Optimization Techniques for Data Mining and Information Reconstruction

Candidate: Gianpiero Bianchi

Supervisor: Renato Bruni

Dissertation submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

in

Operations Research

"Sapienza" University of Rome

Contents

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Renato Bruni, for his constant support, guidance and encouragement. His competence and skill in Operations Research and related fields helped me in solving difficult mathematical questions arising from important practical problems, allowing me to successfully treat very large datasets and to produce a number of research papers.

I also thank the other faculty members of the DIAG Department of the “Sapienza” University of Rome, for their interest in my research work on the proposed topics.

I am grateful to my colleagues at Istat for their contribution to the development of Data Editing and Imputation systems and in particular to all the members of the MTO-D operating unit, that was charged to perform the several thousands of runs needed for the treatment of the Census data.

Finally, I take this opportunity to express my deepest gratitude to my family for their encouragement and constant support.

Preface

Data Mining problems are very important and frequent in several applicative fields. Extracting knowledge from large datasets is a demanding problem that requires indeed powerful computational resources. However, succeeding in these tasks depends not only on brute computational strength, but also, and often critically, on the mathematical quality of the models and of the algorithms underlying the solution procedures.

Linear optimization models with integer, binary or continuous variables possess a high expressive power, and are especially suitable to represent several Data Mining problems. Continuous variables can be used for example to represent a wide variety of real-valued quantities, possibly bounded to be non-negative. Discrete variables, on the other hand, allow the representation of options in the cases where indivisibility is required or where there is not a continuum of solutions, as it happens in many real problems characterized by a choice.

The presence of discrete variables generally makes the problem more difficult. The finite number of alternatives is in fact not a simplification, with respect to the case of a continuum of possibilities, because of the impressive number of such alternatives in real world problems. However, recent years have witnessed a huge amount of research in this field, and consequent decisive algorithmic improvements.

A number of new approaches to classical Data Mining problems are presented in this thesis. Some of them have been applied to Agricultural data, because the author is employed in the Italian National Institute of Statistics (Istat), in particular in the Census data service, and during the years of his PhD study the main issue of the service was treating data from the Italian Census of Agriculture 2010. This was the occasion for applying the proposed techniques to very important real-world problems. The contents of this thesis have been presented to some conferences and published in international journals.

This work is organized as follows.

Chapter 1 provides a brief overview of Data Mining and Information Reconstruction. After this, a general introduction to Integer and Mixed Integer Linear Programming is reported. Some introductory elements of Propositional Logic are also given, and the conversion of logical rules into linear inequalities is explained.

Chapter 2 presents an innovative classification procedure based on discretization and statistical analysis that allow to classify with a good degree of accuracy and in short times even when the available training sets are small. The proposed methodology uses, as much as possible, all the information extracted from the training set. The problems are formulated as mixed integer linear programming. The procedure has been tested on a test bed of public available datasets from UCI repository. Results are reported and discussed.

Chapter 3 presents an innovative methodology for solving the problem of balancing of Agricultural Census data by using a mixed integer linear model. This mathematical problem is called matrix balancing. The proposed procedure has been applied in the case of the Italian Census of Agriculture 2010 to restore data consistency when a total cultivation area, or a total number of livestock, is not equal to the sum of the detailed values representing the parts of the above totals.

Chapter 4 presents an innovative automatic procedure for assigning the correct cultivations to the area for which the farm declarations have been detected as unreliable. The proposed procedure is based on a discrete mathematical optimization model. This approach has been applied in the specific case of vineyard data reconstruction of the mentioned Agricultural Census.

Finally, Chapter 5 presents an innovative automatic procedure for finding contradictions into a set of rules, that operates by converting those rules into linear inequalities. The problem is thus converted into a linear programming problem. The proposed procedure has been tested in the case of a set of simulated rules for economical regulation.

Obviously, the techniques described in Chapters 3 and 4 by referring to agricultural data can be used to solve many other problems of different origin but sharing the same logical characteristics.

Roma, Italy, September 2013. GIANPIERO BIANCHI

Chapter 1

Introduction to Data Mining and Information Reconstruction

1.1 Generalities of Data Mining

Data mining is an interdisciplinary field of computer science, involving methods at the intersection of artificial intelligence, machine learning, statistics, optimization and database systems. The objective of data mining is to identify previously unknown, potentially useful, and understandable correlations and patterns in large datasets. Aside from the raw analysis step, it involves database and data management aspects, data preprocessing, model development, inference, study of metrics, complexity considerations, post-processing of discovered structures, visualization, updating. Consequently, data mining consists of more than collection and managing data, it also includes analysis and prediction.

The term Knowledge Discovery in Databases (KDD) is generally used to refer to the overall process of discovering useful knowledge from data, where data mining becomes a particular step in this process. Note, however, that there are communities in which the whole process in also called data mining. The KDD process consists of the following steps:

- **Data cleaning**: in this phase, noise and errors in data are removed from the dataset, and missing values are imputed. This may in many cases be a difficult problem, a specially in the case of very large datasets.

