as percentage of size and time of the exact algorithm EMP).

by the heuristics with those of the combined algorithm CHA. We also include in Table 3.3 the numbers of positive and negative observations in each dataset, as well as the number of attributes in the original datasets and the number of binary attributes obtained after discretizing each original dataset. Clearly, the results on the UCI datasets corroborate those observed in our experiments with synthetic datasets.

The results of all these experiments suggest that the BLA heuristic is the most effective individual heuristic among those proposed here. However, it relies on solving an NP-complete problem and, therefore, as the size of the problem grows (number of attributes in the datasets increases), it becomes increasingly more expensive to solve. For small problems, the BLA algorithm performs substantially faster than any of the two combinatorial heuristics MPP or MSP. Moreover, the patterns produced by BLA are consistently larger than those produced by MPP or MSP, and frequently larger than those obtained with the combination MPSP. In the case of typical machine learning problems (such as the UCI problems studied here) BLA is clearly the heuristic of choice. Unfortunately, when the problems become sufficiently large ($n \geq 100$ in the case of the synthetic problems examined here) the BLA heuristic becomes prohibitively expensive.

On the other hand, the quality of the patterns produced with the use of the two combinatorial procedures (MPP and MSP) is still quite reasonable. While the computing time for the combinatorial algorithms was higher than the time required by the BLA heuristic in the case of smaller problems, the combinatorial heuristics remained the only viable options in the case of instances of larger sizes. Moreover, in a case when an efficient integer linear programming solver is not available, the use of the combinatorial heuristics becomes an invaluable option. Note that as the estimates of computational complexity of these two heuristics (which are provided in Sections 3.4.2 and 3.4.3) indicate, the running time of MPP is quite sensitive to the number of binary variables, while the running time of MSP is much more sensitive to the number of observations in the dataset. Therefore, in the case of very large problems, one would have to resort to the use of only one of these heuristics, with the choice depending on whether the number of variables or the number of observations is exceedingly large.

3.8 Application to Classification

One of the most important applications of maximum patterns (or their heuristically generated approximations) is in the Logical Analysis of Data (LAD). Patterns are the building blocks of LAD models, and the accuracy of these models depends on the type of patterns used for their construction. It is therefore important to empirically evaluate the effect of using (exact or approximate) maximum patterns on the accuracy of LAD classification models. We present below the results of such evaluation carried out on the ten datasets used in the previous section.

Given a dataset $\Omega = \Omega^+ \cup \Omega^-$, we recall that a collection of positive and negative patterns $\Pi^+ \cup \Pi^-$ is called a LAD model, if every $\omega^+ \in \Omega^+$ is covered by at least one pattern in Π^+, and every $\omega^- \in \Omega^-$ is covered by at least one pattern in Π^-. For the purpose of classifying a "new" observation $\omega \notin \Omega$ in the experiments described here, we utilized the usual definition of a discriminant function $\Delta(\omega)$:

$$\Delta(\omega) = \frac{\pi^+(\omega)}{|\Pi^+|} - \frac{\pi^-(\omega)}{|\Pi^-|}.$$

We evaluated the accuracy of classification models in this chapter using one random 10-fold cross-validation. For the purpose of comparing our results with those of algorithms implemented in the Weka package [128], we utilized the LAD accuracy definition, presented in Chapter 2.

Each of the LAD models used in our experiments was built on a collection of maximum patterns, constructed by one of the exact or heuristic algorithms described in the previous sections. The only exception to this statement concerns the CAP model (*Combined Approximate Patterns*), which uses the union of the two pattern collections constructed by the heuristic algorithms MPP and MSP which separately enlarge each minterm of Ω to a maximized prime, respectively strong pattern. In Table 3.4 the combination CHA utilizes all three heuristics BLA, MPP and MSP.

Tables 3.4 and 3.5 below report the accuracy of various classification algorithms obtained as the average result of a 10-fold cross-validation experiment. In both tables we display the highest accuracy for each dataset in boldface characters. Moreover, at the bottom of the tables we display the so-called Borda count [21] of each algorithm. The Borda count in Table 3.4 is obtained by ranking the seven algorithms based on their accuracies on a given dataset. The most accurate algorithm is assigned a score of 7, the second best is assigned a score of 6, and so on until a score of 1 is assigned to the algorithm with the worst accuracy on that dataset. The Borda count of an algorithm is the sum of these scores over the ten datasets. The Borda count for Table 3.5 is computed analogously.

The column labeled "LAD" in Table 3.4 refers to the results obtained using the current implementation of LAD described in [11]. This and other LAD systems following the implementation framework described in [28] utilize several important parameters and, therefore, have to be extensively calibrated to achieve the best possible results.

Dataset	LAD	EMP	BLA	MPP	MSP	CHA	CAP
breast-w	0.945 ± 0.011	0.949 ± 0.017	0.953 ± 0.017	0.953 ± 0.014	0.952 ± 0.019	0.957 ± 0.017	**0.966 ± 0.011**
credit-a	0.840 ± 0.031	***	0.840 ± 0.029	0.833 ± 0.034	0.835 ± 0.031	0.762 ± 0.050	**0.867 ± 0.025**
hepatitis	0.770 ± 0.056	0.713 ± 0.107	0.743 ± 0.072	**0.785 ± 0.104**	0.712 ± 0.069	0.760 ± 0.099	0.779 ± 0.104
krkp	0.848 ± 0.006	***	0.992 ± 0.003	0.534 ± 0.007	0.990 ± 0.005	0.530 ± 0.010	**0.993 ± 0.002**
boston	0.848 ± 0.021	0.724 ± 0.044	**0.867 ± 0.019**	0.860 ± 0.031	0.847 ± 0.031	0.668 ± 0.075	0.855 ± 0.029
bupa	0.665 ± 0.053	0.640 ± 0.047	0.663 ± 0.055	0.673 ± 0.043	0.614 ± 0.037	0.668 ± 0.052	**0.734 ± 0.052**
heart	**0.832 ± 0.041**	0.812 ± 0.043	0.826 ± 0.031	0.825 ± 0.026	0.809 ± 0.029	0.823 ± 0.031	0.826 ± 0.032
pima	0.741 ± 0.026	0.577 ± 0.029	0.725 ± 0.027	0.743 ± 0.025	0.697 ± 0.025	0.575 ± 0.028	**0.747 ± 0.022**
sick	0.720 ± 0.020	0.722 ± 0.036	0.734 ± 0.033	0.713 ± 0.033	0.695 ± 0.023	0.739 ± 0.035	**0.825 ± 0.019**
voting	0.953 ± 0.019	**0.956 ± 0.020**	0.949 ± 0.012	0.949 ± 0.014	0.947 ± 0.008	0.945 ± 0.021	0.952 ± 0.021
Borda count	45	25	45	46	24	31	64

Table 3.4: Classification accuracy and Borda counts of LAD using different types of patterns. Entries marked with "***" could not be finished within the total running time limit of 48 hours.

The LAD results reported here were obtained after using only moderate calibration and can potentially be improved with more extensive calibration. In contrast, all the algorithms proposed in this chapter utilize only a single parameter of fuzziness and, therefore, require only minimal calibration.

Note that none of the new algorithms for constructing LAD models shown in Table 3.4 clearly dominates the others in terms of accuracy. The results in Table 3.4 indicate that the computational expense of producing patterns achieving the exact maximum coverage does not translate into an (even marginally) superior classification performance of the resulting EMP model of LAD, since the LAD models using heuristically generated patterns are always of comparable quality (being even better on many datasets). Furthermore, in spite of the fact that CHA chooses for every point the best of the three patterns generated by BLA, MPP and MSP, the CHA model does not always perform better than its individual components. Overall it looks like the CAP model of LAD provides a reasonable compromise between achieving superior classification performance and keeping the computational expense reasonably low. The accuracies of the CAP models are at least comparable to and often exceed the highest accuracies of the models constructed by the other algorithms, as well as the accuracies of the current implementation of LAD [11].

Based on the above, we shall use the CAP model of LAD (termed CAP-LAD in the comparisons presented below) as LAD's reference implementation. A significant advantage of this reference implementation of LAD is its non-reliance on various control parameters (utilized in the original framework of LAD [28]), the only one still used being the fuzziness parameter φ. This drastically reduces the amount of fine tuning and therefore simplifies the usage of the LAD methodology.

In Table 3.5 we present the cross-validated accuracies of CAP-LAD and of five commonly used machine learning algorithms as implemented in the publicly available software package Weka [128]: Support Vector Machines (SMO), C4.5 Decision Trees (J48), Random Forests (Rand.For.), Multilayer Perceptron (Mult.Perc.), and Simple Logistic Regression (S.Log.). The comparison of these accuracies shows that CAP-LAD is a very competitive classification method, whose accuracy is on par with the accuracies of most accurate of the Weka algorithms.

Dataset	SMO	J48	Rand.For.	Mult.Perc.	S. Log.	CAP-LAD
breast-w	0.965 ± 0.011	0.939 ± 0.012	**0.967 ± 0.009**	0.956 ± 0.012	0.963 ± 0.013	0.966 ± 0.011
credit-a	0.864 ± 0.025	0.856 ± 0.031	**0.882 ± 0.027**	0.831 ± 0.032	0.873 ± 0.025	0.867 ± 0.025
hepatitis	0.772 ± 0.084	0.652 ± 0.086	0.722 ± 0.101	0.727 ± 0.065	0.764 ± 0.090	**0.779 ± 0.104**
krkp	**0.996 ± 0.003**	0.994 ± 0.003	0.992 ± 0.003	0.993 ± 0.002	0.975 ± 0.005	0.993 ± 0.002
boston	0.889 ± 0.028	0.837 ± 0.045	0.875 ± 0.024	**0.893 ± 0.031**	0.874 ± 0.021	0.855 ± 0.029
bupa	0.701 ± 0.045	0.630 ± 0.041	0.731 ± 0.046	0.643 ± 0.020	0.662 ± 0.048	**0.734 ± 0.052**
heart	**0.837 ± 0.039**	0.799 ± 0.052	0.834 ± 0.051	0.815 ± 0.025	0.834 ± 0.040	0.826 ± 0.032
pima	0.727 ± 0.029	0.722 ± 0.026	0.736 ± 0.030	0.726 ± 0.023	0.729 ± 0.031	**0.747 ± 0.022**
sick	0.824 ± 0.027	**0.926 ± 0.020**	0.832 ± 0.023	0.852 ± 0.049	0.808 ± 0.023	0.825 ± 0.019
voting	0.961 ± 0.018	0.960 ± 0.015	0.961 ± 0.016	0.944 ± 0.025	**0.961 ± 0.014**	0.952 ± 0.021
Borda count	42	22	43	27	35	41

Table 3.5: Classification accuracy and Borda counts of Weka algorithms and CAP-LAD.

In order to perform a fair comparison between CAP-LAD and the Weka algorithms, we calibrated a few parameters of each of the five Weka algorithms. Below we list for each type of classifier the parameters and their associated values used for calibration. A 10-fold experiment was carried out with each possible combination of parameters and the highest average accuracy obtained in those experiments is reported in Table 3.5.

Support Vector Machines (SMO): (i) C: complexity parameter, which controls the tradeoff between the maximization of the margin of separation and the minimization of the misclassification rate in the training set (taking values 1, 10, and 50); (ii) exponent: exponent of polynomial kernel, when using a standard polynomial kernel (taking values 1, 2, 3); and (iii) useRBF: whether or not to use a radial basis function kernel, instead of a polynomial one.

Multilayer Perceptron: (i) learningRate: amount by which the weights are updated (taking values 0.3 and 0.5); (ii) momentum: momentum applied to the weights during updating (taking values 0.2 and 0.4); and (iii) hiddenLayers: number of hidden layers in the network (taking values "i" and "t", which correspond to the number of attributes and to the sum of the number of attributes and the number of classes in the dataset, respectively.

Simple Logistic Regression: (i) heuristicStop: the fitting of logistic models is stopped if no new error minimum has been reached in the last "heuristicStop" iterations (taking values 50, 100); (ii) maxBoostingIterations: maximum number of LogitBoost iterations performed while fitting the logistic models (taking values 100, 500); and (iii) errorOnProbabilities: whether or not to use error on the probabilities as a measure when determining the best number of LogitBoost iterations.

Random Forests: (i) numFeatures: number of features to be used in random selection (taking values 2 and $\lfloor log_2(n) + 1) \rfloor$; and (ii) numTrees: number of trees to be generated (10, 100, 1000).

Decision Trees C4.5 (J48): (i) reducedErrorPruning: whether or not to use reduced-error pruning; (ii) binarySplits: whether or not to use binary splits on nominal attributes when building the tree; and (iii) minNumObj: minimum number of instances per leaf (taking values 1 and 2).

Table 3.6 summarizes the results shown in Table 3.5. It reports the pairwise comparisons of the five algorithms from Weka and CAP-LAD. Each cell entry contains the number of wins, losses and ties (respectively) between the algorithm corresponding to the row and the one corresponding to the column, over the ten datasets used to construct Table 3.5. Every comparison here is performed on the basis of a t test at 95% confidence level. As discussed in [45], this test is prone to errors of Type I. Therefore, the conclusion that the performance of a certain algorithm is superior to that of another one on an individual instance is to be regarded cautiously, while the conclusion that the accuracies of two algorithms are not statistically different at the 95% confidence level on a given instance can be trusted. Based on this insight, we can conclude that the performance of CAP-LAD is indistinguishable from that of Random Forests and SVMs – despite the (slight) difference in the Borda counts of these algorithms – while being at least as good as the performances of the other three algorithms from Weka.

	SMO	J48	Rand.For.	Mult.Perc.	S. Log.
J48	4-1-5				
Rand.For.	1-1-8	4-1-5			
Mult.Perc.	0-4-6	2-2-6	0-2-8		
S.Log.	1-1-8	1-2-7	0-2-8	1-2-7	
CAP-LAD	0-1-9	2-1-7	0-0-10	2-1-7	1-0-9

Table 3.6: Matrix of wins, losses and ties.

3.9 Conclusions

In this chapter we investigated the problem of constructing maximum coverage patterns containing any given observation point in the dataset. The main contributions of this chapter include the development of both exact and heuristic methods for the construction of such maximum patterns. The results obtained here show that the heuristically constructed patterns can achieve more than 81%-98% of the maximum possible coverage, while requiring only a fraction of the computing time of the exact method. These results were shown to hold on both real-life datasets from the UCI machine learning repository [97], as well as on varied synthetic datasets.

We showed that maximum patterns are useful for constructing highly accurate LAD classification models. One of the advantages of such models is that in contrast with the original LAD models, the maximum pattern based ones do not require extensive calibration (determination of good values of several control parameters), and can therefore be easily used by non-experts.

It is also interesting that the LAD models built using the exact patterns of maximum coverage do not exhibit any superior classification performance as compared to the models built using approximate maximum patterns – in agreement with similar phenomena observed in different machine learning contexts [44].

In comparisons with the commonly used machine learning algorithms implemented in the publicly available Weka software package, the proposed reference implementation of LAD (termed CAP-LAD) was shown here to be a highly competitive classification algorithm.

Chapter 4

A Branch-and-Bound Algorithm

4.1 Introduction

In this chapter we describe a branch-and-bound algorithm for a family of pseudo-Boolean optimization problems, and report the results of a series of computational experiments that compare the solution quality and running time of our algorithm with those of the Xpress [43] integer linear programming solver.

The branch-and-bound approach dates back to 1960 when the pioneer work of Land and Doig [84] was published, in which they described a novel algorithm for solving integer linear programming problems. Since then, several versions of the algorithm were proposed for special classes of optimization problems, such as the traveling salesman problem [55, 99, 102], facility location [53, 94, 117], network design [42, 64, 75], nonlinear programming problems [3, 71, 121, 124], to name a few.

The algorithm is essentially a combination of two procedures: the partitioning of the original problem into a series of successively simpler subproblems (*branching*), and the computation of bounds that permit the elimination of a significant fraction of the subproblems (*bounding*). The expectation is that large parts of the search space can be removed from consideration and only a small number of solutions have to be actually computed.

The branching procedure basically partitions the space of feasible solutions. The typical branching operation consists of (i) enumerating (or implicitly describing) the possible values of one or more variables that can actually be realized in a feasible solution, and (ii) partitioning the space of feasible solutions according to those values. Each partition is called a *branch* and creates a restricted version of the original problem.

Therefore, the objective function value in each branch can be at most as good as that in the original problem. Note, however, that branching operations as described here do not remove any feasible solution from consideration. Thus, after a branching operation the set of solutions of a problem is completely preserved in its subproblems.

Due to the successively more refined partitions of the search space generated by the algorithm, we often refer to the *search tree* or the *branch-and-bound tree* associated to the application of a branch-and-bound algorithm to a particular instance. At the root of the tree we have a special node corresponding the original problem. Each of the remaining nodes of such a tree corresponds to a version of the original problem whose space of feasible solutions has been reduced by a branching operation. Two nodes are connected by an arc if one of them was obtained from the other by the application of a branching operation, i.e., one of them is a restricted version of the other. We usually say that the restricted subproblem is a *descendant* or a *child* node of the other, which is referred to as the *parent* node. In this chapter we will sometimes identify a node in the search tree with the subproblem associated to it.

The bounding procedure relies on the computation of an explicit bound on the optimal objective function value for a given problem. Let us assume that our original problem is a maximization one, and let z_f be the objective function value associated to a given feasible solution. We can use an upper bound on the optimal objective function value of a subproblem to conclude that it cannot contain an optimal solution to the original problem. If the upper bound on the optimal value of a given subproblem is $\hat{z}$, and $\hat{z} < z_f$, then it is clear that the given subproblem cannot contain an optimal solution to the original problem.

In most applications, the bounding procedure produces a solution for a relaxation of the subproblem, and that can be used as a candidate solution for the original problem. For example, in the case of integer linear programming problems, solving the linear relaxation of the current problem S provides a bound on the objective value of S while producing a candidate solution x for the original problem. If x turns out to be integral and the objective function value associated to it is better than that of the best solution of the original problem produced so far, we declare x the best known solution so far.

If x is fractional, problem S can be either discarded or further branched on, depending on the value of the bound computed: if the objective function value associated to x is better than that of the current best known solution, then problem S is still considered for further branching; otherwise, x is discarded, and the current subproblem is removed from the search tree (as discussed above) without producing any descendant node. If the procedure for computing a bound on the optimal objective value of a subproblem does not produce a candidate solution, some mechanism of producing one or more candidate solutions from each subproblem is needed as a way of generating feasible solutions for the original problem.

More recently many modifications of the original branch-and-bound algorithm have been proposed based on combinations of the original principles of branching and bounding with those of other techniques, such as primal heuristics [12, 15, 98], cutting planes [55, 74, 99, 100, 102, 124], column generation [14, 15, 111, 123], and constraint satisfaction [2, 87, 101, 125]. These variants are mainly used for solving specific classes of problems, as they typically explore the structure of the problem being solved, for instance by the use of valid inequalities and special-purpose heuristics. For the solution of real-life problems, commercial solvers rely heavily on branch-and-bound algorithms, implementing some version of the standard algorithm enriched by a number of preprocessing techniques, heuristics for fixing of variables, branching and node selection strategies [2, 43, 74, 112, 50], and general-purpose cutting planes [61, 62, 36].

Certain classes of combinatorial optimization problems possess structural properties that allow the opportunistic use of branch-and-bound algorithms. In this chapter we describe a class of pseudo-Boolean optimization problems [22, 23, 40, 67, 68, 69] with applications in Logical Analysis of Data, and present a simple branch-and-bound algorithm that satisfactorily solves problems from this class.

In the next section we describe the pseudo-Boolean optimization problem considered in this chapter. Section 4.3 presents the branching strategies of our algorithm, and Section 4.4 describes the node selection strategy utilized. In Section 4.5 we discuss the bounding procedure, and in Section 4.6 we report on computational experiments carried out on randomly generated instances. Finally, Section 4.7 discusses the main contributions in this chapter.

4.2 Problem Formulation

Let us consider a binary dataset $\Omega \subset \{0,1\}^n$, with $|\Omega| = m$, and let us associate to each point $\omega^k \in \Omega$ a real weight β_k. Consider now the problem of finding a conjunction C such that the sum of the weights of the points in Ω satisfied by C is maximized. The following pseudo-Boolean optimization problem asks for such a conjunction:

$$\begin{array}{lll} \text{(PB)} & \text{maximize} & \phi = \sum_{k=1}^{m} \beta_k \left(\prod_{i:\omega_i^k=0} \overline{y_i} \prod_{j:\omega_j^k=1} \overline{z_j} \right) \\ & \text{subject to:} & y_j, z_j \in \{0,1\},\ j = 1, \ldots, n, \end{array}$$

where y_j is a Boolean variable associated to the inclusion of literal x_j in the resulting conjunction C, and z_j is a Boolean variable representing the inclusion of literal $\overline{x_j}$ in C, for $j = 1, \ldots, n$. Note that the weights β_k $(k = 1, \ldots, m)$ do not have necessarily the same sign.

Problem (PB) describes a particular class of pseudo-Boolean optimization problems in which every term has degree equal to n and variables appear only complemented in the expression of ϕ. As a simple example, let $n = 3$, $\Omega = \{(1,0,1), (1,1,0)\}$, $\beta_1 = 1$ and $\beta_2 = -1$. Then, function ϕ equals to $\overline{z_1}\,\overline{y_2}\,\overline{z_3} - \overline{z_1}\,\overline{z_2}\,\overline{y_3}$.

Note that the maximum pattern problem described in Chapter 3 is a special case of problem (PB) if the fuzziness parameter is set to zero. In such a case, we assume that $\Omega = \Omega^+ \cup \Omega^-$, with $\Omega^+ \cap \Omega^- = \emptyset$, and we ask, for instance, for a conjunction that covers the largest number of points from Ω^+ and does not cover any of the points in Ω^-. This is equivalent to setting weights $\beta_j = 1$ for the points $\omega^j \in \Omega^+$, $\beta_j = -|\Omega^+|$ for the points $\omega^j \in \Omega^-$, and solving problem (PB).

Problem (PB) can also be used to produce maximum patterns with nonzero fuzziness by a proper setting of the weights β_j. In that case, however, (PB) is not equivalent to the formulation described in Chapter 3, since we cannot impose a nontrivial strict upper bound on the number of points from Ω^- that are covered. On the other hand, by adjusting the weights β_j we can obtain fuzzy patterns with optimal coverage with respect to a criterion closer to the one used in the context of subgroup discovery [86]. In Chapters 5 and 6 we will present other applications in which problem (PB) and a slight variation of it arise naturally.

4.3 Branching Strategy

A simple branch-and-bound algorithm for a problem with binary variables fixes one variable at each branch. In such a case, every subproblem in the branch-and-bound algorithm produces exactly two descendants, one corresponding to fixing $x_j = 0$ and the other to fixing $x_j = 1$. The enumeration associated to such a branching policy is typically very large, causing such an algorithm to be of little practical use. In general, if a simple branching strategy is to be used, it becomes necessary to use preprocessing techniques that substantially simplify the problem (reducing the number of variables or the space of feasible solutions) and to develop sharp bounds to prune large parts of the search tree.

Some classes of problems display certain structural properties that allow the development of specific branching strategies that fix a larger number of variables at each branch of the search tree, quickly reducing the size of the subproblems during the execution of the algorithm. In this section we describe a branching strategy that takes advantage of the particular structure of problem (PB) and attempts to fix a large number of variables at each branch.

We propose a branching rule that takes into account the large number of variables involved in the clauses (or terms) in problem (PB). We perform the branching step on an entire clause of ϕ, instead of on a single variable. Branching on a clause $T = \prod_{i \in A} \overline{y_i} \prod_{j \in B} \overline{z_j}$ can fix a large number of variables at once, significantly reducing the sizes of the resulting subproblems. For instance, if we fix clause T at value 1, we simultaneously fix all variables $y_i = 0$ $(i \in A)$ and $z_j = 0$ $(j \in B)$. Fixing $T = 0$ leaves us with a number of options: fixing any of the variables involved in T at value 1 is enough to ensure that $T = 0$. We can then create $|A| + |B|$ descendant nodes associated with the case $T = 0$, with an increasing number of variables fixed over these nodes. To achieve that, let us assume some order among the variables involved in T. We can fix the first variable at the value 1, leaving the remaining variables free. As a separate branch we can fix the first variable at the value 0 and the second variable at the value 1, with the remaining variables being free. Following this process, the third variable can be fixed at 1, and the first two at 0, and so on, until we have a descendant node that fixes the last variable at 1 and all other variables in T at 0. Clearly, the set of nodes described above contains all possible assignments with $T = 0$, and every such assignment is implicitly defined by exactly one of the nodes in the set.

Consider the example of $T = \overline{y_1}\,\overline{z_2}\,\overline{z_3}\,\overline{y_4}$. Branching on $T = 1$ fixes $y_1 = z_2 = z_3 = y_4 = 0$ simultaneously and reduces the number of variables in the problem by 4. Fixing $T = 0$ can be done in four different ways: fixing each variable in T at the value 1 separately. If we simply fix each variable separately at the value 1, we have a symmetry effect that can clearly create an overhead in the performance of the algorithm. For example, as we fix $y_1 = 1$, and in a separate branch fix $z_2 = 1$, we are potentially considering a number of identical subproblems in subsequent branches: at subsequent rounds of branching we could obtain $(y_1 = 1, z_2 = 1)$ in one part of the search tree and $(z_2 = 1, y_1 = 1)$ in another part. Since the number of possible solutions is exponential, this symmetry effect can result in a large increase in the running time of the algorithm. In order to prevent such a situation we fix an increasingly larger set of variables at each branch by, say, fixing $y_1 = 1$, and in a subsequent branch fixing simultaneously $y_1 = 0$ and $z_2 = 1$. In a third branch we can fix $y_1 = z_2 = 0$ and $z_3 = 1$, and finally fix $y_1 = z_2 = z_3 = 0$ and $y_4 = 1$ in the last branch. Note that this policy does not remove any potential solution from consideration, while preventing the undesirable overhead of evaluating equivalent subproblems multiple times.

4.3.1 Branching Criterion

Let us assume that ϕ has both terms with positive and with negative coefficients. Indeed, if all weights β are positive, the optimal solution of (PB) is simply the empty pattern (i.e., a conjunction containing no literals, covering all points in Ω), since problem (PB) requires the maximization of ϕ. Similarly, if all points have negative weights β, then any conjunction that is not satisfied by any point in Ω is an optimal solution of (PB). For instance, any conjunction of the type $x_j\overline{x_j}$ (corresponding to $y_j = z_j = 1$ in (PB)) is an optimal solution in this case.

A simple criterion for selecting the next clause to branch on is the value of the coefficient of the clauses. The clause T with largest positive coefficient will hopefully provide a large increase in objective function when fixed to 1. Thus, we first branch on the case $T = 1$, postponing the branches with $T = 0$.

Similarly, if in a certain node there are only negative clauses, we choose the most negative clause and evaluate first the branches with $T = 0$, and subsequently the one with $T = 1$. The advantages of such a criterion are that its implementation is straightforward and its complexity is linear on the number clauses at the given node.

Obviously, more sophisticated criteria can be used, which can potentially reduce the depth of the search tree and provide sharper upper bounds. In particular, the selection of the order in which the variables should be fixed in the branches with $T = 0$ can make a difference, since fixing certain variables at the value 1 can potentially cause other clauses to vanish.

In addition to that, one could take into account the total effect of fixing a certain clause to the value 1. Fixing $T = 1$ can simultaneously fix other clauses at the value 1 (those involving only a subset of the literals in T), thereby affecting the total contribution of that fixing to the objective function.

In our computational experiments with randomly generated problems the use of the alternative criterion described above did not result in an overall improvement of the algorithm, as compared to the use of the simple criterion. In some cases, marginally superior solutions were obtained early in the search, but that behavior was not consistent. Moreover, in all cases the running time of the algorithm was largely increased, sometimes to twice as much as the time required with the simple branching criterion. Indeed, it is clear that the complexity of the more sophisticated criterion is significantly higher.

In the remainder of this thesis we utilize solely the simple criterion proposed in the beginning of this section.

4.4 Initialization and Node Selection

In this section, we describe a simple heuristic for finding initial solutions of good quality, and the node selection strategy used throughout the rest of the algorithm.

4.4.1 Generating an Initial Solution

As described above, the bounding principle depends on the objective function value of the best solution found so far. Near-optimal solutions typically allow the pruning of considerable parts of the search tree, while solutions of poor quality render the bounding procedure ineffective [15, 51, 77, 96]. Since at early stages of the search the space of solutions is still very large, finding a solution with a high objective function value at the outset of the search contributes to an aggressive pruning that can reduce the overall running time of the algorithm.

We utilize the greedy branching strategy described in the previous section, along with a depth-first queue discipline in order to produce one or more solutions of good quality at an early stage of the search process. We select the next clause to be branched on, and follow the appropriate branch, postponing the other branches. This policy is applied until a maximum number of K nodes is visited, where K was selected to be either 500 or 1,000 in our computational experiments.

As long as there are unresolved clauses, i.e., clauses that have not yet been fixed, the algorithm branches on a clause whose fixing seems to be the most advantageous one – according to the branching criterion described in the previous section – among all unresolved clauses.

This depth-first strategy for selecting the next node to be examined is extremely fast since it does not require the consideration of which node should be examined next based on its upper bound. The procedure usually produces solutions of good quality after a relatively small number of branching operations.

For problems of small dimension, the procedure described above for generating an initial solution frequently exhausts the search space. For larger problems, the solutions produced in this initial phase of the algorithm are typically among the best solutions found during the remainder of the search.

In Section 4.6 we present the results of a series of computational experiments that support these claims.

4.4.2 Node Selection Strategy

We discuss below how the general node selection strategy used throughout the algorithm differs from the one used during the initialization phase.

At any stage of the search procedure, there is a set of nodes that have not yet been examined and whose corresponding upper bounds are still larger than the objective function value of the best solution found so far. We refer to these nodes as *open nodes.* A careful choice of the next open node to be processed can lead to substantial gains in computing time as solutions of good quality may be found more quickly, helping to further prune the search tree.

We select the best open node to be processed according to a discipline based on the depth-first strategy described for obtaining initial solutions. The main difference consists in the periodic interruption of the search every time a certain number of nodes has been examined. This interruption of the branch-and-bound search has the purpose of reorganizing the memory used by the algorithm. Two main tasks are accomplished during each interruption: (i) open nodes, whose upper bounds have become worse than the objective function value of the best known solution since the time they were created, are discarded, and (ii) the remaining set of open nodes is sorted in decreasing order of their upper bounds, with the intent of giving priority of processing to nodes that are more likely to contain improved solutions. We apply such an interruption every time the number of nodes processed equals a multiple of an integer number, typically chosen between 100 and 2,000.

The strategy described above for selecting the next clause to branch on, and for deciding the order in which to create descendants of a subproblem is used throughout the entire search procedure.

4.5 Bounding

As discussed before, the combined use of an effective bounding procedure with feasible solutions of good quality is a crucial part of the branch-and-bound algorithm. At the same time, a good bounding procedure should be relatively fast. Indeed, the node selection strategy described above relies on the explicit computation of an upper bound for each open node in the search tree.

We utilize as an upper bound on the objective function value at any given node, having an associated function ϕ, the sum of the coefficients of the clauses of ϕ having positive coefficients, plus the independent term of ϕ. This is an optimistic upper bound, since we are expecting that there is a certain assignment of the remaining variables for which all clauses with a negative coefficient will vanish, while all clauses with a positive coefficient will be satisfied.

This rather simplistic upper bounding procedure is extremely inexpensive and turned out to work remarkably well in our computational experiments described in Section 4.6 below.

4.6 Computational Experience

In this section we report the results of a number of computational experiments with our branch-and-bound algorithm. We compare the performance of our algorithm with that of the commercial solver Xpress-MP [43] in terms of running time and quality of solutions produced. We also report the quality of the upper bounds generated by both algorithms. Our experiments were performed on randomly generated instances of problem (PB).

In Chapters 5 and 6 we describe two applications of problem (PB) in which one is interested in obtaining a solution of good quality, rather than requiring an optimal solution. In view of that, we report here the performance of what we call "truncated" branch-and-bound searches, in which explicit limits are imposed on the maximum running time and on the maximum number of branch-and-bound nodes evaluated.

We execute three types of experiments: a truncated search with 1,000 nodes, a truncated search with 2,000 nodes, and a truncated search without a limit on the maximum number of branch-and-bound nodes evaluated. For each type of experiment we established the maximum limit of 30 minutes of running time. It turned out that most of the small instances were solved to optimality by both our algorithm and the Xpress-MP solver during one of the truncated searches. Neither our algorithm nor the Xpress-MP solver were able to solve the larger instances within the truncated searches.

Our algorithm was implemented in C++ using the *MS Visual C++ .NET 1.1* environment, utilizing version 16.10.02 of the Xpress-MP solver in our experiments. The computer utilized to run all experiments reported in this chapter was an *Intel Pentium* 4, 3.4GHz, with 2GB of RAM.

Table 4.1 presents the number of variables and the number of clauses of the set of randomly generated instances used in our experiments. Note that n is the number of variables in the dataset Ω. However, in problem (PB) the total number of binary variables is in fact equal to $2n$, while the total number of clauses equals m, each clause having a coefficient randomly chosen between -1 and 1.

Instance	n	m
p_20_1	20	30
p_20_2	20	30
p_20_3	20	30
p_30_1	30	50
p_30_2	30	50
p_30_3	30	50
p_40_1	40	70
p_40_2	40	70
p_40_3	40	70
p_50_1	50	100
p_50_2	50	100
p_50_3	50	100
p_70_1	70	120
p_70_2	70	120
p_70_3	70	120
p_100_1	100	150
p_100_2	100	150
p_100_3	100	150
p_150_1	150	200
p_150_2	150	200
p_150_3	150	200
p_200_1	200	250
p_200_2	200	250
p_200_3	200	250
p_250_1	250	300
p_250_2	250	300
p_250_3	250	300
p_300_1	300	400
p_300_2	300	400
p_300_3	300	400

Table 4.1: Characteristics of randomly generated instances of problem (PB): n = number of features of dataset Ω, m = number of points in dataset Ω.

Table 4.2 shows the results of a truncated search using a maximum of 1,000 branch-and-bound nodes for each problem. The columns labeled "Xpress-MP" refer to the solution and the running time required by the Xpress-MP solver to perform the truncated search. "BBPBF" stands for our "branch-and-bound algorithm for maximizing a pseudo-Boolean function." Within the limit of 1,000 branch-and-bound nodes, BBPBF found solutions with objective function values that are 37% larger on average than those found by the Xpress-MP solver, while requiring an average of 26% of the running time of the Xpress-MP solver.