- **Data integration**: at this stage, multiple data sources, often heterogeneous, are combined in a common dataset. Relevant linkage problem may be involved.

- **Data selection**: at this step, the data relevant to the analysis are isolated and retrieved from the dataset, and are converted into forms appropriate for the mining procedure. The latter operation is also known as data transformation or data consolidation.

- **Data mining**: it is the crucial analysis step in which clever techniques are applied to extract the potentially useful patterns, i.e. the knowledge. Several types of techniques exist for doing this.

- **Pattern evaluation**: in this step, interesting patterns representing useful knowledge are identified, evaluated and, if required, visualized (knowledge representation).

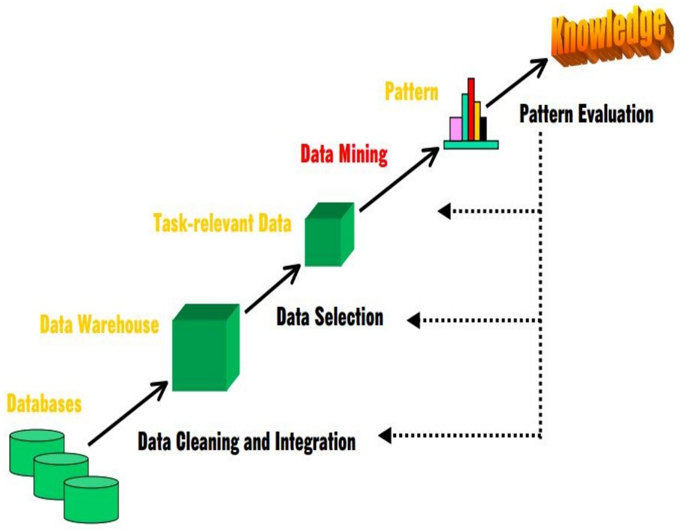


Figure 1.1: Knowledge Discovery (KDD) Process.

It is common to combine some of these steps together. For instance, data cleaning and data integration can be performed together as a pre-processing phase to generate a data warehouse. The KDD process can also work incrementally. Once the discovered knowledge is presented to the user, the evaluation measures can be enhanced, the mining can be further refined, new data can be selected or further transformed, or new data sources can be integrated, in order to get different, more appropriate results.

Data mining techniques are nowadays used in several applicative fields, for example:

- Database Analysis (Extraction of rules, Associations).
- Market Analysis (Customer profiling, Marketing).
- Risk Analysis (Finance planning, Investments).
- Fraud Detection (Credit cards, Food adulteration).
- Decision Support (Resource management, Allocation).
- Medical Analysis (Diagnosis, Management donors).
- Text mining (news group, email, documents) on the Web.
- Economic and Social Policy Analysis (Rule learning).
- Analysis of Rare Events.

1.2 Data Mining Tasks

Several types of data mining problem, or analysis tasks are typically encountered during a data mining project. Depending on the desired outcome, several data analysis techniques with different goals may be applied successively to achieve a desired result. For example, to determine which customers are likely to buy a new product, a business analyst may need first to use cluster analysis to segment the customer database, then apply regression analysis to predict buying behavior for each cluster. The data mining tasks typically fall into one of the general categories listed below.

Classification: assumes that the objects should be associable with classes, that are the elements of a discrete set of labels. The objective is to build classification models, i.e. classifiers, that assign the correct class to previously unseen and unlabeled objects. Classification approaches normally use a training set where all objects are already associated with known class labels. This constitute the source of information for the classification algorithm, which learns from the training set and builds the classifier(s). Classifier(s) are then used to classify new objects. Examples of classification methods are decision trees, k-nearest neighbors, neural networks, support vector machines, boolean approaches, bayesian approaches, logistic regression. Since a function is inferred from labeled training data, classification constitute a case of supervised learning.

Prediction: is very similar to classification. The difference is that in prediction, the class is not a discrete attribute but a continuous value. The goal of prediction is to find a numerical value for unseen objects by using a training set of objects already associated with similar values. This task is also known as regression, and if the prediction deals with time series data, then it is often called forecasting. Examples of prediction methods are statistical techniques used for regression analysis, decision trees, neural networks.

Segmentation or Clustering: separates the data into interesting and meaningful sub-groups or clusters, by simply using the object description and some similarity criterion. Automatic clustering techniques can detect previously unsuspected and hidden structures in data that allow segmentation. Clustering is a case of unsupervised learning, because, no training set is used. For this reason, it is sometimes called unsupervised classification.