	Obj. Function Value		Running Time (s)	
Instance	Xpress-MP	BBPBF	Xpress-MP	BBPBF
p_20_1	3.86*	3.86*	0.4	0.0
p_20_2	3.69*	3.69*	0.3	0.0
p_20_3	4.44*	4.44*	0.4	0.1
p_30_1	5.11*	5.11*	2.0	0.4
p_30_2	6.37*	6.37*	1.6	0.4
p_30_3	9.19*	9.19*	1.3	0.1
p_40_1	6.62	7.50*	4.1	1.0
p_40_2	7.25	7.25*	4.1	1.3
p_40_3	5.11*	5.11*	3.8	1.6
p_50_1	7.92	8.27	6.3	1.1
p_50_2	5.85	6.61	5.8	3.0
p_50_3	6.45	7.46	5.5	2.5
p_70_1	5.69	8.74	9.1	1.3
p_70_2	8.29	8.86	9.6	1.3
p_70_3	9.85	11.37	10.5	2.6
p_100_1	7.33	8.22	14.0	1.7
p_100_2	9.86	9.90	15.1	1.8
p_100_3	11.87	20.40*	14.2	33.1
p_150_1	8.30	11.66	27.9	4.1
p_150_2	10.54	12.71	26.9	4.3
p_150_3	8.96	10.75	29.7	3.8
p_200_1	11.10	12.09	48.6	7.6
p_200_2	10.57	10.97	49.7	7.9
p_200_3	6.70	9.95	45.4	7.5
p_250_1	8.39	14.53	82.6	14.0
p_250_2	7.19	16.56	81.5	14.6
p_250_3	6.91	16.79	85.1	15.2
p_300_1	9.90	17.90	220.2	29.0
p_300_2	7.60	14.49	229.7	27.0
p_300_3	5.43	18.94	229.9	32.7

Table 4.2: Quality of solutions for randomly generated instances of problem (PB) using a maximum of 1,000 branch-and-bound nodes. Objective function values marked with an asterisk (*) were proven to be optimal during the truncated search.

Table 4.3[1] shows the results of a truncated search using a maximum of 2,000 branch-and-bound nodes. BBPBF clearly outperforms Xpress-MP in the vast majority of the instances. Not only it finds solutions that are on average 39% superior to those of Xpress-MP, but the running time required is 35% (on average) of the one required by Xpress-MP.

	Obj. Function Value		Running Time (s)	
Instance	Xpress-MP	BBPBF	Xpress-MP	BBPBF
p_40_1	7.50*	—	6.3	—
p_40_2	7.25*	—	5.9	—
p_40_3	—	—	—	—
p_50_1	7.92	8.27	9.2	4.3
p_50_2	5.88	6.61*	10.3	4.5
p_50_3	6.98	7.46*	10.6	3.3
p_70_1	6.79	8.75	14.8	5.4
p_70_2	8.29	9.03	13.7	6.0
p_70_3	10.15	11.37	16.0	9.2
p_100_1	7.33	8.39	23.1	3.3
p_100_2	10.49	9.90	22.3	3.4
p_100_3	14.66	—	22.6	—
p_150_1	8.30	13.38	46.3	40.1
p_150_2	10.54	12.71	46.8	8.1
p_150_3	9.35	12.65	46.7	16.0
p_200_1	14.06	14.70	84.2	14.5
p_200_2	10.87	10.97	82.4	11.0
p_200_3	10.77	13.11	80.5	11.4
p_250_1	8.39	14.53	132.1	18.2
p_250_2	8.17	16.56	157.0	70.0
p_250_3	11.48	18.31	155.8	39.8
p_300_1	9.90	20.32	356.6	250.0
p_300_2	7.60	14.49	365.4	59.2
p_300_3	5.43	18.94	350.4	92.0

Table 4.3: Quality of solutions for randomly generated instances of problem (PB) using a maximum of 2,000 branch-and-bound nodes. Objective function values marked with an asterisk (*) were proven to be optimal during the truncated search. Instances marked with "—" were solved to optimality using at most 1,000 branch-and-bound nodes.

Table 4.4 presents the final upper bounds obtained by BBPBF and Xpress-MP after the end of the truncated searches of 1,000 and 2,000 nodes. Clearly, the upper bound computed by BBPBF is sharper than that computed by the standard integer

[1]The problems with at most 30 variables are not shown here because each of them was solved to optimality by both algorithms using at most 1,000 branch-and-bound nodes.

programming solver of Xpress-MP.

	After 1,000 nodes		After 2,000 nodes	
Instance	Xpress-MP	BBPBF	Xpress-MP	BBPBF
p_20_1	3.86*	3.86*	—	—
p_20_2	3.69*	3.69*	—	—
p_20_3	4.44*	4.44*	—	—
p_30_1	5.11*	5.11*	—	—
p_30_2	6.37*	6.37*	—	—
p_30_3	9.19*	9.19*	—	—
p_40_1	9.53	7.50*	7.50*	—
p_40_2	9.70	7.25*	7.25*	—
p_40_3	5.11*	5.11*	—	—
p_50_1	16.35	16.96	15.29	14.51
p_50_2	10.61	11.03	9.34	6.61*
p_50_3	11.38	13.54	10.23	7.46*
p_70_1	18.90	18.73	17.85	15.57
p_70_2	17.61	18.05	16.58	15.87
p_70_3	22.79	21.79	21.72	18.38
p_100_1	26.18	24.63	25.47	22.70
p_100_2	26.36	25.32	25.74	24.74
p_100_3	33.39	20.40*	32.60	—
p_150_1	34.70	31.11	34.22	30.73
p_150_2	33.79	30.75	33.10	29.51
p_150_3	36.92	31.90	36.22	31.42
p_200_1	51.55	44.85	50.92	44.20
p_200_2	43.85	36.62	43.19	36.59
p_200_3	47.96	40.46	47.00	40.08
p_250_1	55.12	45.80	54.69	45.48
p_250_2	56.17	47.10	55.72	45.90
p_250_3	59.37	50.39	58.89	50.08
p_300_1	75.61	61.76	75.13	60.38
p_300_2	73.35	62.17	72.89	60.46
p_300_3	71.92	59.97	71.22	57.92

Table 4.4: Upper bound on the value of the optimal solution of randomly generated instances of problem (PB). Instances marked with an asterisk were solved to optimality.

Tables 4.5 and 4.6 presents the results of a truncated search in which no limit is imposed on the number of branch-and-bound nodes, but a limit of 30 minutes is enforced on the running time of each algorithm. We report the objective function value of the best solution found during the search by both the BBPBF and Xpress-MP algorithms, along with the total running time and the number of nodes examined during the search.

4.7 Conclusions

It is easy to see that the BBPBF algorithm outperforms the Xpress-MP solver in each of the criteria evaluated in this study. While this might come as a surprise given that Xpress-MP is a well-established integer linear programming solver, we believe that it simply reflects our use of a priori information about the problem's structure.

In particular, it is natural that our bounding procedure proved more effective than the one provided by the continuous relaxation of the integer linear formulation of the problem. Moreover, our algorithm works on the original space of problem (PB), while the use of the Xpress-MP solver requires the rewriting of the original problem as an integer linear program, with as many additional binary decision variables as the number of clauses, and several additional constraints.

It is possible that a proper fine tuning of the cutting-plane regime and preprocessing techniques of the Xpress-MP solver could prove useful in speeding up the solution of (PB) instances. However, the consideration of a fine tuning of the parameters of both algorithms would be part of a larger study and is beyond of the scope of this chapter, especially due to the intended use of the algorithm for producing solutions of reasonable quality at low computational cost, rather than proving optimality.

Although we have focused our description of the algorithm and our computational experiments on problems of the form (PB), it is clear that the BBPBF algorithm can be applied to arbitrary unconstrained pseudo-Boolean optimization problems, without any restriction on the degree of the terms. It is likely that for problems with small degree terms, such as quadratic unconstrained binary optimization (QUBO) [24, 25, 26], the performance advantage of our algorithm will be diminished, as compared to algorithms such as the Xpress-MP solver or a standard branch-and-bound algorithm that branches on individual variables. For problems having a relatively large average degree of terms, it is more likely that the BBPBF algorithm will still retain a significant advantage over the above mentioned algorithms.

Finally, another desirable feature of our algorithm is its trivial portability to different computer platforms, since it is a standard stand-alone *C++* implementation that does not rely on the use of any third-party solver or library.

	Objective Function Value		Running Time (s)	
Instance	Xpress-MP	BBPBF	Xpress-MP	BBPBF
p_40_1	7.50*	7.50*	6.0	1.0
p_40_2	7.25*	7.25*	5.8	1.3
p_40_3	5.11*	5.11*	3.7	1.6
p_50_1	8.27*	8.27*	55.9	9.0
p_50_2	6.61*	6.61*	19.8	4.5
p_50_3	7.46*	7.46*	19.2	3.3
p_70_1	8.75*	8.75*	237.3	26.6
p_70_2	9.03*	9.03*	189.7	23.8
p_70_3	11.37*	11.37*	190.3	18.2
p_100_1	11.00*	11.00*	1,281.4	317.9
p_100_2	11.11*	11.11*	1,203.1	236.2
p_100_3	20.40*	20.40*	572.4	33.2
p_150_1	11.88	14.66*	1,825.3	1,477.8
p_150_2	12.78	12.71	1,826.4	1,800.6
p_150_3	11.61	13.65	1,823.3	1,801.7
p_200_1	18.97	20.54	1,823.2	1,801.3
p_200_2	14.67	17.62	1,827.1	1,800.5
p_200_3	15.83	14.42	1,824.9	1,800.5
p_250_1	14.22	18.60	1,820.8	1,800.2
p_250_2	14.53	18.83	1,826.5	1,801.0
p_250_3	15.16	19.67	1,825.9	1,800.5
p_300_1	13.62	20.32	1,820.8	1,800.1
p_300_2	12.65	19.93	1,824.5	1,800.3
p_300_3	11.73	18.94	1,832.5	1,800.1

Table 4.5: Results of the truncated search with a limit of 30 minutes on the running time and no limit on the number of branch-and-bound nodes. Instances marked with an asterisk were solved to optimality.

Instance	Number of nodes		Upper Bound	
	Xpress-MP	BBPBF	Xpress-MP	BBPBF
p_40_1	1,668	369	7.50	7.50
p_40_2	1,570	603	7.25	7.25
p_40_3	942	885	5.11	5.11
p_50_1	12,864	3,098	8.27	8.27
p_50_2	3,995	1,237	6.61	6.61
p_50_3	3,825	1,242	7.46	7.46
p_70_1	32,587	4,574	8.75	8.75
p_70_2	22,031	4,112	9.03	9.03
p_70_3	35,049	2,633	11.37	11.37
p_100_1	145,547	22,904	11.00	11.00
p_100_2	139,567	15,557	11.11	11.11
p_100_3	71,388	970	20.40	20.40
p_150_1	137,138	22,270	25.87	14.66
p_150_2	121,295	30,059	24.78	23.72
p_150_3	138,396	28,812	27.91	26.69
p_200_1	79,278	12,905	44.33	39.66
p_200_2	77,706	11,547	36.43	34.02
p_200_3	79,233	16,591	40.42	37.05
p_250_1	50,739	10,439	50.00	42.97
p_250_2	53,361	7,477	50.15	44.63
p_250_3	44,891	8,358	54.18	46.45
p_300_1	20,118	5,635	71.94	59.38
p_300_2	18,470	7,426	70.12	58.03
p_300_3	19,004	7,028	68.07	56.95

Table 4.6: Results of the truncated search with a limit of 30 minutes on the running time and no limit on the number of branch-and-bound nodes.

Chapter 5

Large Margin LAD Classifiers

5.1 Introduction

In this chapter we present an optimization approach for model construction in LAD that unifies the distinct tasks of pattern generation and creation of a discriminant function.

Consider a dataset $\Omega = \Omega^+ \cup \Omega^-$, with $\Omega^+ \cap \Omega^- = \emptyset$. In this chapter we assume for the sake of simplicity that Ω is a dataset with n binary attributes. The ideas described here, however, can be applied to datasets with numerical attributes by the use of a discretization procedure, such as the one described in [27] and discussed in Chapter 2.

We recall that a LAD model consists of a set M of patterns, so that every observation in Ω^+ satisfies at least one of the positive patterns in M, and every observation in Ω^- satisfies at least one of the negative patterns in M. As discussed in Chapters 2 and 3, the construction of a LAD model is a computationally expensive task that typically involves the enumeration of a large set of patterns satisfying a certain property, followed by the selection of a relatively small subset of those.

The fact that only a fraction of the patterns generated is ultimately used for the construction of a LAD model suggests that the approach described above for model construction may not be the most efficient one. Moreover, the choice of enumerating a specific family of patterns, and the usual selection of a small number of those patterns – which is frequently guided by a greedy criterion – may not be the most appropriate for certain problems, resulting in sub-optimal LAD models. In view of this, it is appropriate to say that there is a clear need for a global criterion for choosing an overall "best" LAD model.

For a given set of patterns, the construction of a discriminant function for classification is usually done in one of the ways discussed in Chapter 2: by majority vote among positive and negative patterns, or by solving a linear program that provides a set of weights that result in an optimal separation of the two classes [28]. In this chapter, we propose an algorithm based on the linear programming formulation of [28] that finds an optimal discriminant function defined over the set of all patterns.

Since the total number of patterns can be extremely large, even for datasets of modest dimensions, the direct solution of the LP formulation of [28] over the set of all patterns is of no practical use. We propose the use of the classical column generation technique [58, 59, 85, 95, 129] for iteratively generating patterns as needed during the search for an optimal discriminant function. We show how a pseudo-Boolean optimization subproblem can be formulated which returns the most suitable pattern for improving the discriminant function at any given iteration. The algorithm starts from an arbitrary LAD model and proceeds until no pattern can be found that improves the current discriminant function, or until only marginal improvements of the discriminant function can be obtained. We discuss some computational issues about the implementation of the algorithm and report numerical results obtained on publicly available datasets from the UCI Machine Learning Repository [97].

5.2 Constructing an Optimal Discriminant Function

Several pattern generation algorithms [4, 11, 20] and selection criteria for constructing LAD models [8, 66] have been proposed. However, none of these approaches for pattern generation or model construction proved to be consistently superior to the others on the datasets used for experimentation. Indeed, practice has shown that fine tuning the control parameters that govern the construction of LAD models, including the choice of pattern generation and model construction algorithms, is a time-consuming task. It is therefore important to develop an algorithm that selects a set of patterns and a set of weights to build an optimal LAD model according to a certain performance criterion that relates to the robustness of classification of unseen observations. A natural such criterion would the *separation margin* of the classifier [118, 120]. In the context of

LAD, the separation margin is simply the difference between the smallest value that the discriminant function takes over the positive points of the training set that are correctly classified and the largest value that the discriminant function takes over the negative points in the training set that are correctly classified. We recall here that the LAD discriminant function corresponding to sets of positive and negative patterns $\mathcal{P}$ and $\mathcal{N}$, and associated vectors of weights α and β (respectively) is given by

$$\Delta(\omega) = \sum_{P_i \in \mathcal{P}} \alpha_i P_i(\omega) - \sum_{N_i \in \mathcal{N}} \beta_i N_i(\omega).$$

Thus, the separation margin corresponding to a given discriminant function is

$$\min\{\Delta(\omega) : \omega \in \Omega^+, \Delta(\omega) > 0\} - \max\{\Delta(\omega) : \omega \in \Omega^-, \Delta(\omega) < 0\}.$$

By maximizing the separation margin, we expect a robust classification of unseen examples. This expectation has been confirmed in practice in many situations [30, 81, 114]. The good performance of boosting and voting-based classifiers has also been shown to be related to the implicit maximization of the separation margin of the resulting classifier [114].

5.2.1 Maximizing the Separation Margin

In this subsection we show how to formulate the problem of constructing an optimal discriminant function by using a linear programming formulation similar to the one described in Chapter 2.

Let $\mathbb{P}$ be the set of all positive patterns having homogeneity at least ρ^+, and let $\mathbb{N}$ be the set of all negative patterns having homogeneity at least ρ^-. Let $\mathbb{P} = \{P_1, \ldots, P_{|\mathbb{P}|}\}$, $\mathbb{N} = \{N_1, \ldots, N_{|\mathbb{N}|}\}$, and for every $\omega \in \Omega$ let $P_i(\omega) = 1$, if observation ω is covered by pattern P_i, and 0 otherwise. Similarly, let $N_j(\omega) = 1$ if ω is covered by N_j, and 0 otherwise. We are interested in solving the following problem:

$$
\begin{aligned}
\text{(P)} \quad \text{maximize} \quad & r + s \; - \; C\sum_{\omega\in\Omega}\epsilon_\omega \\
\text{subject to:} \quad & \\
& \sum_{i=1}^{|\mathbb{P}|}\alpha_i P_i(\omega) - \sum_{j=1}^{|\mathbb{N}|}\beta_j N_j(\omega) + \epsilon_\omega \geq r, \;\; \forall\, \omega \in \Omega^+ && (5.1)\\
& \sum_{i=1}^{|\mathbb{P}|}\alpha_i P_i(\omega) - \sum_{j=1}^{|\mathbb{N}|}\beta_j N_j(\omega) - \epsilon_\omega \leq -s, \;\; \forall\, \omega \in \Omega^- && (5.2)\\
& \sum_{i=1}^{|\mathbb{P}|}\alpha_i = 1 \\
& \sum_{j=1}^{|\mathbb{N}|}\beta_j = 1 \\
& r \geq 0, s \leq 0 \\
& \alpha_i \geq 0, i = 1, \ldots, |\mathbb{P}| \\
& \beta_j \geq 0, j = 1, \ldots, |\mathbb{N}| \\
& \epsilon_\omega \geq 0, \; \forall\, \omega \in \Omega,
\end{aligned}
$$

where the values of α_i ($i = 1, \ldots, |\mathbb{P}|$) and β_j ($j = 1, \ldots, |\mathbb{N}|$) are the weights of the positive and negatives patterns, respectively, r represents the positive part of the separation margin, s represents the negative part of the separation margin, and C is a nonnegative penalization parameter that controls how much importance is given to the violations ϵ_ω of the separating constraints (5.1) and (5.2).

Due to the potentially very large total number of patterns, it is highly unlikely that in real-world applications one will encounter a dataset where such a formulation can be directly tackled. Thus, let us consider a subset of positive patterns $\mathcal{P}^0 \subset \mathbb{P}$ and a subset of negative patterns $\mathcal{N}^0 \subset \mathbb{N}$, and let us denote by $I^0 \subset \{1, \ldots, |\mathbb{P}|\}$ and $J^0 \subset \{1, \ldots, |\mathbb{N}|\}$ the sets of indices corresponding to the elements of $\mathcal{P}^0$ and $\mathcal{N}^0$, respectively. Then, we can write a restricted version of (P) as

$$\text{(RP)} \quad \text{maximize} \quad r + s \; - \; C\sum_{\omega\in\Omega}\epsilon_\omega$$

subject to:

$$r - \sum_{i\in I^0}\alpha_i P_i(\omega) + \sum_{j\in J^0}\beta_j N_j(\omega) - \epsilon_\omega \le 0, \;\; \forall\, \omega \in \Omega^+ \tag{5.3}$$

$$s + \sum_{i\in I^0}\alpha_i P_i(\omega) - \sum_{j\in J^0}\beta_j N_j(\omega) - \epsilon_\omega \le 0, \;\; \forall\, \omega \in \Omega^- \tag{5.4}$$

$$\sum_{i\in I^0}\alpha_i = 1 \tag{5.5}$$

$$\sum_{j\in J^0}\beta_j = 1 \tag{5.6}$$

$$r \ge 0, s \le 0$$

$$\alpha_i \ge 0, \; \forall\, i \in I^0$$

$$\beta_j \ge 0, \; \forall\, j \in J^0$$

$$\epsilon_\omega \ge 0, \; \forall\, \omega \in \Omega.$$

Note that we rearranged constraints (5.3) and (5.4) in order to write (RP) in a standard maximization form. It is easy to choose $\mathcal{P}^0$ and $\mathcal{N}^0$ in such a way that (RP) has a feasible solution. We recall that the *minterm* corresponding to a given observation ω is simply the pattern with the largest possible number of literals that is satisfied by observation ω, i.e., it is the pattern given by the following conjunction: $T = \prod_{i:\omega_i=1} x_i \prod_{j:\omega_j=0} \overline{x}_j$.

Clearly, T is a pattern that covers ω and does not cover any observation from the opposite part of the dataset (since $\Omega^+ \cap \Omega^- = \emptyset$). Let us choose $\mathcal{P}^0$ and $\mathcal{N}^0$ to be the sets of minterms corresponding to the positive and negative observations in the dataset, respectively. Then, the vectors α and β given by $\alpha_i = \frac{1}{|\mathcal{P}^0|}$, $i \in \mathcal{P}^0$ and $\beta_j = \frac{1}{|\mathcal{N}^0|}$, $j \in \mathcal{N}^0$, along with $r = \frac{1}{|\mathcal{P}^0|}$, and $s = \frac{1}{|\mathcal{N}^0|}$ constitute a feasible solution to (RP). In fact, constructing a feasible set of patterns to initialize (RP) is not a difficult task; one pattern of each class would suffice. However, finding a good set of patterns to initialize the column generation algorithm is always advantageous as such patterns are likely to be part of the optimal set of patterns and their use in early iterations may significantly speed up the entire procedure. A reasonable alternative to construct an initial set of good patterns is to use one of the pattern generation algorithms described in Chapter 3.

5.2.2 The Pricing Subproblem

In this subsection we address the subproblem phase in the column generation algorithm. We formulate this problem as a pseudo-Boolean optimization problem that can also be seen as a weighted version of the *maximum pattern problem* [20, 51] described in Chapter 3.

The solution of problem (RP) provides an optimal discriminant function for the set of patterns $\mathcal{P}^0 \cup \mathcal{N}^0$. However, that discriminant function may not be optimal with respect to the entire set of all patterns. In order to verify global optimality, we need to make sure that there is no pattern that, once added to the current set of patterns, allows for an improvement in the value of the objective function of (RP).

In the simplex method, one of the commonly used heuristics for choosing a non-basic variable to enter the current basis is to choose the variable with the best (in the maximization case, the largest) reduced cost [85, 95, 38, 129]. This criterion is usually referred to as *Dantzig's rule* or *steepest descent rule* [38, 95]. Similarly, we use the reduced cost of a specific pattern P to measure the potential improvement in the objective function of (RP) resulting from the introduction of P into the set of known patterns $\mathcal{P}_0 \cup \mathcal{N}_0$. To verify the optimality of the current solution of (RP), we solve a pricing problem that will return a pattern with the best reduced cost with respect to the current values of the dual variables of (RP). Because (RP) is a maximization problem, we are interested in finding a pattern with the largest reduced cost.

According to the usual optimality criterion of the primal simplex method, if there is no pattern (not in $\mathcal{P}^0 \cup \mathcal{N}^0$) with a nonnegative reduced cost, then the current solution of (RP) is optimal to (P); otherwise, a new pattern can be added to the set $\mathcal{P}^0 \cup \mathcal{N}^0$ and the algorithm proceeds. Notice here that we allow the introduction of a pattern whose reduced cost is zero. Since the column generation algorithm is equivalent to applying the simplex algorithm to a large formulation, there is a possibility of having degenerate iterations in which no progress is made in terms of the objective function, exactly as in the case of the simplex method [38, 95, 129]. We will refer to a pattern with nonnegative reduced cost at a given iteration as a *candidate pattern*.

Note that since $r + s$ corresponds to the separation margin of the current discriminant function, solving the pricing problem corresponds to asking if the current separation margin can be enlarged by the addition of a pattern. Thus, the column generation algorithm applied to problem (P) attempts to increase the separation margin by including patterns that make the distinction between the sets Ω^+ and Ω^- more pronounced.

From now on, let us assume that k iterations of the column generation algorithm have been executed. We will refer to the corresponding sets of patterns as $\mathcal{P}^k$ and $\mathcal{N}^k$, and to the corresponding version of problem (RP) as (RP^k). Let λ and μ be the vectors of dual variables associated to constraints (5.3) and (5.4), respectively, at an optimal solution of (RP^k). Similarly, let θ^+ and θ^- be the dual variables corresponding to constraints (5.5) and (5.6). The reduced costs corresponding to a positive pattern P_i and a negative pattern N_j are given by $\theta^+ + [\lambda^\top, -\mu^\top]\pi_i$ and $\theta^- + [-\lambda^\top, \mu^\top]\nu_j$ [38, 129, 85], respectively, where $\pi_i = [P_i(\omega)]_{\omega\in\Omega}$ and $\nu_j = [N_j(\omega)]_{\omega\in\Omega}$ are the characteristic vectors of the observations satisfying P_i and N_j, respectively.

Let us define binary decision variables x_i and x_i^c ($i = 1, \dots, n$) associated to attribute a_i in the following way: (i) $x_i = 1$ and $x_i^c = 0$ if the literal associated to $a_i = 1$ is used in the resulting pattern; (ii) $x_i = 0$ and $x_i^c = 1$ if the literal associated to $a_i = 0$ is present in the resulting pattern. Thus, the positive pricing subproblem associated to optimal vectors of dual variables λ^* and μ^* of (RP) can be formulated as the following pseudo-Boolean optimization problem:

$$(\mathrm{SP}^+) \qquad \text{maximize} \quad \sum_{\omega\in\Omega^+} \lambda^*_\omega \left(\prod_{i:\omega_i=0} \overline{x_i} \prod_{j:\omega_j=1} \overline{x_j^c} \right) - \sum_{\omega\in\Omega^-} \mu^*_\omega \left(\prod_{i:\omega_i=0} \overline{x_i} \prod_{j:\omega_j=1} \overline{x_j^c} \right)$$

subject to:

$$x_j, x_j^c \in \{0,1\}, \ j = 1, \dots, n,$$

where each term $\prod_{i:\omega_i=0} \overline{x_i} \prod_{j:\omega_j=1} \overline{x_j^c}$ takes the value 1 whenever ω is covered by the resulting conjunction, and 0 otherwise. In addition to the formulation of problem (S^k), we need to enforce that the resulting conjunction is a positive pattern.

Clearly, problem (SP$^+$) is an instance of the family of pseudo-Boolean optimization problems whose solution was addressed in Chapter 4. In fact, we utilize a simple modification of our branch-and-bound algorithm BBPBF to solve (SP$^+$). The only difference between the algorithm described in Chapter 4 and the one used in this chapter to solve (SP$^+$) is that whenever we consider a candidate solution for problem (SP$^+$) we first check whether it satisfies the definition of a pattern, according to the homogeneity level required. If the solution satisfies the definition, it is considered as a feasible solution, otherwise it is not taken into account and the algorithm proceeds normally. This procedure is equivalent to the introduction of a large negative penalty for violating the definition of a pattern.

Dual variable θ^+ appears in the formulation of (SP$^+$) because the problem is formulated with the goal of constructing a positive pattern, and therefore it is implicit that the column corresponding to this pattern has a coefficient equal to 1 in the row corresponding to constraint (5.5). Thus, the value of θ^+, despite being a constant, is part of the formulation of (SP$^+$).

While (SP$^+$) is aimed at finding the best positive candidate pattern at the current iteration, it may be necessary to search for a negative candidate pattern as well. In fact, even if we have already found a positive candidate pattern, it may be advantageous to generate a negative candidate pattern, if there is any. We suggest to solve both (SP$^+$) and its negative counterpart, (SP$^-$), at every iteration. For the negative pricing problem (SP$^-$), we simply minimize the same objective function as in problem (SP^k) — as opposed to maximizing in (SP$^+$) — and require that the resulting conjunction satisfies the condition of being a negative pattern. Here, as in the case of (SP$^+$), a slight modification of algorithm BBPBF can be used to solve (SP$^-$).

The solutions of (SP$^+$) and (SP$^-$) provide either: (a) a certificate of optimality of the current discriminant function, in case the objective function value of (SP$^+$) is strictly positive and that of (SP$^-$) is strictly negative; or (b) a new positive or negative pattern (or both) to be added to our current set of known patterns. In general, if at iteration k we generate new patterns $P^* \in \mathbb{P}$ and $N^* \in \mathbb{N}$, then we define the sets of patterns for the next iteration as $\mathcal{P}^{k+1} \leftarrow \mathcal{P}^k \cup \{P^*\}$ and $\mathcal{N}^{k+1} \leftarrow \mathcal{N}^k \cup \{N^*\}$.

Let (x^*, x^{c*}) be an optimal solution of (SP^+). In practice, by adding a new positive pattern to the current set of patterns we are actually adding the following column to problem (RP^k):

$$\begin{bmatrix} \left(-\prod_{i:\omega_i=0} \overline{x_i^*} \prod_{j:\omega_j=1} \overline{x_j^{c*}}\right)_{\omega\in\Omega^+} \\ \left(\prod_{i:\omega_i=0} \overline{x_i^*} \prod_{j:\omega_j=1} \overline{x_j^{c*}}\right)_{\omega\in\Omega^-} \\ 1 \\ 0 \end{bmatrix}.$$

5.2.3 The Column Generation Algorithm

The column generation algorithm as described above alternates the solution of problem (RP) with the solution of (SP^+) and (SP^-). While problem (RP) optimizes the discriminant function over the current set of patterns, problems (SP) verify the optimality of the discriminant function obtained by (RP) and, if necessary, produce new patterns to improve the current discriminant function. Thus, the iterative process stops at one of the following events: (i) the maximum number of iterations is reached; (ii) no candidate pattern can be found; (iii) no significant improvement of the objective function of (RP) takes place after a large number of iterations. For the latter case, we need to define two control parameters: a tolerance value corresponding to the smallest increase in the objective function of (RP) to be regarded as a significant increase, and the maximum number of iterations to be allowed without a significant increase. These control parameters are important, since in practice it is possible to have long sequences of iterations in which the objective function of (RP) remains unchanged or changes only slightly.

Although column generation algorithms usually approach the neighborhood of a near-optimal solution in a reasonably small number of iterations [89, 95], achieving or proving optimality can take a very large number of iterations. Typically, towards the end of the algorithm's execution, the graph corresponding to the progress in the objective function value exhibits a long "tail", reflecting the failure in making significant progress during a large number of iterations. This phenomenon is known as the *tailing-off effect*

[59, 85, 89, 95, 127]. Aside from numerical instability due to the finite precision used in the implementation of such algorithms, the tailing-off phenomenon can be explained by the fact that the solutions produced by (RP) may be interior points to the original problem (P). Therefore, as the algorithm approaches an optimal solution, a significant amount of "cleaning up" may be necessary to arrive at a vertex of the original polyhedron [95]. This may require the generation of several columns in order to make small adjustments in the values of some variables of (RP). In our experiments, we used the tolerance parameter equal to 10^{-4} and the maximum number of degenerate iterations equal to 10. The maximum number of iterations allowed for the entire execution of the algorithm was set to $1,000$, but was never reached in our experiments.

Figure 5.1 summarizes the algorithm. We use the notation (RP(k)), (SP(k)$^+$) and (SP(k)$^-$) to denote the optimization problems solved at iteration k.

```
1.  Input: $\Omega^+, \Omega^-$, initial sets $\mathcal{P}^0$ and $\mathcal{N}^0$ of patterns. Set $k = 0$.
2.  Solve (RP($k$)), with optimal primal and dual solutions $(r^*, s^*, \alpha^*, \beta^*)$ and
    $(\lambda^*, \mu^*, \theta^{+*}, \theta^{-*})$.
3.  Solve (SP($k$)$^+$) and (SP($k$)$^-$), with optimal positive and negative
    patterns $P^*$ and $N^*$.
4.  If at least one of the stopping criteria is met, stop.
5.  If $P^*$ or $N^*$ (or both) are candidate patterns, define $\mathcal{P}^{k+1} = \mathcal{P}^k \cup \{P^*\}$ and
    $\mathcal{N}^{k+1} = \mathcal{N}^k \cup \{N^*\}$ accordingly, and set $k = k + 1$.
6.  Go to Step 2.
```

Figure 5.1: Pseudo-code of the column generation algorithm.

As an alternative, our algorithm can be implemented with the utilization of the original BBPBF algorithm for the solution of problems (SP$^+$) and (SP$^-$). By not imposing specific limits on the coverage of the conjunctions generated by BBPBF (i.e., by not requiring that the conjunctions are patterns with prescribed minimum prevalence and homogeneity), we allow the algorithm to self-adjust to the given training set. The algorithm will generate the types of patterns required for a robust classification, without the need for an initial estimation of how much homogeneity or prevalence should be required for the given training data. Such a modification reduces even more the complexity of the LAD training procedure, as it removes the need to fine tune the homogeneity and prevalence parameters. In fact, this is the version of the algorithm actually utilized in the computational experiments reported in the next section.

5.2.4 Overfitting

An important remark with respect to the termination criterion of our algorithm is that it may force termination at a sub-optimal solution. In many applications of optimization algorithms, reaching optimality is not a real concern. In our current context, this is also true. The large margin criterion is simply a heuristic objective function that we use for guiding our search for a good classifier. In machine learning, finding an optimal solution frequently corresponds to selecting from a certain family of functions one that fits best a given dataset. This can clearly lead to *overfitting* [92, 44, 73] the given data. Indeed, as suggested in [44], by solving the optimization problem (P) we are trying to find the best possible separation of the training data, while the actual objective function should be to provide a good separation for unseen observations.

The early termination of our iterative algorithm due to the tailing-off phenomenon can be seen as a simple procedure for avoiding overfitting the training data. Another simple stopping criterion used in our experiments refers to the patterns obtained by solving problems (SP^+) and (SP^-). If the coverage of the patterns obtained during a number of consecutive iterations has always been less than a certain percentage of the training set, we stop the algorithm. The rationale behind this test is that the inclusion of patterns of low coverage suggests that, at the current iteration, no significant global trend in the training data can be found in order to improve the separation margin. To avoid making small adjustments of fit local characteristics of the training sample, we stop the algorithm and report the current solution as the best one found.

Other usual overfitting-preventing mechanisms involve imposing an explicit limit on, or penalizing the complexity of the classifier built [44, 73, 92]. As suggested above, this mostly accounts for preventing the construction of a discriminant function based on a large collection of patterns, many of which cover small subsets of the training data. Our algorithm allows a certain level of extra control over such issues via the setting of parameter C in problem (RP), by means of which we can define how much emphasis we want to give to an accurate separation of the training data.

5.3 Computational Experiments

The algorithm proposed in this chapter was implemented in C++, using the *MS Visual C++ .NET 1.1* compiler. The linear programming formulations solved as part of the column generation algorithm were solved using the callable library of the Xpress-MP optimizer [43]. The computer used for carrying out the experiments reported here was an *Intel Pentium* 4, 3.4GHz, with 2GB of RAM.