Dependency Analysis: deals with finding a model that describes significant dependencies (or associations) between data items or events. Dependencies can be used to predict the value of an item given information on other data items. Dependency analysis may have connections with classification and prediction because the dependencies are implicitly used for the formulation of predictive models. Correlation analysis, regression analysis, association rules, case-based reasoning and visualization techniques are often applied.

Data Summarization: gives the user a compact description of the structure of the data, providing the property or the properties that describe each element of a dataset. This type of initial exploratory data analysis can help to understand the nature of the data and to find potential hypotheses for hidden information. Simple descriptive statistical and visualization techniques are often used.

1.3 Overview on Data Mining Classification Methods

In data mining, classification is one of the most important task, and in this thesis we will focus on that. As seen, the aim of the classification is to build a classifier based on a set of already classified cases (the training set). Then, the classifier is used to predict the class of new cases based on the values of their attributes. Commonly used methods for classification can be subdivided into the following groups. Note, however, that the following list is not meant to cover all possible classification techniques, since this task could be pursued by using the most heterogeneous approaches.

Decision Trees (DTs): are flowchart-like tree structures, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label [80]. The topmost node in a tree is the root node. During tree construction, attribute selection measures are used to select the attribute which best partitions the tuples into distinct classes. Three popular attribute selection measures are Information Gain, Gain Ratio, and Gini Index. When DTs are built, many of the branches may reflect noise or outliers in the training data. Tree pruning attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data.

k-Nearest Neighbor: is based on learning by analogy, that is by comparing a given test tuple with training tuples which are similar to it [38]. The training tuples are described by n attributes. Each tuple represents a point in an n-dimensional space. In this way, all of the training tuples are stored in an n-dimensional pattern space. When given an unknown tuple, a k-nearest neighbor (k-NN) classifier searches the pattern space for the k training tuples which are closest to the unknown tuple. These k training tuples are the k-nearest neighbors of the unknown tuple. "Closeness" is defined in terms of a distance metric, such as Euclidean distance. The Euclidean distance between two points or tuples $X_1 = (x_{\mathtt{11}}, x_{\mathtt{12}}, \ldots, x_{\mathtt{1n}})$ and $X_2 = (x_{\mathtt{21}}, x_{\mathtt{22}}, \ldots, x_{\mathtt{2n}})$ obtained from the following equation:

$$dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{\mathtt{1i}} - x_{\mathtt{2i}})^2} \tag{1.1}$$

The basic steps of the k-NN algorithm are:

- to compute the distances between the new sample and all previous samples that have already been classified into clusters;
- to sort the distances in increasing order and select the k samples with the smallest distance values;
- to apply the voting principle. A new sample will be added (classified) to the largest cluster out of k selected samples [68].

Neural Networks (NN): are those systems modeled based on the human brain working citerosenblatt. As the human brain consists of millions of neurons that are interconnected by synapses, a neural network is a set of connected input/output units in which each connection has a weight associated with it. The network learns in the learning phase by adjusting the weights so as to be able to predict the correct class label of the input. An artificial neural network consists of connected set of processing units. The connections have weights that determine how one unit will affect other. Two subsets of such units act as input nodes and output nodes, while remaining nodes constitute the hidden layer. By assigning activation to each of the input node, and allowing them to propagate through the hidden layer nodes to the output nodes, neural network performs a functional mapping from input values to output values [100].

Support Vector Machine (SVM): attempts to find a linear hyperplane separating the different classes. Since very often they are not directly linearly separable, this technique uses a nonlinear mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane. A hyperplane is a "decision boundary" separating the tuples of one class from another. With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using *support vectors* ("essential" training tuples) and *margins* (defined by the support vectors) [37]. SVM performs classification tasks by maximizing the margin separating both classes while minimizing the classification errors.

Logical and boolean approaches: these methodologies aim at extracting or discovering knowledge from data in logical form. The most known boolean approach is the Logical Analysis of Data (LAD) [18]. The key features of the LAD are the discovery of minimal sets of features necessary for explaining all observations and the detection of hidden patterns in the data capable of distinguishing observations describing positive outcome events from negative outcome events. Combinations of such patterns are used for developing general classification procedures. LAD methodology is based on discrete mathematics, and all data should be encoded into binary form by means of a process called “binarization”. This process consisting in the replacement of each numerical variable by binary indicator variables, each showing whether the value of the original variable is present or absent, or is above or below a certain level. This is done by using the training set for computing specific values for each field.

Naïve Bayes: are statistical classifiers. They can predict class membership probabilities [101]. Naïve Bayes (NB) probabilistic classifiers are commonly studied in machine learning. The basic idea in NB approaches is to use the joint probabilities of words and categories to estimate the probabilities of categories given a document. The naïve part of NB methods is the assumption of word independence, i.e. the conditional probability of a word given a category is assumed to be independent from the conditional probabilities of other words given that category. This assumption makes the computation of the NB classifiers far more efficient than the exponential complexity of non-naïve Bayes approaches because it does not use word combinations as predictors.