Problems (SP^+) and (SP^-) were not necessarily solved to optimality in our experiments. Given the satisfactory performance of the truncated versions of the BBPBF algorithm, as reported in Chapter 4, we established explicit limits on the execution of the variant of the BBPBF algorithm utilized in this chapter. The maximum number of branch-and-bound nodes explored in the initialization procedure was set to 1,000, while the number of additional nodes explored during the remainder of the search was set to 2,000. The maximum time limit of 30 minutes was also imposed, but never reached in our experiments.

In Figure 5.2, we show six histograms illustrating a typical example of how the frequencies of values of the discriminant function evolve during the execution of our algorithm. The graphs correspond to the application of our training algorithm to a randomly extracted sample of 90% of the observations in the "heart" dataset, from the UCI Repositrory [97]. The empty region in the middle of the graphs is the separation margin between positive and negative points. The graphs shows how our algorithm iteratively improves the separation margin as it finds candidate patterns and adjusts the weights of the discriminant function accordingly.

Although all observations are correctly separated in each of the iterations, we can see that a large number of observations are on the border of the separating region in early iterations, i.e., their corresponding constraints from sets (5.3) and (5.4) are binding. The observations on the border of the margin of separation can be seen as "support observations", just as in the case of support vector machines (see, e.g., [115, 120]).

Clearly, the ideal situation would be to have just a few observations on the border, and a larger number of observations as we move towards the extreme regions of the graph. Discriminant functions with such a distribution can be expected to perform better on unseen examples than the ones shown in Figure 5.2(a) and 5.2(b), for instance. In subsequent iterations (see Figures 5.2(c) and 5.2(d)) we can notice a better distribution of the observations in terms of discriminant value, with substantially less observations on the border of the separating region. Finally, in Figures 5.2(e) and 5.2(f) we see that while trying to increase margin of separation, the algorithm adjusts the discriminant function causing a simultaneous increase in the number of support observations. There is here a natural trade-off between the margin of separation and the number of support observations. The early termination criteria discussed in Section 5.2 cause the automatic interruption of the algorithm at iteration 72, after a sequence of 10 iterations has taken place without a substantial increase in the size of the margin of separation.

Note that here, as in the case of support vector machines, there is a natural interpretation of the dual solution of problem (RP(k)). The values of the dual variables can be used to describe a *dual classifier* consisting of a linear combination of the support observations (described as 0-1 vectors in the pattern-space corresponding to iteration k). The smaller the number of supporting observations, the simpler the dual classifier will be.

In Figure 5.3 we present the values of the positive and negative margins during the execution of the same experiment on the “heart” dataset. It can be seen that roughly after iteration 50 the algorithm stops making significant improvements in terms of increase in the separation margin (the tailing-off phenomenon described in the previous section). That fact seems to be accompanied by an increase in the number of observations at the border of the separation margin, as discussed above (see Figure 5.2). This observation helped us to adjust the parameters for early termination that prevent the tailing-off phenomenon and avoid overfitting the discriminant function to the training sample.

Table 5.1 reports the cross-validated accuracies of five algorithms implemented in the Weka package, CAP-LAD, and those of LAD models built utilizing the algorithm described in this chapter. The table is based on the one reported in Chapter 3, with the addition of the "LM-LAD" column, which refers to the version of LAD based on our algorithm and stands for *Large Margin LAD*. The highest accuracy for each dataset is emphasized with boldface characters. The Borda counts are computed based on ranking the seven algorithms and summarize their comparative performance.

Dataset	SMO	J48	Rand.For.	Mult.Perc.	S. Log.	CAP-LAD	LM-LAD
breast-w	0.965 ± 0.011	0.939 ± 0.012	**0.967 ± 0.009**	0.956 ± 0.012	0.963 ± 0.013	0.966 ± 0.011	0.942 ± 0.024
credit-a	0.864 ± 0.025	0.856 ± 0.031	**0.882 ± 0.027**	0.831 ± 0.032	0.873 ± 0.025	0.867 ± 0.025	0.815 ± 0.044
hepatitis	0.772 ± 0.084	0.652 ± 0.086	0.722 ± 0.101	0.727 ± 0.065	0.764 ± 0.090	**0.779 ± 0.104**	0.738 ± 0.091
krkp	**0.996 ± 0.003**	0.994 ± 0.003	0.992 ± 0.003	0.993 ± 0.002	0.975 ± 0.005	0.993 ± 0.002	0.962 ± 0.031
boston	0.889 ± 0.028	0.837 ± 0.045	0.875 ± 0.024	**0.893 ± 0.031**	0.874 ± 0.021	0.855 ± 0.029	0.840 ± 0.045
bupa	0.701 ± 0.045	0.630 ± 0.041	0.731 ± 0.046	0.643 ± 0.020	0.662 ± 0.048	**0.734 ± 0.052**	0.678 ± 0.034
heart	**0.837 ± 0.039**	0.799 ± 0.052	0.834 ± 0.051	0.815 ± 0.025	0.834 ± 0.040	0.826 ± 0.032	0.814 ± 0.033
pima	0.727 ± 0.029	0.722 ± 0.026	0.736 ± 0.030	0.726 ± 0.023	0.729 ± 0.031	**0.747 ± 0.022**	0.682 ± 0.023
sick	0.824 ± 0.027	**0.926 ± 0.020**	0.832 ± 0.023	0.852 ± 0.049	0.808 ± 0.023	0.825 ± 0.019	0.815 ± 0.041
voting	0.961 ± 0.018	0.960 ± 0.015	0.961 ± 0.016	0.944 ± 0.025	**0.961 ± 0.014**	0.952 ± 0.021	0.945 ± 0.025
Borda count	52	28	52	35	41	50	20

Table 5.1: Classification accuracy and Borda counts of Weka algorithms, CAP-LAD and LM-LAD.

In Table 5.2 we present the counts of wins, losses, and ties for the pairwise comparison of the seven algorithms. Each pairwise comparison in Table 5.2 is made with respect to the 95% confidence interval presented in Table 5.1.

	SMO	J48	Rand.For.	Mult.Perc.	S. Log.	CAP-LAD
J48	4-1-5					
Rand.For.	1-1-8	4-1-5				
Mult.Perc.	0-4-6	2-2-6	0-2-8			
S.Log.	1-1-8	1-2-7	0-2-8	1-2-7		
CAP-LAD	0-1-9	2-1-7	0-0-10	2-1-7	1-0-9	
LM-LAD	0-0-10	0-1-9	0-1-9	0-0-10	0-0-10	0-1-9

Table 5.2: Matrix of wins, losses and ties.

While being superior to some of the other algorithms for some datasets, the performance of the LM-LAD algorithm is, strictly speaking, generally inferior to that of other algorithms on the datasets used in our experiments. The Borda count shown in Table 5.1 summarizes this observation. However, the accuracies obtained by our algorithm are reasonably close to those of the other algorithms (as can be seen from Table 5.2). Indeed, as discussed in [45], the direct comparison of the accuracy of two algorithms in the experimental setup utilized here must be regarded cautiously, while the conclusion that the two accuracies are not statistically different is to be trusted. Moreover, if for each dataset we compare the accuracy of LM-LAD with the highest accuracy of the other six algorithms, we can see that LM-LAD achieves on average 94.3% of the accuracy of the best algorithm.

It is important to mention that the LM-LAD algorithm was not calibrated for any of the ten benchmark datasets. In fact, the C parameter was fixed at 1.0 in all experiments and no constraint has been imposed on the degree, prevalence or homogeneity of the patterns generated.

5.4 Conclusions

In this chapter we have described a novel algorithm for constructing LAD models that unifies the usually distinct tasks of generating a set of patterns and defining a discriminant function. We have also investigated how accurate are the LAD models built with such an algorithm and how they compare with other classification models.

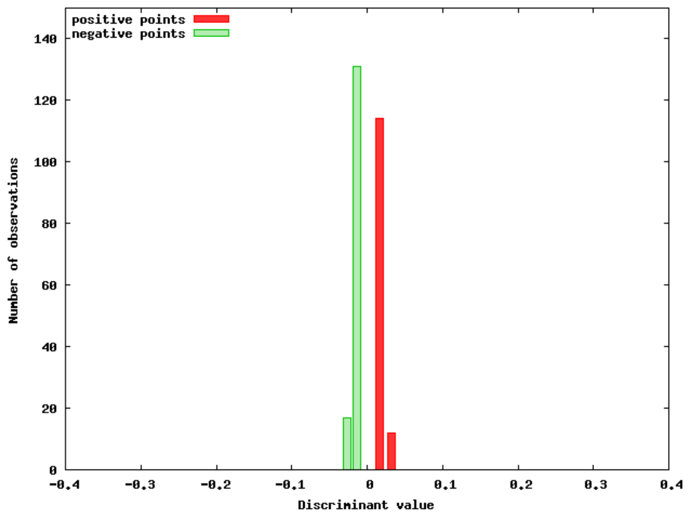
The main novelty in our algorithm is that the underlying optimization model proposed here is more general than other existing methods for model construction in LAD. Other existing approaches for pattern generation have limitations of a certain type: on the types of patterns generated (spanned [4], prime [11], maximum [20]) and on the degrees of the patterns enumerated (except for the case of maximum patterns). Moreover, the discriminant functions are in general of a very simple form, with patterns of the same class having equal weights [20, 28]. In each of the previous LAD implementations, the choices of a model and a discriminant function were limited by one or more such factors, none of which is present in our approach.

Indeed, the pricing problems (SP^+) and (SP^-) can produce patterns of any type, including patterns of arbitrary degree. Moreover, the iterative procedure presented here allows for the construction of a globally optimal discriminant function.

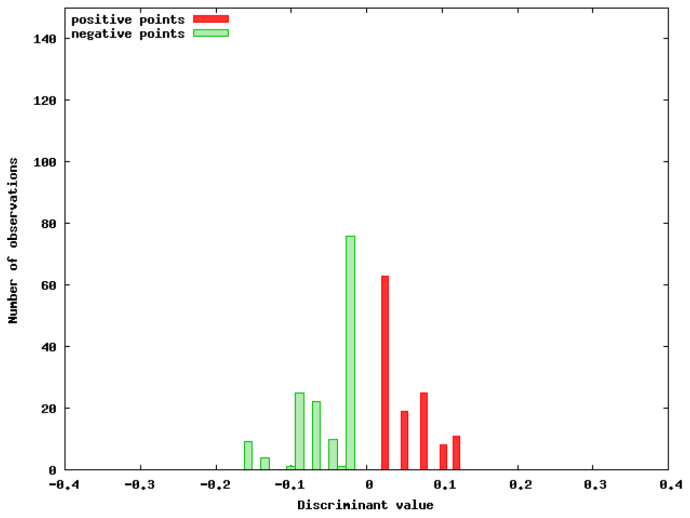
In addition to this conceptual simplification in the construction of a LAD model, our algorithm is practically parameter-free, having only a few technical control parameters primarily related to the underlying optimization process. Our computational experiments suggest that the set of values described here for these parameters is quite stable, and therefore fixing the parameters at these values allows the straightforward use of the algorithm without any need for careful calibration in order to obtain reasonable results. While the LAD models constructed by our algorithm are not optimal in terms of accuracy, they have accuracies which are typically close to those of the most accurate algorithms used in our experiments.

Moreover, the optimization model utilized by our algorithm is quite simple and can be easily modified to account for different objective functions, or to include alternative penalty terms. The similarities between our algorithm and the standard support vector machines algorithm suggest that some improvement could be made by the use of ideas coming from the SVM literature, such as the use a quadratic formulation, or the use of an alternative parameter ν that bounds from above the fraction of misclassified observations in the training dataset.

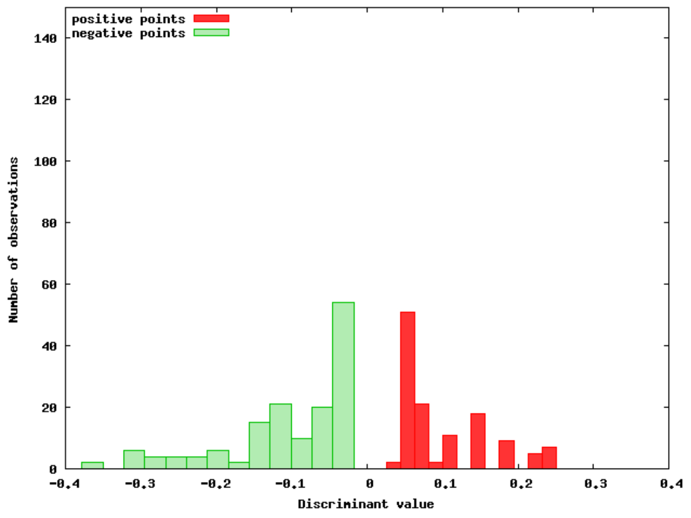
Finally, the LM-LAD algorithm described in this chapter can be seen as a first version of a LAD procedure that completely dispenses with the calibration of parameters such as degree, homogeneity, and prevalence of patterns.



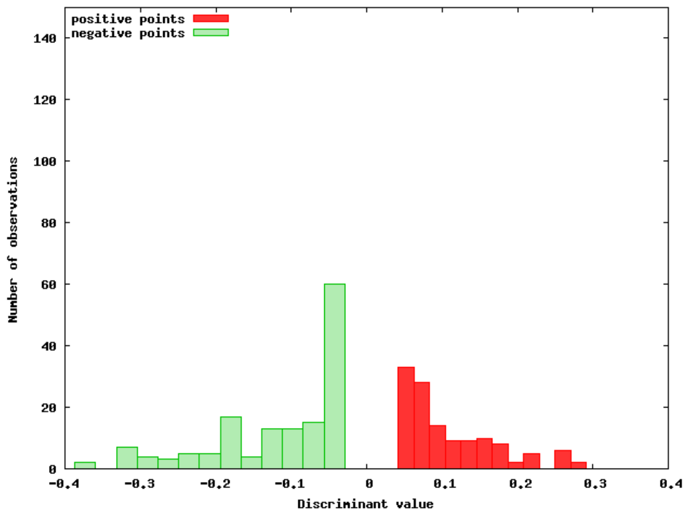
(a) At iteration 2



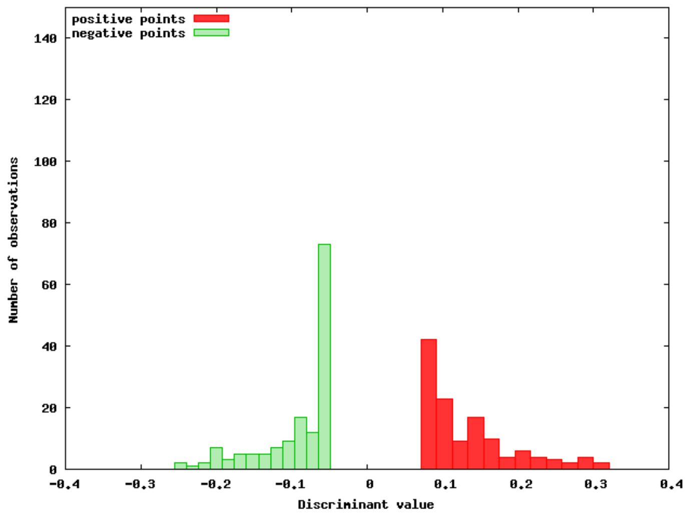
(b) At iteration 5



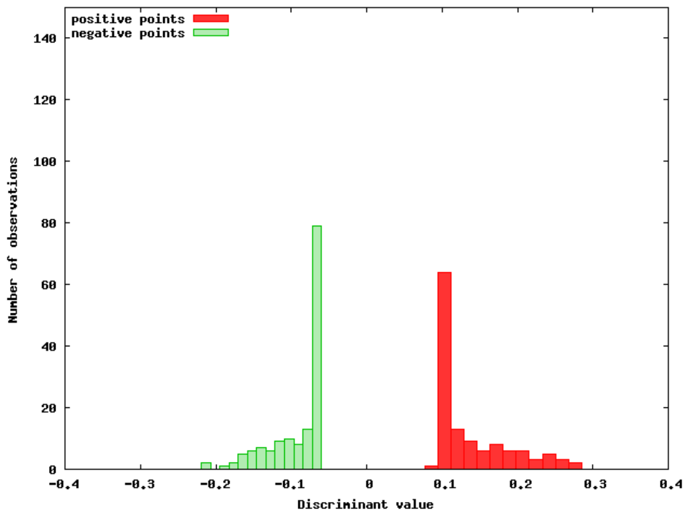
(c) At iteration 10



(d) At iteration 20



(e) At iteration 50



(f) At iteration 72 (last)

Figure 5.2: Histograms of observations at different discriminant levels for the "heart" dataset.