Boosting: is a meta-algorithm which can be viewed as a model averaging method and belongs to the class of ensemble techniques [86]. We first create a “weak” classifier, such that its accuracy on the training set is only slightly better than random guessing. A succession of models are built iteratively, each one being trained on a dataset in which it is assigned more weight to misclassified points by the previous model. Finally, all of the successive models are weighted according to their success and then the outputs are combined using voting thus creating a final model.

Logistic Regression (LR): measures the relationship between a categorical dependent variable and one or more independent variables, which are usually (but not necessarily) continuous, by using probability scores as the predicted values of the dependent variable [77]. Rather than choosing parameters that minimize the sum of squared errors (like in ordinary regression), estimation in logistic regression chooses parameters that maximize the likelihood of observing the sample values. Frequently LR is used to refer specifically to the problem in which the dependent variable is binary, that is, the number of available categories is two. LR is being used as a binary classification model. To measure the suitability of a binary regression model, one can classify both the actual value and the predicted value of each observation as either 0 or 1 [75]. The predicted value of an observation can be set equal to 1 if the estimated probability that the observation equals 1 is above 1/2, and set equal to 0 if the estimated probability is below 1/2.

Genetic Algorithms / Evolutionary Programming: are algorithmic optimization strategies that are inspired by the principles observed in natural evolution [92]. Of a collection of potential problem solutions that compete with each other, the best solutions are selected and combined with each other. In doing so, one expects that the overall goodness of the solution set will become better and better, similar to the process of evolution of a population of organisms. Genetic algorithms and evolutionary programming are used in data mining to formulate hypotheses about dependencies between variables, in the form of association rules or some other internal formalism. A disadvantage of Genetic algorithms is that the solutions are difficult to explain. Also, they do not provide interpretive statistical measures that enable the user to understand why the procedure arrived at a particular solution.

Fuzzy Sets: form a key methodology for representing and processing uncertainty [98]. Uncertainty arises in many forms in todays databases: imprecision, non-specificity, inconsistency, vagueness, etc. Fuzzy sets exploit uncertainty in an attempt to make system complexity manageable. As such, fuzzy sets constitute a powerful approach to deal not only with incomplete, noisy or imprecise data, but may also be helpful in developing uncertain models of the data that provide smarter and smoother performance than traditional systems. Fuzzy classification is the process of grouping elements into a fuzzy set whose membership function is defined by the truth value of a fuzzy propositional function [102]. A fuzzy propositional function is an expression containing one or more variables, such that, when values are assigned to these variables, the expression becomes a fuzzy proposition in the sense of [99].

1.4 Information Reconstruction Problems

In the past, for several fields, an automatic information processing has often been prevented by the scarcity of available data. Nowadays data are very abundant, but the problem that frequently arises is that such data may contain *errors*. This again makes an automatic processing not applicable, since the result is not reliable. Data correctness is indeed a crucial aspect of data quality. The relevant problems of error *detection* and *correction* should therefore be solved. When dealing with massive datasets, such problems are particularly difficult to formalize and very computationally demanding to solve. Since these problems have been studied in different fields of research they received different names. While in the field of database management they are called *data cleaning*, in the field of statistics they are called *data editing and imputation*, and the correction process is often subdivided into an *error localization* phase and a *data imputation* phase.

As customary for structured information, data are organized into records. The structure of records, called record scheme R, consists in a set of fields f_i, with $i = 1 \dots m$. A record instance r, also simply called record, consists in a set of values v_i, one for each field of the scheme.

$$R = f_1, \dots, f_m \qquad r = v_1, \dots, v_m \tag{1.2}$$

Each field f_i, with $i = 1 \dots m$, has its domain D_i, which is the set of every possible value for that field. Since we are dealing with errors, the domains include all values that can be found in data, even the erroneous ones. A record instance p is declared *correct* if and only if it respects a *set of rules* denoted by $R = r_1, \dots, r_u$. Each rule can be seen as a mathematical function r_k from the Cartesian product of all the domains to the Boolean set {0,1}, as follows.

$$\begin{array}{rccc} r_k : & D_1 \times \cdots \times D_m & \to & \{0,1\} \\ & p & \mapsto & 0,1 \end{array} \tag{1.3}$$

Rules are such that p is a correct record if and only if $r_k(p) = 1$ for all $k = 1 \dots u$.

Error Localization The problem of error localization is to find a set H of fields of minimum total cost such that a corrected record p_c can be obtained from an erroneous record p_e by changing (only and all) the values of H. Since H is a subset of the set of all fields $\{1, \dots, m\}$, this problem has a combinatorial optimization structure.