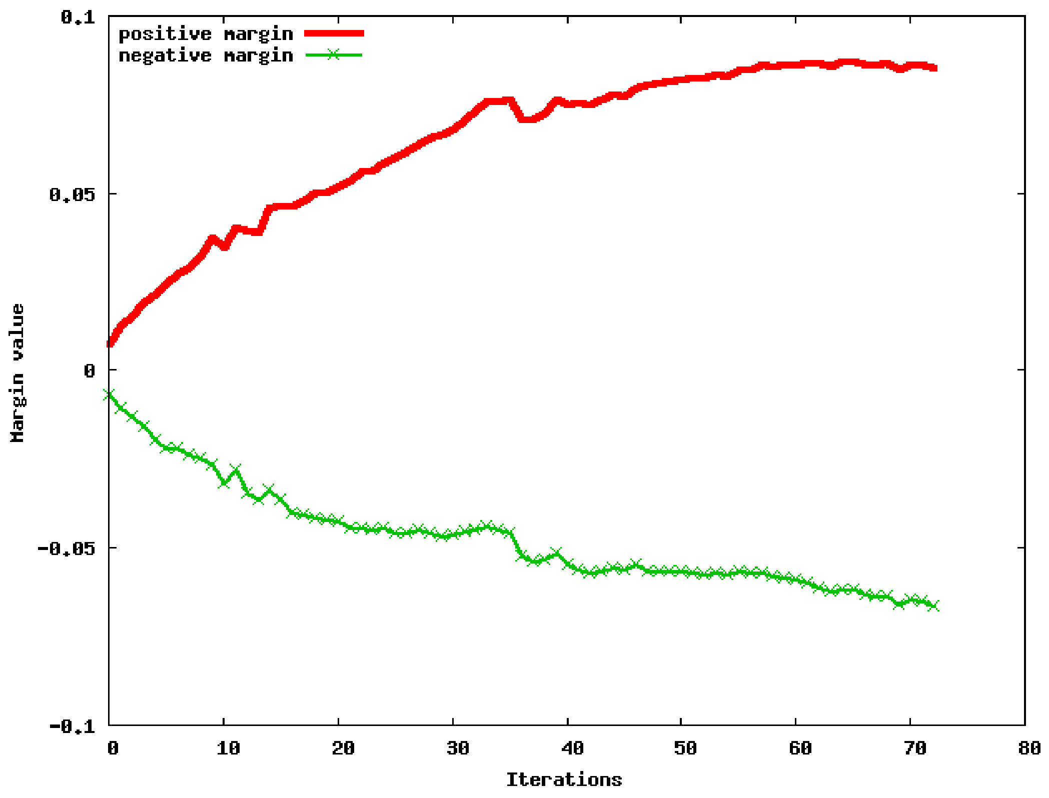


Figure 5.3: Behavior of the positive and negative margins for the “heart” dataset.

Chapter 6

Pseudo-Boolean Regression

6.1 Introduction

Linear regression is a frequently used tool in statistics and machine learning with a vast literature devoted to it [34, 60, 73, 83, 93, 115, 119]. Let $\Omega = \{\omega^1, \ldots, \omega^m\}$ be a set of m real n-vectors, and let $\{y^1, \ldots, y^m\}$ be the values of a real-valued *target function* $r : \Omega \to \mathbb{R}$, with $y^i = r(\omega^i)$ $(i = 1, \ldots, m)$, whose expression is unknown. The linear regression problem consists in finding a function of the form $f(\omega) = \beta_0 + \sum_{j=1}^{n} \beta_j \omega_j$ that approximates the values of the target function r in Ω as closely as possible, with $\beta_j \in \mathbb{R},\ j = 0, \ldots, n$.

The n components of the vectors in Ω are usually called the n *independent variables* or *predictors* of the regression problem, while the vector $\mathbf{Y} = (y^1, \ldots, y^m)^\top$ is called the *dependent variable*. In this chapter we assume that the given dataset Ω consists of binary n-vectors. If Ω is not originally binary, we apply the procedure described in Chapter 2 as a preprocessing step in order to bring the data into a binary representation.

A commonly used criterion to measure how closely f approximates the values of r is the *Least Absolute Residuals* (LAR) criterion [16, 18, 29, 48, 60, 72, 78, 88], also known as *Least Absolute Deviation*, *L_1–Approximation*, or *Least Absolute Values*, which measures the sum of the absolute values of the residuals $y^i - f(\omega^i)$, i.e., $\sum_{i=1}^{m} |y^i - f(\omega^i)|$. The problem of finding a LAR-optimal linear approximation f of r in Ω can be solved by linear programming and other techniques [16, 18, 88]. In this chapter we solve the following optimization problem to obtain values of $\beta = (\beta_0, \beta_1, \ldots, \beta_n)^\top$ that minimize the LAR criterion:

$$\text{minimize} \quad \sum_{i=1}^{m} |e_i| \quad \text{subject to} : \beta_0 + \sum_{j=1}^{n} \beta_j \omega_j^i + e_i = y^i, \quad i = 1, \ldots, m, \qquad (6.1)$$

where $\beta \in \mathbb{R}^{n+1}$ and $e_i \in \mathbb{R}$, for every $\omega^i \in \Omega$. We show later that with a proper rewriting of its constraints, problem (6.1) becomes a linear program.

In many cases, a linear function in the original independent variables is not suitable to approximate r. To overcome this problem one can try to fit a more complex function to the data, or map the data points to a new space, in which a hopefully more accurate linear fit is possible. In Section 6.2 we briefly discuss such alternatives for regression.

In the remainder of this chapter we show that a better approximation of r in Ω can be found as a linear combination of conjunctions involving the original (binary) variables. In Section 6.3 we propose an iterative algorithm for finding such an approximation by solving a large linear program by column generation, utilizing an approach similar to the one described in Chapter 5. In the iterative process additional conjunctions are gradually introduced into the expression of the approximant with the aim of compensating for large deviations, until we obtain a set of conjunctions over which the LAR criterion is minimized.

In Section 6.4 we present a comparison of the results of our algorithm with those of frequently used regression algorithms in terms of both mean absolute error and correlation. We conclude in Section 6.5 that our algorithm displays superior performance, in addition to providing a relatively simple regression function.

6.2 Related Work

While a linear regression frequently provides a satisfactory approximation of the original function r, in many cases a linear combination of the original variables cannot describe the underlying phenomenon to a sufficient extent. In some applications, certain transformations of the original variables can be used in conjunction with them in order to produce a reasonable approximation. In other cases, a function which depends nonlinearly on the parameters β is more appropriate to approximate r. In this section we briefly outline such alternatives.

A common way of obtaining more accurate approximations is to transform the original data by constructing new predictors based on the original independent variables.

The target function may depend on certain relations between the original independent variables, such as products and ratios. In this context, we have methods such as *polynomial regression* [83, 93] and *support vector regression* [115, 118, 119].

In polynomial regression the predictors are the set of monomials up to a certain degree. Note that polynomial regression is still a linear regression technique. As we shall see, polynomial regression in binary space differs from our algorithm in that the Boolean conjunctions used in our pseudo-Boolean regression can involve complemented variables, and can have arbitrary degree. Moreover, our approach constructs the conjunctions as needed, while in polynomial regression the maximum degree k is fixed a priori and all monomials of degree up to k are enumerated.

The standard support vector regression algorithm implicitly maps the points in Ω to a higher-dimensional space and applies a linear regression technique in that space. This is achieved with the use of the so-called *kernel function* that allows for the regression calculations to be performed in the large dimensional transformed space without the need to actually carry out the transforming computations. The choice of the kernel function to be used is highly dependent on the nature of the data, and different kernel functions have been developed for particular types of data [73, 115, 119].

The concept of Pseudo-Boolean Regression described in this chapter bears some similarity with that of *Logic Regression*, recently proposed in [108, 109]. While the algorithm described here can be seen as a natural extension of the LAD methodology (accompanied by a novel and opportunistic use of the column generation technique for constructing the regression function while maximizing fitness), Logic Regression seems to have evolved on its own, as a method aimed at tackling bioinformatics problems. Despite of the fact that these two regression methods are in principle equivalent (i.e., the same functions can potentially the obtained, depending on the scoring function utilized and on the random factor involved in the Logic Regression algorithm), we would like to remark an important difference.

In Logic Regression the terms of the regression function are Boolean functions in disjunctive normal form (DNF), represented as trees, with a corresponding weight. Given the complexity of searching in the space of such trees, the fitting of a Logic Regression is performed in a heuristic way: at a given iteration one of the trees in the current regression function is modified by means of one of several elementary operations, and the parameters of the function are re-adjusted. This local search is embedded in a simulated annealing scheme with a scoring function such as the residual sum of squares.

On the other hand, the pseudo-Boolean Regression algorithm globally optimizes the fitness criterion (least absolute residuals) in a deterministic way. We will show that since the least absolute residuals criterion can be written as a linear function of the model parameters, its optimization becomes straightforward. Moreover, the algorithm proposed here allows for a number of simple mechanisms for preventing overfitting ad controlling model complexity.

In *nonlinear regression* the approximant function f depends nonlinearly on the parameters β. A common misconception regarding nonlinear regression is that it consists of fitting a function involving nonlinear relations between the original variables to the data. As long as the approximant function is linear on the parameters β, regardless of what type of independent variables are used, the problem is still considered as linear regression. In order to successfully apply nonlinear regression, it is usually necessary to have some previous knowledge about the nature and shape of the function r. In many cases, such information is not available. Moreover, depending on the function used, the optimization of a fit criterion such as least squares or LAR can become rather complicated. The optimization of the least squares criterion in linear regression does not present any computational difficulty, while for nonlinear regression this is not the case in general [57, 93, 116]. Indeed, a number of software packages provide extensive support to linear regression, but support to nonlinear regression capabilities is scarce and limited to certain types of functions.

6.3 Data Transformation

In this section we describe the algorithm proposed for constructing a set of conjunctions that allows the global minimization of the LAR criterion over the set of all possible conjunctions. The algorithm can be regarded as a feature construction procedure that maps the original dataset to a higher-dimensional space.

6.3.1 Binarization

Many real-life datasets contain numerical or nominal variables. In such cases the application of our algorithm requires a preprocessing step, just as it was the case for the algorithms proposed in Chapters 3 and 5.

For the purpose of describing the algorithm in this chapter we assume that the dataset Ω is given in binary form. If the original dataset is not binary, we assume that the procedure described in Section 2.2 has been previously applied to it.

6.3.2 Conjunction-space

We recall that a conjunction, or monomial, is simply a product of literals from $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$. Similarly to a pattern, we say that a conjunction *covers* a point in $\{0,1\}^n$ if it takes the value 1 (true) on that point. The *coverage* of a conjunction (with respect to Ω) is simply the number of points of Ω covered by it. Note, however, that unlike a pattern, there is no constraint on the set of points covered by a conjunction, considering that there is no distinction between classes of points in the regression setting.

Our algorithm consists in creating a set $\mathcal{C}$ of conjunctions and associating to it an extended representation $\widehat{\Omega}$ of Ω, where each point $\omega \in \Omega$ is mapped to the characteristic vector of conjunctions in $\mathcal{C}$ that are satisfied by ω. Let $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ be a set of conjunctions in n binary variables. The extended dataset $\widehat{\Omega}$ associated to $\mathcal{C}$ is constructed by mapping each point ω in Ω to $\big(C_1(\omega), \ldots, C_{|\mathcal{C}|}(\omega)\big)$, where $\mathcal{C}_j(\omega) = 1$ if $\mathcal{C}_j$ covers ω, and $\mathcal{C}_j(\omega) = 0$ otherwise.

Ideally, we would like to map Ω to the set of all conjunctions up to a certain degree $k > 1$ and run a standard regression algorithm on that space. Since this set of conjunctions includes all the positive (i.e., uncomplemented) literals as a subset, it is to be expected that, by using these conjunctions as independent variables, the values of the target function can be approximated with significantly more precision than they would be if only the original variables were used. On the other hand, not only the number of such conjunctions can be very large, depending on n and k, but many of these conjunctions may not be necessary for the construction of a good approximant. In particular, if the number of conjunctions of degree up to k is much larger than m, we can expect that a relatively small subset of conjunctions is enough to provide a good approximation of r. In the next section we present an algorithm that constructs a set $\mathcal{C}^*$ of conjunctions of arbitrary degree based on which we can obtain a solution to the minimization of the LAR criterion over the set of all conjunctions.

6.3.3 Constructing an Optimal Conjunction-Space

As discussed in the previous section, the computational effort of generating *all* conjunctions satisfying certain requirements and of applying a regression algorithm on a very large set of variables is not justified, in general.

Let $\mathbb{C}$ denote the set of all conjunctions satisfied by at least one point in Ω. In this section, we propose a column generation algorithm for constructing a subset $\mathcal{C} \subset \mathbb{C}$ of conjunctions that allows for an approximation of the target function which is as accurate as the one that would be obtained by using $\mathbb{C}$.

Rewriting (6.1) as a linear program we obtain:

$$
\begin{aligned}
\text{(R)} \quad & \text{minimize} \quad \sum_{i=1}^{m} e_i \\
& \text{subject to:} \\
& \qquad e_i + \beta_0 + \sum_{j=1}^{n} \beta_j \omega_j^i \geq \quad y^i, \;\; i = 1, \ldots, m \\
& \qquad e_i - \beta_0 - \sum_{j=1}^{n} \beta_j \omega_j^i \geq -y^i, \;\; i = 1, \ldots, m, \\
& \qquad e_i \geq 0, \;\; i = 1, \ldots, m,
\end{aligned}
$$

where each variable e_i corresponds to the absolute value of the error $y^i - f(\omega^i)$, for $i = 1, \ldots, m$. Let $\mathcal{C}^0 = \{C_1, \ldots, C_{|\mathcal{C}^0|}\} \subset \mathbb{C}$, and let us denote by $(\mathrm{R}(\mathcal{C}^0))$ the variant of problem (R) in which we seek an LAR-optimal approximation of function r in the truncated transformed space $\{0,1\}^{|\mathcal{C}^0|}$ corresponding to the projection of Ω on $\mathcal{C}^0$:

$$\begin{aligned}
(\mathrm{R}(\mathcal{C}^0)) \quad & \text{minimize} \quad \sum_{i=1}^{m} e_i \\
& \text{subject to:} \\
& \qquad e_i + \beta_0 + \sum_{j=1}^{|\mathcal{C}^0|} \beta_j C_j(\omega^i) \geq \ \ y^i, \ \ i = 1, \ldots, m \qquad (6.2) \\
& \qquad e_i - \beta_0 - \sum_{j=1}^{|\mathcal{C}^0|} \beta_j C_j(\omega^i) \geq -y^i, \ \ i = 1, \ldots, m \qquad (6.3) \\
& \qquad e_i \geq 0, \ \ i = 1, \ldots, m. \qquad (6.4)
\end{aligned}$$

The solution of $(\mathrm{R}(\mathcal{C}^0))$ provides a vector β of weights, defining a function that minimizes the LAR criterion over the current set of conjunctions $\mathcal{C}^0$. Clearly, this function is not necessarily optimal for $(\mathrm{R}(\mathbb{C}))$, or even for any $(\mathrm{R}(\tilde{\mathcal{C}}))$, with $\mathcal{C}^0 \subset \tilde{\mathcal{C}}$. In order to verify global optimality with respect to the entire family of conjunctions under consideration, say $\tilde{\mathcal{C}}$, we need to answer the question of whether there is any conjunction, or set of conjunctions in $\tilde{\mathcal{C}} \setminus \mathcal{C}^0$ whose addition to $\mathcal{C}^0$ may decrease the objective function of $(\mathrm{R}(\mathcal{C}^0))$. Similarly to our description in Chapter 5, this question can be answered by verifying whether there exists a conjunction in $\tilde{\mathcal{C}} \setminus \mathcal{C}^0$ with negative reduced cost.

Let us associate to constraints (6.2) and (6.3) the vectors of dual variables λ and μ, respectively. Let (e^*, β^*) be the optimal solution of $(\mathrm{R}(\mathcal{C}^0))$, and let (λ^*, μ^*) be the corresponding dual variables.

By standard linear programming manipulations, we can find that the reduced cost of a conjunction $C_j \in \mathbb{C} \setminus \mathcal{C}^0$ is given by $-(\lambda^* - \mu^*)^\top a_j$, where a_j is the characteristic vector $\big(C_j(\omega^1), \ldots, C_j(\omega^m), -C_j(\omega^1), \ldots, -C_j(\omega^m)\big)^\top$ of the set of points covered by C_j. The following pseudo-Boolean optimization problem finds the conjunction with the most negative reduced cost:

$$\begin{aligned}
(\mathrm{S}(\lambda^*, \mu^*)) \qquad & \text{maximize} \qquad \sum_{k=1}^{m} \left(\prod_{i:\omega_i^k = 0} \overline{p_i} \prod_{j:\omega_j^k = 1} \overline{p_j^c} \right) (\lambda_k^* - \mu_k^*) \\
& \text{subject to:} \qquad p_j, p_j^c \in \{0,1\}, \ j = 1, \ldots, n,
\end{aligned}$$

where the binary decision variable p_j corresponds to the inclusion or not of literal x_j in the resulting conjunction, binary variable p_j^c corresponds to the inclusion or not of literal $\overline{x}_j$ in the resulting conjunction, and each term $\prod_{i:\omega_i^k=0} \overline{p_i} \prod_{j:\omega_j^k=1} \overline{p_j^c}$ takes the value 1 whenever ω^k is covered by the resulting conjunction, and 0 otherwise.

Once a conjunction $C_j \in \mathbb{C} \setminus \mathcal{C}^0$ with negative reduced cost has been found, it is introduced into the set of conjunctions under consideration, and an augmented version of problem (R($\mathcal{C}^0$)) is solved. The algorithm proceeds with the alternate solution of an instance of problem (R($\mathcal{C}^i$)), for some $\mathcal{C}^i \subset \mathbb{C}$, and an instance of problem (S). In general, at iteration i we solve (R($\mathcal{C}^i$)), obtain the dual variables λ^* and μ^* associated to its optimal solution, and solve the corresponding problem (S(λ^*, μ^*)), obtaining a conjunction C_j. If the optimal objective function value of (S(λ^*, μ^*)) is strictly positive, then we define $\mathcal{C}^{i+1} := \mathcal{C}^i \cup \{C_j\}$ and go on to the next iteration by solving (R($\mathcal{C}^{i+1}$)). Otherwise, the function f constructed at the current iteration is optimal with respect to the LAR criterion over $\mathbb{C}$.