Data Imputation Imputation of actual values of H can then be performed in a deterministic or probabilistic way. This causes the minimum changes to erroneous data, but may have little respect for the original frequency distributions. A donor record p^d is a correct record which should be as similar as possible to the (unknown) original record p^o. This is obtained by selecting p^d being as close as possible to p^e, according to a suitable function δ giving a value v called the distance between p^e and p^d.

$$\begin{array}{rccl} \delta : & (D_1 \times \cdots \times D_m) \times (D_1 \times \cdots \times D_m) & \rightarrow & \mathbb{R}_+ \\ & (p^e, p^d) & \mapsto & v \end{array} \tag{1.4}$$

The problem of imputation through a donor is to find a set K of fields of minimum total cost such that p^c can be obtained from p^e by copying from the donor p^d (only and all) the values of K. This is generally recognized to cause low alteration of the original frequency distributions, although changes caused to erroneous data may be not minimum. This is deemed to produce a record which should be as close as possible to the original record (the record that would be present in absence of errors). The correction by means of a donor is also referred to as *data driven* approach. Because of its relevance and spread, the above problem has been extensively studied in a variety of scientific communities. Several different rules encoding and solution algorithm have been proposed (e.g. [8, 43, 78, 94]). A very well-known approach to the problem, which implies the generation of all rules logically implied by the initial set of rules, is due to Fellegi and Holt [47]. In practical case, however, such methods suffer from severe computational limitations [78, 94], with consequent heavy limitations on the number of rules and records that can be considered.

Several approaches to data correction problems use mathematical programming techniques. By requiring to change at least one of the values involved in each violated rule, a (mainly) set covering model of the error localization problem has been considered by many authors. Such model have been solved by means of cutting plane algorithms in [53] for the case of categorical data, and in [54, 81] for the case of continuous data. The above procedures has been adapted to the case of a mix of categorical and continuous data in [43], were a branch-and-bound approach to the problem is also considered. Such model, however, does not represent all the problem‘s features, in the sense that the solution to such model may fail to be a solution to the localization problem, the separation of the error localization phase from the imputation phase may originate artificial restrictions during the latter one, and computational limitations still hold. An automatic procedure for generic data correction by using a more effective discrete mathematical model of the problem is presented in [24, 26, 27]. This approach overcomes the computational limits of other techniques (see e.g. [8, 71, 94]), based on the Fellegi Holt approach, and allows to preserve, as far as it is possible, the marginal and joint distribution within the data.

1.5 Integer and Mixed Integer Linear Programming Models

Integer and mixed integer programming are subsets of the broader field of mathematical programming. Mathematical programming formulations use a set of *decision variables*, which represent actions or decisions that can be taken in the system being modeled. One then attempts to optimize (either in the minimization or maximization sense) an *objective*, that is a function of these variables which maps each possible set of decisions into a single score that assesses the quality of the solution. The limitations of the system are included as a set of *constraints*, which are usually stated by restricting functions of the decision variables to be equal to, not more than, or not less than, a certain numerical value. Another type of constraint can simply restrict the set of values to which a variable might be assigned.

Several applications involve decisions that are *discrete* (e.g., to which hospital an emergency patient should be assigned), while some other decisions are *continuous* in nature (e.g., determining the dosage of fluids to be administered to a patient). When a problem contains only continuous variables and linear objective and constraints, the problem is called *linear programming.* When on the contrary a problem contains only discrete variables and linear objective and constraints, the problem is called *integer linear programming.* Note that a parallelism can be traced between integer linear programming and combinatorial optimization with linear objective function [76]. When a problem contains both types of variables and linear objective and constraints, the problem is called *mixed integer linear programming.*

While discrete variables may appear easy to handle, the number of combinations of their values is usually huge, and so complete enumeration techniques have important implications on processing time. As the problem size increases, complete enumeration approaches are not computationally viable. Computer speedups, however impressive, are simply no match for exponential enumeration problems. Therefore, more efficient techniques are required to solve problems containing discrete variables. Those techniques do not explicitly examine every possible combination of discrete solutions, but instead examine a subset of possible solutions, and use optimization theory to prove that no other solution can be better than the best one found. This type of technique is referred to as implicit enumeration.

Linear Programming Linear Programming problems (LP, also called "linear programs") use a set of decision variables, which are the unknown quantities or decisions that are to be optimized. In the context of linear and mixed integer programming problems, the function that assesses the quality of the solution, called the "objective function", should be a linear function of the decision variables. An LP will either minimize or maximize the value of the objective function. Finally, the decisions that must be made are subject to certain requirements and restrictions of a system. We enforce these restrictions by including a set of constraints in the model. Each constraint requires that a linear function of the decision variables is either equal to, not less than, or not more than, a scalar value. A common condition simply states that each decision variable must be nonnegative. In fact, all linear programming problems can be transformed into an equivalent minimization problem with nonnegative variables and equality constraints [9].

A solution that satisfies all constraints is called a *feasible solution*. Feasible solutions that achieve the best objective function value (according to whether one is minimizing or maximizing) are called *optimal solutions*. Sometimes no feasible solution exists, and the optimization problem itself is called *infeasible*. On the other hand, some feasible LP problems have no optimal solution, because it is possible to achieve infinitely good objective function values with feasible solutions. Such problems are called *unbounded*.

Thus, suppose we denote $x_1, \dots, x_n$ to be our set of decision variables. Linear programming problems take on the form:

$$\begin{array}{ll} \text{min or max} & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \\ \text{subject to} & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \;\; (\leq, =, \text{or} \geq) \;\; b_1 \\ & a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \;\; (\leq, =, \text{or} \geq) \;\; b_2 \\ & \dots \\ & a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \;\; (\leq, =, \text{or} \geq) \;\; b_m \\ \\ & x_j \geq 0 \quad \forall j = 1, \dots, n \end{array} \tag{1.5}$$

Values $c_j, \forall j = 1, \dots, n$, are referred to as objective coefficients, and are often associated with the costs associated with their corresponding decisions in minimization problems, or the revenue generated from the corresponding decisions in maximization problems. The values $b_1, \dots, b_m$ are the right-hand-side values of the constraints, and often represent amounts of available resources (especially for $\leq$ constraints) or requirements (especially for $\geq$ constraints). The a_{ij}-values thus typically denote how much of resource/requirement i is consumed/satisfied by decision j. Note that nonlinear terms are not allowed in the model, prohibiting for instance the multiplication of two decision variables, the maximum of several variables, or the absolute value of a variable.

Any maximization (minimization) problem can be converted into a minimization (maximization) problem by multiplying the coefficients of the objective function by -1.

$$\max \sum_{j=1}^{n} c_j x_j = -\min \sum_{j=1}^{n} -c_j x_j$$

Moreover, each linear programming problem in generic form can be transformed into an equivalent problem in *canonical* form:

$$\begin{array}{lll} \min & \sum_{j=1}^{n} c_j x_n & \\ \textit{subject to} & \sum_{j=1}^{n} a_{\mathtt{ij}} x_j \geq b_i & \forall i = 1 \dots m \\ & x_j \geq 0 & \forall j = 1, \dots, n \end{array} \tag{1.6}$$

This canonical form can be expressed in a compact notation as follows.

$$\begin{array}{ll} \min & c^T x \\ & Ax \geq b \\ & x \in \mathbb{R}^{\mathrm{n}} \end{array} \tag{1.7}$$

where x represents the vector of variable (to be determined), c e b are vector of coefficients, A is a (known) matrix of coefficients. The inequalities $Ax \geq b$ are constraints which specify a convex politope over which the objective function is to be optimized. Linear programming problems can be converted into canonical form as follows:

- For each variable x_j, add the equality constraint $x_j = x_j^+ - x_j^-$ and the inequalities $x_j^+ \geq 0$ and $x_j^- \geq 0$.
- Replace any equality constraint $\sum_j a_{\mathtt{ij}} x_j = b_i$ with two inequality constraints $\sum_j a_{\mathtt{ij}} x_j \geq b_i$ and $\sum_j a_{\mathtt{ij}} x_j \leq b_i$.
- Replace any constraint $\sum_j a_{\mathtt{ij}} x_j \leq b_i$ with the equivalent constraint $\sum_j -a_{\mathtt{ij}} x_j \geq -b_i$.

Another useful format for linear programming problems is *standard* form, which is expressed as:

$$\begin{array}{ll} \min & c^T x \\ \textit{subject to} & Ax = b \\ \textit{and} & x \geq 0 \end{array} \tag{1.8}$$

Note that a LP not in standard form can be converted to standard form by eliminating inequalities by introducing slack and/or surplus variables and replacing variables that are not sign-constrained with the difference of two sign-constrained variables.

Mixed Integer Linear Programming When some of the variables are restricted to take integer values, the problem becomes a Mixed Integer Linear Programming one (MILP, also called "mixed integer linear programs"). When variables are restricted to take on either 0 or 1 values the term "integer" is replaced with "0-1" or "binary". All that was specified for the case of linear programming holds, *mutatis mutandis*, for the mixed integer case. Typically, modeling MILP requires the definition of a set of decision variables, that represent choices that must be optimized in the system, and the statement of an objective function and constraints (see also [93]).

It is very common, though, to recognize during model construction that the initial set of decision variables defined for the model are inadequate. Often, decision variables that seem to be implied consequences of other actions must also be defined. The addition of new variables after an unsuccessful attempt at formulating constraints and objectives is the "loop" in the process. The correct definition of decision variables can be especially complicated in modeling with integer variables. If one is allowed to use binary variables in a formulation, it is possible to represent yes-or-no decisions, enforce if-then statements, and even permit some sorts of nonlinearity in the model (which can be transformed to an equivalent mixed integer linear program).