It can be seen that the above algorithm fits the usual prototype of column generation algorithms, in which a large LP problem is solved by generating only a relatively small subset of its columns, taking advantage of the fact that the columns of the problem possess a well-defined structure. In Figure 6.1 we show the pseudo-code of the algorithm.

```
1. Input: $\Omega^+, \Omega^-$, initial set $\mathcal{C}^0$ of conjunctions. Set $k = 0$.
2. Solve (R($\mathcal{C}^k$)), with optimal primal and dual solutions $(e^*, \beta^*)$ and $(\lambda^*, \mu^*)$.
3. Solve (S($\lambda^*, \mu^*$)), with optimal conjunction $T^*$.
4. If at least one of the stopping criteria is met, stop.
5. Define $\mathcal{C}^{k+1} = \mathcal{C}^k \cup \{T^*\}$ and set $k = k + 1$.
6. Go to Step 2.
```

Figure 6.1: Pseudo-code of the pseudo-Boolean regression algorithm.

6.3.4 Implementation

We remark that problem (S(λ^*, μ^*)) is precisely the same problem for which we developed the branch-and-bound algorithm of Chapter 4. In the computational experiments reported in this chapter, we applied algorithm BBPBF to solve (S(λ^*, μ^*)). As in Chapter 5, we imposed a maximum limit of 1,000 branch-and-bound nodes in the initialization phase of the BBPBF algorithm, allowing for extra 2,000 nodes for the remainder of the search, and a maximum running time of 30 minutes.

Given the typical convergence behavior of column generation algorithms we adopt a slightly more flexible stopping criterion than the one described above for the case when the value of the optimal solution of $(\mathrm{S}(\lambda^*, \mu^*))$ is close to zero.

As argued in Chapter 5, the column generation algorithm can display some inconvenient behavior such as degenerate iterations and the tailing-off phenomenon towards the end of its execution. Here we applied a similar policy to the one described in Chapter 5 to deal with such situations. A tolerance value of 10^{-4} was used to define the minimum decrease in the objective function value of $(\mathrm{R}(\mathcal{C}^i))$ at any given iteration i that is considered a "significant decrease." The maximum number of iterations without a significant decrease in the objective function was set to 10. Obviously, these settings do not rule out the possibility of the algorithm stopping at a sub-optimal solution. As pointed out in Chapter 5, this is not a real concern in our context as in general we are not interested in a perfect fit of the data.

Two more stopping criteria were included in our implementation with the aim of preventing the overfitting of the training data. We describe them below.

- If during 10 consecutive iterations the conjunctions generated by solving problem $(\mathrm{S}(\lambda^*, \mu^*))$ cover a small number of points from Ω we stop the execution of the algorithm. Such a behavior suggests that the algorithm is overfitting the training data by adjusting the regression function to very particular trends in the data, rather than capturing global trends. We utilized a threshold of 5% of $|\Omega|$, below which the coverage of a conjunction is considered too small.

- If during 10 consecutive iterations the number of conjunctions having a nonzero coefficient β_j in the optimal solution of $(\mathrm{R}(\mathcal{C}^i))$ has been larger than $0.1|\Omega|$, we stop the execution of the algorithm. A rule of thumb for the practical application of regression algorithms is that the number of predictors used should be limited by a fraction of $\frac{1}{10}$ or $\frac{1}{8}$ of the number of data points [63].

6.4 Computational Experiments

In this Section we report the results of an experimental comparison of the algorithm presented in the previous section, termed PBR (Pseudo-Boolean Regression), and three algorithms available from the Weka package [128]: Support Vector Regression (SVR), Multilayer Perceptron Neural Network (MP) and Linear Regression (LR).

In our experiments we utilized five datasets that are freely available and frequently used as benchmarks for regression algorithms in the machine learning and statistics literature. The datasets were obtained from the Machine Learning Repository of the University of California Irvine and from the DELVE repository (`http://www.cs.toronto.edu/~delve/`). The list of datasets is: Abalone (AB), Boston Housing (BH), Auto-mpg (MPG), Robot Arm Kinematics (RAK), and Servo (SV).

	Number of		
Datasets	Observations	Original Vars.	Binarized Vars.
AB	4,177	10	38
BH	506	13	42
MPG	398	7	23
RAK	8,192	8	8
SV	167	4	15

Table 6.1: Datasets used in regression experiments.

Since none of these datasets was originally binary, we binarized them using the procedure discussed in Section 2.2. The binarization procedure was carried out with the right-hand side equal to 1 for constraints (2.1), and with values of the τ parameter between 1% and 50%. Table 6.1 shows the number of original and binarized variables for each dataset.

We report in Tables 6.2 and 6.3 the average mean absolute error and correlation of each algorithm over 10 random experiments on each dataset. In each experiment a randomly selected set of observations, consisting of 90% of the original dataset, was used for fitting a regression function, and the mean absolute error and correlation on the remaining 10% set of observations are recorded. For datasets AB and RAK we utilized 75% of the original training data to fit the regression function and recorded the values of mean absolute error and correlation on the remaining 25%. In all cases, 10 random experiments were performed. We also include in the two tables the Borda count of each algorithm.

	Mean Absolute Error					
Algorithms	AB	BH	MPG	RAK	SV	Borda
LR	1.59	3.33	2.73	0.16	0.87	9
MP	1.62	2.94	2.90	0.13	0.44	14
SVR	1.54	3.17	2.63	0.16	0.70	12
PBR	1.82	3.11	2.37	0.15	0.29	15

Table 6.2: Mean absolute error of regression algorithms applied to 5 datasets.

	Correlation					
Algorithms	AB	BH	MPG	RAK	SV	Borda
LR	0.73	0.86	0.89	0.64	0.67	9
MP	0.75	0.91	0.92	0.82	0.90	19
SVR	0.73	0.84	0.89	0.64	0.63	9
PBR	0.51	0.86	0.89	0.68	0.94	13

Table 6.3: Correlation of regression algorithms applied to 5 datasets.

It can be seen that the pseudo-Boolean regression algorithm presents a performance comparable to that of Multilayer Perceptron, which is the best among the Weka algorithms. According to the Borda count, the pseudo-Boolean regression algorithm is slightly superior in terms of mean absolute error, while Multilayer Perceptron outperforms all other algorithms in terms of correlation.

6.5 Conclusions

In this chapter we proposed a novel algorithm that extends the LAD methodology to deal with regression problems. We presented the results of computational experiments that suggest that our algorithm is comparable to standard regression algorithms in terms of mean absolute error and correlation.

The pseudo-Boolean regression algorithm described here can be viewed as the iterative construction of a pseudo-Boolean function f on the n binary variables of Ω with the objective of fitting the values of r. Our algorithm directly optimizes the Least Absolute Residual criterion over the set of all conjunctions. The coordinating linear programming formulation of (R($\mathcal{C}^i$)) provides the necessary information for improving the quality of the approximation at each iteration, while the subproblem procedure exploits the type of additional predictors (binary conjunctions) to search for the best additional predictor in a computationally affordable manner.

Note that, although the resulting pseudo-Boolean function f is a nonlinear function of the original n variables, our algorithm is a linear regression technique, since f depends linearly on the parameters β.

Our algorithm can be applied in a more general context, in which numerical variables are used as predictors, in addition to a set of binary predictors. In this setting, our algorithm will construct a set of conjunctions that best complements the given binary and numerical variables. The same column generation algorithm can be applied to generate different types of predictors, as long as the search for a *candidate predictor* to improve the current regression function can be carried out.

Finally, the conjunctions in the resulting pseudo-Boolean approximant may suggest relations between the original variables that may not be apparent at a first glance. Due to the binarization procedure and the construction of arbitrary conjunctions, our algorithm constructs a set of additional predictors that cannot be obtained by most regression approaches, where typically numerical products of two or three of the original variables are used as additional predictors.

Chapter 7
Conclusions

In this thesis we have described some novel and opportunistic applications of optimization to the construction of LAD models for classification and regression. Below, we quickly review the standard implementation of LAD and highlight some ways in which this study has made a meaningful contribution.

7.1 The Standard LAD Implementation

The standard implementation of LAD relies on the calibration of a number of important parameters that control the type of patterns generated, and the type of models constructed.

The typical set of parameters related to the pattern generation procedure that have to be adjusted is: the degree of patterns, the family of patterns generated (prime, spanned, strong patterns, or some combination of those), the minimum homogeneity of a pattern, and its minimum prevalence.

In addition to the pattern generation process, other choices have to be made regarding the models constructed. The number of times each observation in the training data must be covered by the patterns in the model represents a tradeoff between model complexity and robustness. Another important choice – which has not been fully explored in the LAD literature – is the type of discriminant function utilized once a model has been constructed. The standard implementation of LAD utilizes a simple linear function, where all positive patterns have equal weights, and all negative patterns have equal weights. Consequently, the discriminant function typically used in LAD classifies an observation according to the difference between the percentage of positive patterns and of negative patterns covering it.

The LAD discriminant function can be seen as a separating hyperplane in the so-called pattern-space. Clearly, different discriminant functions can be constructed on the same pattern-space with very different classification accuracies. It is to be expected that a carefully defined discriminant function could make a difference in the resulting accuracy of a model.

It has been observed in several real-life studies [1, 5, 9, 10, 31, 6] that a careful calibration of LAD parameters allows the construction of highly accurate models that frequently outperform those obtained with commonly used machine learning methods, such as SVM, decision trees, neural networks, etc. Moreover, LAD models typically provide additional insight into the data being analyzed that can hardly be obtained with the use of a single machine learning/data mining algorithm. As examples we mention the relative importance of variables, the automatic detection of outliers and possibly misclassified observations, and the detection of representative subgroups of observations within one of the classes considered.

However, to fully benefit from the classification power and data mining capabilities of LAD, one must often go through a time-consuming process of calibrating the parameters mentioned above. The main goal of this thesis is to provide, and evaluate computationally, some alternative ways of constructing LAD models.

7.2 Our Main Contributions

The major contribution of this thesis can be summarized as the application of optimization methods to overcome some of the difficulties in the construction of accurate LAD models. In order to achieve this goal, two main points had to be addressed. First, a precise definition of an objective function, such as a global fitness criterion, was necessary. Second, efficient ways of solving the associated optimization problems were needed. Below we discuss how we addressed these points in the chapters of this thesis.

Clearly, the accuracy of a LAD model is directly related to the quality of the patterns in the model (i.e., mostly their prevalence and homogeneity). Therefore, a natural idea is to define a LAD model on the basis of a set of patterns having very large prevalences and homogeneities. Along these lines, we defined in Chapter 3 a nonlinear set covering problem for finding a pattern of largest prevalence covering a given observation and respecting a certain measure of fuzziness, which is a concept closely related to that of homogeneity. Since the exact solution of the nonlinear set covering problem required a substantial running time, we proposed three simple and efficient heuristics that allowed the construction of "approximately maximum" patterns, while requiring just a fraction of the running time required by the exact approach. The LAD models constructed with these patterns were shown to be highly accurate, clearly outperforming most of the other machine learning algorithms used as benchmarks. Another important feature of these models is the utilization of a single control parameter to be calibrated.

In Chapter 5 we proposed an algorithm for the construction of LAD models that optimizes a different criterion: the margin of separation between the two classes of observations. In other words, we were interested in defining a linear discriminant function on the set of *all* patterns, such that the difference between the smallest value of the discriminant function on a positive observation and the largest value of the discriminant function on a negative point is maximized. We formulated this problem as a linear program and showed how to solve it by column generation. While the solution of the linear program for a given (relatively small) set of patterns is a routine task, finding a new pattern to be included in the formulation is not trivial. To accomplish this task we utilized a branch-and-bound algorithm, which was proposed in Chapter 4 and showed to be significantly superior to the commercial integer linear programming solver Xpress [43]. The linear programming formulation, coupled with the fast solution of the column generation subproblem, allowed us to construct LAD models that are optimal according to the margin of separation criterion, while requiring an affordable running time. Moreover, such LAD models are parameter-free and practically as accurate as the best of the machine learning algorithms used for comparison.

In Chapter 6 we presented a column generation model similar to the one of Chapter 5 for addressing regression problems, an extension of the LAD methodology that had not been explored in detail in the LAD literature. Our algorithm minimizes the *Least Absolute Residuals* criterion by solving a linear program on the set of all *conjunctions*

– a conjunction being simply a set of conditions similar to a pattern, but lacking the associated concept of homogeneity. The branch-and-bound algorithm of Chapter 4 was utilized for finding the most suitable conjunction to be included in the formulation at each iteration. Our computational results showed that the procedure is effective and provides an attractive alternative to well-known regression methods.

In summary, this thesis has (i) introduced novel, efficient ways of constructing LAD classification models having high accuracies and requiring minimal, if any, calibration of control parameters; and (ii) extended the LAD methodology to deal with the important class of regression problems that appears frequently in data analysis tasks.

References

[1] Abramson S., G. Alexe, P.L. Hammer, D. Knight and J. Kohn, *A computational approach to predicting cell growth on polymeric biomaterials*, Journal of Biomedical Material Research Part A, 73 (2005), pp. 116–124.

[2] Achterberg T., *SCIP-a framework to integrate constraint and mixed integer programming*, Konrad-Zuse-Zentrum fur Informationstechnik Berlin, ZIB-Report, (2004), pp. 04–19.

[3] Ahmed S., M. Tawarmalani, N.V. Sahinidis, *A finite branch-and-bound algorithm for two-stage stochastic integer programs*, Mathematical Programming, 100 (2004), pp. 355–377.

[4] Alexe G., P.L. Hammer, *Spanned patterns for the logical analysis of data*, Discrete Appl. Math., 154 (2006), pp. 1039–1049.

[5] Alexe G., S. Alexe, D. Axelrod, P.L. Hammer, D. Weissmann, *Logical analysis of diffuse large B-cell lymphomas*, Artificial Intelligence in Medicine, 34 (2005), pp. 235–67.

[6] Alexe G., S. Alexe, D. E. Axelrod, T. O. Bonates, I. Lozina, M. Reiss, P. L. Hammer, *Breast cancer prognosis by combinatorial analysis of gene expression data*, Breast Cancer Research, 8:R41 (2006).

[7] Alexe G., S. Alexe, P.L. Hammer, *Pattern-based clustering and attribute analysis*, Soft Computing, 10 (2006), pp. 442–452.

[8] Alexe G., S. Alexe, P.L. Hammer, A. Kogan, *Comprehensive vs. comprehensible classifiers in logical analysis of data*, Discrete Applied Mathematics, in press, (2007).

[9] Alexe G., S. Alexe, P.L. Hammer, L. Liotta, E. Petricoin, and M. Reiss, *Ovarian cancer detection by logical analysis of proteomic data*, Proteomics, 4 (2004), pp. 766–783.

[10] Alexe S., E. Blackstone, P.L. Hammer, H. Ishwaran, M. Lauer, C. Pothier Snader, *Coronary risk prediction by logical analysis of data*, Annals of Operations Research, 119 (2003), pp. 15–42.

[11] Alexe S., P.L. Hammer, *Accelerated algorithm for pattern detection in logical analysis of data*, Discrete Appl. Math., 154 (2006), pp. 1050–1063.

[12] Balas E., M.J. Saltzman, *An algorithm for the three-index assignment problem*, Operations Research, 39 (1991), pp. 150–161.

[13] Banfield R.E., L.O. Hall, K.W. Bowyer, W.P. Kegelmeyer, *A comparison of decision tree ensemble creation techniques*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 29 (2007), pp. 173–180.

[14] Barnhart C., C.A. Hane, P.H. Vance, *Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems*, Operations Research, 48 (2000), pp. 318–326.

[15] Barnhart C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, P.H. Vance, *Branch-and-price: Column generation for solving huge integer programs*, Operations Research, 46 (1998), pp. 316–329.

[16] Barrodale I., F.D.K. Roberts, *An improved algorithm for discrete l_1 linear approximation*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 839–848.

[17] Bazan J.G., M.S. Szczuka, J. Wroblewski, *A new version of rough set exploration system*, Proceedings of the Third International Conference on Rough Sets and Current Trends in Computing, (2002), pp. 397–404.

[18] Bloomfield P., W.L. Steiger, *Least Absolute Deviations: Theory, Applications, and Algorithms*, Birkhäuser Boston, Boston, Mass, USA, 1983.

[19] Blum A., P. Langley, *Selection of relevant features and examples in machine learning*, Artificial Intelligence, 97 (1997), pp. 245–271.

[20] Bonates T.O., P.L. Hammer, A. Kogan, *Maximum patterns in datasets*, Discrete Appl. Math. (in press), (2007).

[21] Borda J.C., *Mémoire sur les élections au scrutin*, Histoire de l'Academie Royale des Sciences, (1781).

[22] Boros E., P.L. Hammer, *Cut-polytopes, Boolean quadratic polytopes and nonnegative quadratic pseudo-Boolean functions*, Mathematics of Operations Research, 18 (1993), pp. 245–253.

[23] ——, *Pseudo-Boolean optimization*, Discrete Applied Mathematics, 123 (2002), pp. 155–225.

[24] Boros E., P.L. Hammer, G. Tavares, *Local search heuristics for unconstrained quadratic binary optimization*, Tech. Rep. RRR 09-2005, RUTCOR - Rutgers Center for Operations Research, Rutgers University, 2005.

[25] ——, *A max-flow approach to improved lower bounds for quadratic 0-1 minimization*, Tech. Rep. RRR 07-2006, RUTCOR - Rutgers Center for Operations Research, Rutgers University, 2006.

[26] ——, *Preprocessing of unconstrained quadratic binary optimization*, Tech. Rep. RRR 10-2006, RUTCOR - Rutgers Center for Operations Research, Rutgers University, 2006.

[27] Boros E., P.L. Hammer, T. Ibaraki, A. Kogan, *Logical analysis of mumerical data*, Mathematical Programming, 79 (1997), pp. 163–190.

[28] Boros E., P.L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz, I. Muchnik, *An implementation of logical analysis of data*, IEEE Transactions on Knowledge and Data Engineering, 12 (2000), pp. 292–306.

[29] Boscovich R.J., *De litteraria expeditione per pontificiam ditioned, et synopsis amplioris operis, ac habentur plura ejus ex exemplaria etiam sensorum impressa*, Bononiensi Scientiarum et Artum Instituto Atque Academia Commentarii, 4 (1757), pp. 353–396.

[30] Boser B.E., I.M. Guyon, V.N. Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the fifth annual workshop on Computational learning theory, (1992), pp. 144–152.

[31] Brauner M.W., N. Brauner, P.L. Hammer, I. Lozina, D. Valeyre, *Logical analysis of computed tomography data to differentiate entities of idiopathic Interstitial pneumonias*, Data Mining in Biomedicine, Biocomputing, Springer, (2007).

[32] Breiman L., *Bagging predictors*, Machine Learning, 24 (1996), pp. 123–140.

[33] ——, *Random forests*, Machine Learning, 45 (2001), pp. 5–32.

[34] Breiman L., J. Friedman, C.J. Stone, R.A. Olshen, *Classification and Regression Trees*, Chapman & Hall/CRC, 1984.

[35] Bshouty N.H., N. Eiron, *Learning monotone DNF from a teacher that almost does not answer membership queries*, Journal of Machine Learning Research, 3 (2003), pp. 49–57.

[36] Chvátal V., *Edmonds polytopes and a hierarchy of combinatorial problems*, Discrete Mathematics, 4 (1973), pp. 305–337.

[37] ——, *A greedy heuristic for the set-covering problem*, Mathematics of Operations Research, 4 (1979), pp. 233–235.

[38] ——, *Linear Programming*, Freeman, New York, 1983.

[39] Cox D.R., *The Analysis of Binary Data*, Chapman and Hall New York, 1970.

[40] Crama Y., P. Hansen, B. Jaumard, *The basic algorithm for pseudo-Boolean programming revisited*, Discrete Applied Mathematics, 29 (1990), pp. 171–185.

[41] Crama Y., P.L. Hammer, T. Ibaraki, *Cause-effect Relationships and Partially Defined Boolean Functions*, Annals of Operations Research, 16 (1988), pp. 299–325.

[42] Cruz F.B., G.R. Mateus, J.M. Smith, *A cranch-and-bound algorithm to solve a multi-level network optimization problem*, Journal of Mathematical Modelling and Algorithms, 2 (2003), pp. 37–56.

[43] Dash Associates, *Xpress-Mosel Reference Manuals and Xpress-Optimizer Reference Manual, Release 2004G*, 2004.

[44] Dietterich T.G., *Overfitting and undercomputing in machine learning*, ACM Computing Surveys (CSUR), 27 (1995), pp. 326–327.

[45] ——, *Approximate statistical tests for comparing supervised classification learning algorithms*, Neural Computation, 10 (1998), pp. 1895–1924.

[46] ——, *An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization*, Machine Learning, 40 (2000), pp. 139–157.

[47] ——, *Ensemble methods in machine learning*, Lecture Notes in Computer Science, 1857 (2000), pp. 1–15.

[48] Dodge Y., *Statistical Data Analysis Based on the L1-Norm and Related Methods*, Birkhäuser Basel, 2002.

[49] Dong G., J. Li, *Efficient mining of emerging patterns: Discovering trends and differences*, Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, (1999), pp. 43–52.

[50] Eckstein J., *Parallel branch-and-bound algorithms for general mixed integer programming on the CM-5*, SIAM Journal on Optimization, 4 (1994), p. 794.

[51] Eckstein J., P.L. Hammer, Y. Liu, M. Nediak, B. Simeone, *The maximum box problem and its application to data analysis*, Computational Optimization and Applications, 23 (2002), pp. 285–298.

[52] Efron B., R. Tibshirani, *Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy*, Statistical Science, 1 (1986), pp. 54–75.

[53] Efroymson M.A., T.L. Ray, *A branch and bound algorithm for plant location*, Operations Research, 14 (1996), pp. 361–368.

[54] Feige U., *A Threshold of ln n for approximating set cover*, Journal of the ACM, 45 (1998), pp. 634–652.

[55] Fischetti M., A. Lodi, P. Toth, *Solving real-world ATSP instances by branch-and-cut*, Lecture Notes In Computer Science, (2003), pp. 64–77.

[56] Freund Y., *Boosting a weak learning algorithm by majority*, Information and Computation, 121 (1995), pp. 256–285.

[57] Gallant A.R., *Nonlinear regression*, The American Statistician, 29 (1975), pp. 73–81.

[58] Gilmore P.C., R.E. Gomory, *A linear programming approach to the cutting-stock problem*, Operations Research, 9 (1961), pp. 849–859.

[59] ——, *A linear programming approach to the cutting-stock problem - part II*, Operations Research, 11 (1963), pp. 863–888.

[60] Giloni A., J.S. Simonoff, B. Sengupta, *Robust weighted LAD regression*, Computational Statistics and Data Analysis, 50 (2006), pp. 3124–3140.

[61] Gomory R.E., *Outline of an algorithm for integer solutions to linear programs*, Bulletin of the American Mathematical Society, 64 (1958), pp. 275–278.

[62] ——, *An algorithm for the mixed integer problem*, Report RM-2597, The Rand Corporation, (1960).

[63] Green S.B., *How many subjects does it take to do a regression analysis?*, Multivariate Behavioral Research, 26 (1991), pp. 499–510.

[64] Günlük O., *A branch-and-cut algorithm for capacitated network design problems*, Mathematical Programming, 86 (1999), pp. 17–39.

[65] Hammer P.L., *The Logic of Cause-effect Relationships.* Lecture at the International Conference on Multi-Attribute Decision Making via Operations Research-based Expert Systems, Passau, Germany, 1986.

[66] Hammer P.L., A. Kogan, B. Simeone, S. Szedmák, *Pareto-optimal patterns in logical analysis of data*, Discrete Applied Mathematics, 144 (2004), pp. 79–102.

[67] Hammer P.L., I. Rosenberg, S. Rudeanu, *On the determination of the minima of pseudo-boolean functions*, Studii si Cercetari Matematice, 14 (1963), pp. 359–364.

[68] Hammer P.L., R. Holzman, *Approximations of pseudo-Boolean functions: Applications to game theory*, ZOR – Methods and Models of Operations Research, 36 (1992), pp. 3–21.

[69] Hammer P.L., S. Rudeanu, *Pseudo-Boolean programming*, Operations Research, 17 (1969), pp. 233–261.

[70] Hammer P.L., T.O. Bonates, *Logical analysis of data: From combinatorial optimization to medical applications*, Annals of Operations Research, 148 (2006), pp. 203–225.

[71] Hansen E.R., *Global Optimization Using Interval Analysis*, Marcel Dekker New York, 2004.

[72] Harris T., *Regression using minimum absolute deviations*, The American Statistician, 4 (1950), pp. 14–15.

[73] Hastie T., R. Tibshirani, J.H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2001.

[74] Hoffman K.L., M. Padberg, *Solving airline crew scheduling problems by branch-and-cut*, Management Science, 39 (1993), pp. 657–682.

[75] Holmberg K., D. Yuan, *A Lagrangean heuristic based branch-and-bound approach for the capacitated network design problem*, Operations Research, 48 (2000), pp. 461–481.

[76] Hosmer D.W., S. Lemeshow, *Applied Logistic Regression*, John Wiley & Sons, 1989.

[77] Ibaraki T., *Theoretical comparisons of search strategies in branch-and-bound algorithms*, International Journal of Parallel Programming, 5 (1976), pp. 315–344.

[78] Karst O.J., *Linear curve fitting using least deviations*, Journal of the American Statistical Association, 53 (1958), pp. 118–132.

[79] Klivans A.R., R.A. Servedio, *Learning DNF in time 2 õ (n 1/3)*, Journal of Computer and System Sciences, 68 (2004), pp. 303–318.

[80] Kohavi R., *A study of cross-validation and bootstrap for accuracy estimation and model selection*, Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 14 (1995), pp. 1137–1143.

[81] Krauth W., M. Mezard, *Learning algorithms with optimal stability in neural networks*, Journal of Physics A Mathematical General, 20 (1987), pp. L745–L752.

[82] Kuncheva L.I., *Diversity in multiple classifier systems*, Information Fusion, 6 (2005), pp. 3–4.

[83] Kutner M.H., C.J. Nachtsheim, W. Wasserman, J. Neter, *Applied Linear Regression Models*, McGraw-Hill/Irwin, 1989.

[84] Land A.H., A.G. Doig, *An automatic method of solving discrete programming problems*, Econometrica, 28 (1960), pp. 497–520.

[85] Lasdon L.S., *Optimization Theory for Large Systems*, Dover Publications Inc., 2002.

[86] Lavrac N., *Subgroup discovery techniques and applications*, in Advances in Knowledge Discovery and Data Mining, vol. 3518 of Lecture Notes in Computer Science, 2005, pp. 2–14.

[87] Li C.M., F. Manya, J. Planes, *Exploiting unit propagation to compute lower bounds in branch and bound max-SAT solvers*, Proc. of the 11 thCP, Sitges, Spain, (2005).

[88] Li Y., G.R. Arce, *A maximum likelihood approach to least absolute deviation regression*, EURASIP Journal on Applied Signal Processing, 2004 (2004), pp. 1762–1769.

[89] Lubbecke M.E., J. Desrosiers, *Selected topics in column generation*, Oper. Res, 53 (2005), pp. 1007–1023.

[90] McCulloch W.S., Pitts W., *A logical calculus of ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics, 5 (1943), pp. 115–137.

[91] Minsky M.L., S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press Cambridge, MA, 1969.

[92] Mitchell T.M., *Machine Learning*, McGraw-Hill Higher Education, 1997.

[93] Montgomery D.C., E.A. Peck, G.G. Vining, *Introduction to Linear Regression Analysis*, Wiley New York, 1982.

[94] Nauss R.M., *An improved algorithm for the capacitated facility location problem*, The Journal of the Operational Research Society, 29 (1978), pp. 1195–1201.

[95] Nazareth J.L., *Computer Solution of Linear Programs*, Oxford University Press, Inc. New York, NY, USA, 1987.

[96] Nemhauser G.L., L.A. Wolsey, *Integer and Combinatorial Optimization*, Wiley-Interscience New York, NY, USA, 1988.

[97] Newman D.J., S. Hettich, C.L. Blake, C.J. Merz, *UCI Repository of Machine Learning Databases*, 2007. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[98] Ortega F., L.A. Wolsey, *A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem*, Networks, 41 (2003), pp. 143–158.

[99] Padberg M., G. Rinaldi, *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems*, SIAM Review, 33 (1991), pp. 60–100.

[100] ——, *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems*, SIAM Review, 33 (1991), pp. 60–100.

[101] Palacios H., H. Geffner, *Planning as branch and bound: A constraint programming implementation*, Proceedings of CLEI, 2 (2002).

[102] Pascheuer N., M. Jünger, G. Reinelt, *A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints*, Computational Optimization and Applications, 17 (2000), pp. 61–84.

[103] Pawlak Z., *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers Norwell, MA, USA, 1992.

[104] Quinlan J.R., *C4. 5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[105] ——, *Bagging, boosting, and C4. 5*, Proceedings of the Thirteenth National Conference on Artificial Intelligence, 725 (1996), p. 730.

[106] Rosenblatt F., *The Perceptron, a Theory of Statistical Separability in Cognitive Systems.(Project PARA)*, US Dept. of Commerce, Office of Technical Services, 1958.

[107] ——, *A Comparison of Several Perceptron Models*, Self-Organizing Systems, (1962).

[108] Ruczinski I., C. Kooperberg, M. LeBlanc, *Logic regression*, Journal of Computational and Graphical Statistics, 12 (2003), pp. 475–511.

[109] ——, *Exploring interactions in high-dimensional genomic data: An overview of Logic Regression, with applications*, Journal of Multivariate Analysis, 90 (2004), pp. 178–195.

[110] Rumelhart D.E., G.E. Hinton, R.J. Williams, *Learning Internal Representations by Error Propagation*, MIT Press Cambridge, MA, USA, 1986.

[111] Savelsbergh M., *A branch-and-price algorithm for the generalized assignment problem*, Operations Research, 45 (1997), pp. 831–841.

[112] Savelsbergh M.W.P., *Preprocessing and probing techniques for mixed integer programming problems*, Eindhoven University of Technology, Department of Mathematics and Computing Science, 1992.

[113] Schapire R.E., *The strength of weak learnability*, Machine Learning, 5 (1990), pp. 197–227.

[114] Schapire R.E., Y. Freund, P. Bartlett, W.S. Lee, *Boosting the margin: A new explanation for the effectiveness of voting methods*, The Annals of Statistics, 26 (1998), pp. 1651–1686.

[115] Schölkopf B., Smola A.J., *Learning with Kernels*, MIT Press Cambridge, Mass, 2002.

[116] Seber G.A.F., C.J. Wild, *Nonlinear Regression*, John Wiley & Sons, New York, 1989.

[117] Senne E.L.F., L.A.N. Lorena, M.A. Pereira, *A branch-and-price approach to p-median location problems*, Computers and Operations Research, 32 (2005), pp. 1655–1664.

[118] Shawe-Taylor J., N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004.

[119] Smola A.J., B. Schölkopf, *A tutorial on support vector regression*, Statistics and Computing, 14 (2004), pp. 199–222.

[120] Smola A.J., P. Bartlett, B. Schölkopf, D. Schuurmans, *Advances in Large-Margin Classifiers*, The MIT Press, 2000.

[121] Tawarmalani M., N.V. Sahinidis, *Global optimization of mixed-integer nonlinear programs: A theoretical and computational study*, Mathematical Programming, 99 (2004), pp. 563–591.

[122] Truett J., J. Cornfield, W. Kannel, *A multivariate analysis of the risk of coronary heart disease in Framingham.*, Journal of Chronic Diseases, 20 (1967), pp. 511–24.

[123] Vance P.H., C. Barnhart, E.L. Johnson, G.L. Nemhauser, *Solving binary cutting stock problems by column generation and branch-and-bound*, Computational Optimization and Applications, 3 (1994), pp. 111–130.

[124] Vandenbussche D., G.L. Nemhauser, *A branch-and-cut algorithm for nonconvex quadratic programs with box constraints*, Mathematical Programming, 102 (2005), pp. 559–575.

[125] Vidal V., H. Geffner, *Branching and pruning: An optimal temporal POCL planner based on constraint programming*, Artificial Intelligence, 170 (2006), pp. 298–335.

[126] Wang L., H. Zhao, G. Dong, J. Li, *On the complexity of finding emerging patterns*, Theoretical Computer Science, 335 (2005), pp. 15–27.

[127] W. W.E., *A technical review of column generation in integer programming*, Optimization and Engineering, 2 (2001), pp. 159–200.

[128] Witten I.H., E. Frank, *Data Mining: Practical Machine Learning Tools with Java Implementations*, Morgan Kaufmann, San Francisco, 1999.

[129] Wolsey L.A., *Integer Programming*, John Wiley & Sons, New York, 1998.

Vita

Tibérius Oliveira Bonates

October, 2007	PhD in Operations Research Rutgers University, Piscataway, NJ
August, 2007 –	Senior Associate Princeton Consultants, Inc., Princeton, NJ
2002–2007	Research/Teaching Assistant Rutgers University, Piscataway, NJ
Summer/2003	Summer Intern Dash Optimization Inc., Englewood Cliffs, NJ
2001–2002	Systems Analyst Eletrobrás Research Center, Brazil
1999–2001	MS in Systems Engineering Federal University of Rio de Janeiro, Brazil
1995–1998	BS in Computer Science State University of Ceará, Brazil

---

Publications

Bonates T.O., P.L. Hammer, A. Kogan, Maximum Patterns in Datasets, *Discrete Applied Mathematics*, in press, doi: 10.1016/j.dam.2007.06.004, 2007.

Alexe G., S. Alexe, T.O. Bonates, A. Kogan, Logical analysis of data – the vision of Peter L. Hammer, *Annals of Mathematics and Artificial Intelligence*, 49:265-312, 2007.

Bonates T.O., P.L. Hammer, An Algorithm for Optimal Training in Logical Analysis of Data, *RUTCOR Research Report RRR 22-2007*, 2007.

Bonates T.O., P.L. Hammer, A Branch-and-Bound Algorithm for a Family of Pseudo-Boolean Optimization Problems, *RUTCOR Research Report RRR 21-2007*, 2007.

Bonates T.O., P.L. Hammer, Pseudo-Boolean Regression, *RUTCOR Research Report RRR 3-2007*, 2007.

Bonates T.O., P.L. Hammer, Logical Analysis of Data: From Combinatorial Optimization to Medical Applications, *Annals of Operations Research*, 148:203–225, 2006.

G. Alexe, S. Alexe, D.E. Axelrod, T.O. Bonates, I.I. Lozina, M. Reiss, P.L. Hammer, Breast Cancer Prognosis by Combinatorial Analysis of Gene Expression Data, *Breast Cancer Research*, 8:R41, 2006.

Bonates T.O., N. Maculan, Performance Evaluation of a Family of Criss-Cross Algorithms for Linear Programming, *International Transactions on Operational Research*, 10(1), 53–64, 2003.