Some common tips and tricks in modeling with integer variables are:

1. *Integrality of quantities.* In staffing and purchasing decisions, it is often impossible to take fractional actions. One cannot hire, for instance, 6.5 new staff members, or purchase 1.3 hospital beds. The most obvious use of integer variables thus arises in requesting integer amounts of quantities that can only be ordered in integer amounts. In general, the optimal solution of an integer program need not be a rounded-off version of an optimal solution to a linear program.

2. *If-then statements.* Consider two continuous (i.e., possibly fractional) variables, x and y, defined so that $0 \leq x \leq 10$ and $0 \leq y \leq 10$.

Suppose we wish to make a statement that if $x > 4$, then $y \leq 6$. On the surface, since no integer quantities are requested, it does not appear that integer variables will be necessary. However, the general form of linear programs as given in equations (1.5) does not permit if-then statements like the one above. Instead, if-then statements can be enforced with the aid of a binary variable, z. We wish to make $z = 1$ if $x > 4$ (note that we make no claims on z if $x \leq 4$). This can be accomplished by adding the constraint:

$$x \leq 4 + 6z \tag{1.9}$$

since the event that $x > 4$ implies that $z = 1$ (even if $z = 1$, the largest value for x is 10, which now makes a constraint of the form x is 10 unnecessary). If $z = 1$, then we must also require that $y \leq 6$. This is achieved by reducing the upper bound of 10 on y to 6 if z is equal to 1 as follows:

$$y \leq 10 + 4z \tag{1.10}$$

where once again, the bound constraint $y \leq 10$ may now be omitted. In general, suppose we wish to make the following statement: "if $q_1x_1 + \cdots + q_nx_n > Q$, then $r_1x_1 + \cdots + r_nx_n \leq R$". The following conditions should be included in the model:

$$q_1x_1 + \cdots + q_nx_n \leq q + M'z \tag{1.11}$$

$$r_1x_1 + \cdots + r_nx_n \leq M'' - (M'' - R)z \tag{1.12}$$

$$z \ binary \tag{1.13}$$

where M' and M'' are "sufficiently large" constants. These values should be just large enough to not add unintentional restrictions to the model. For instance, we are not attempting to place any hard restriction on the quantity $q_1x_1 + \cdots + q_nx_n$ (written conveniently as q^Tx in vector form). If $z = 1$, the upper bound on q^Tx is $Q + M'$, and hence M' must be large enough so that even if constraint (1.11) is removed from the model, q^Tx would still never be more than $Q + M'$. Likewise, if $z = 0$, a large enough value of M'' must be chosen in (1.12) such that r^Tx could never be more than M'' even without the restriction (1.12). It is worth noting that assigning arbitrarily large values for M' and M'' is not recommended.

3. *Enforce at least k out of p restrictions.* This situation is similar to if-then constraints in the way we model such restrictions. For a simple

example, suppose we have nonnegative variables $x_1, \ldots, x_n$, and wish to require that at least three of these variables take on values of 5 or more. Then we can define binary variables $z_1, \ldots, z_n$, such that if $z_j = 1$, then $x_j \geq 5, \forall j = 1, \ldots, n$. This simple if-then constraint can easily be modeled by employing the following constraints:

$$x_j \geq 5z_j \quad \forall j = 1, \ldots, n \tag{1.14}$$

Clearly, if $z_j = 1$, then $x_j \geq 5$. If $z_j = 0$, it is still possible for $x_j \geq 5$, but no such restrictions are enforced. It is necessary to guarantee that three variables take on values of 5 or more, and so the following "k-out-of-p" constraint is added:

$$z_1 + \cdots + z_n = 3 \tag{1.15}$$

Again, this constraint does not state that exactly three variables will be at least 5, but rather that at least three variables are guaranteed to be at least 5. This same trick can be used to enforce the condition that at least k out of p sets of constraints are satisfied, and so on, often by using M-values as introduced in the point on if-then constraints.

4. *Non linear product terms.* In some circumstances, nonlinear terms can be transformed into linear terms by the use of linear constraints. First, note that if x_j is a binary variable, then $x_j = x_j^q$ for any positive constant q. After that substitution is made, suppose that we have a nonlinear term of the form $x_1 \cdot x_2 \cdots x_k \cdot y$, where $x_1, \ldots, x_k$ are binary variables and $0 \leq y \leq u$ is another variable, either continuous or integer. That is, all but perhaps one of the terms is a binary variable. First, replace the nonlinear term with a single continuous variable, w. Using the if-then concept expressed above, note that if x_j equals zero for any $j \in \{1, ..., k\}$, then w equals zero as well. Also, note that w can never be more than the upper bound, u, on the y-variable. Hence, we obtain the constraints

$$w \leq ux_j \quad \forall j = 1, \ldots, k \tag{1.16}$$

Of course, to guarantee that w equals zero in case any x_j-variable equals to zero, we must also state a non-negativity constraint:

$$w \geq 0 \tag{1.17}$$

Now, suppose that all $x_1 = \cdots = x_k = 1$. In this case, it is necessary to add constraints that enforce the condition that $w = y$. Regardless of the x-variable values, w cannot be more than y, and so we state the constraint:

$$w \leq y \tag{1.18}$$

However, in order to get the constraint "$w \geq y$ if $x_1 = \cdots = x_k = 1$," we include the constraint:

$$w \geq u(x_1 + \cdots + x_k - k) + y \tag{1.19}$$

If each x-variable equals to 1, then (1.19) states that $w \geq y$, which along with (1.18) guarantees that $w = y$. On the other hand, if at least one $x_j = 0,\ j \in \{1, \ldots, k\}$, then the term $u(x_1 + \cdots + x_k - k)$ is not more than $-u$, and the right-hand-side of (1.19) is not positive; hence, (1.19) allows w to take on the correct value of zero (as would be enforced by (1.16) and (1.17)). As a final note, observe that even if y is an integer variable, we need not insist that w is an integer variable as well, since (1.16) - (1.19) guarantee that $w = x_1 \cdots x_k \cdot y$, which must be an integer given integer x- and y-values.

1.6 Branch&Bound and Branch&Cut Solution Techniques

Often, there are alternative ways of modeling optimization problems as MILP. There sometimes exist trade-offs in these different modeling approaches. Some models may be smaller (in terms of the number of constraints and variables required), but may be more difficult to solve than larger models. It is important to understand the basics of MILP solution algorithms in order to understand the key principles in MILP modeling. To illustrate the branch-and-bound process, we consider the following example MILP:

$$\begin{array}{ll} \min & 4x_1 + 6x_2 \\ s.t. & 2x_1 + 2x_2 \geq 5 \\ & x_1 - x_2 \leq 1 \\ & x_1, x_2 \geq 0 \textit{ and integer} \end{array} \tag{1.20}$$

A *relaxation* of an MILP is a problem such that (a) any solution to the MILP corresponds to a feasible solution to the relaxed problem, and (b) each solution to the MILP has an objective function value greater than or equal to that of the corresponding solution to the relaxed problem. The most commonly used relaxation for an MILP is its *LP relaxation*, which is identical to the MILP with the exception that variable integrality restrictions are eliminated. Clearly, any integer-feasible solution to the MILP is also a solution to its LP relaxation, with matching objective function values.

When describing the branch-and-bound algorithm for MILP, it is helpful to know how LP is solved. See [9, 63, 88, 95] for an explanation of linear programming theory and methodology. Graphically, Figure 1.2 illustrates the feasible region (set of all feasible solutions) to the LP relaxation of formulation (1.20). The point (1.75, 0.75), is the optimal solution to the LP relaxation, and has an objective function value of 11.5. In general, the optimal solution to the LP is not supposed to be unique, and so it is possible that different MILP solutions exist with an identical objective function to the optimal LP solution. The important result is that a lower bound on the optimal MILP solution is obtained from the LP relaxation. No solution to the MILP can be found with an objective function value less than 11.5.

Of course, the solution (1.75, 0.75) is not a feasible solution to (1.20). All feasible solutions have the trait that either $x_1 \leq 1$ or $x_1 \geq 2$. In fact, the problem (1.20) can be splitted into two subproblems: one in which $x_1 \leq 1$ (called region 1), and one in which $x_1 \geq 2$ (called region 2). All solutions to the original MILP are contained in exactly one of these two new subproblems. This process is called *branching*, and we could have also branched on x_2 instead, by requiring that either $x_2 \leq 0$ or $x_2 \geq 1$.

The feasible regions of the two new subproblems are depicted in Figure 1.3. When $x_1 \leq 1$, the optimal solution is (1, 1.5) with objective function value 13. When $x_1 \geq 2$, the optimal solution is (2, 1) with objective function value 14. In the $x_1 \leq 1$ region, the lower bound is 13. In the $x_1 \geq 2$ region, though, the best solution happens to be an integer solution. Therefore, the best integer solution in the $x_1 \geq 2$ region has an objective function value of 14; there is no need to further search that region. This region is said to be fathomed by integrality. We store the solution (2, 1), and call it *incumbent solution.* If no better solution is found, it will become our optimal solution.

At this point, there is one *active* region (or "active node" in the context of branch-and-bound trees), which is region 1. An active region is one that has not been branched on, and that must still be explored, because there is a possibility that it contains a solution better than the incumbent solution. The initial region is not active, because we have branched on it. Region 2 is not active since the best integer solution has been found in that region. Region 1, however, is still active and must be explored. The lower bound over this region is 13; thus, the optimal solution to the entire problem must have an objective function value somewhere between 13 and 14 (inclusive).

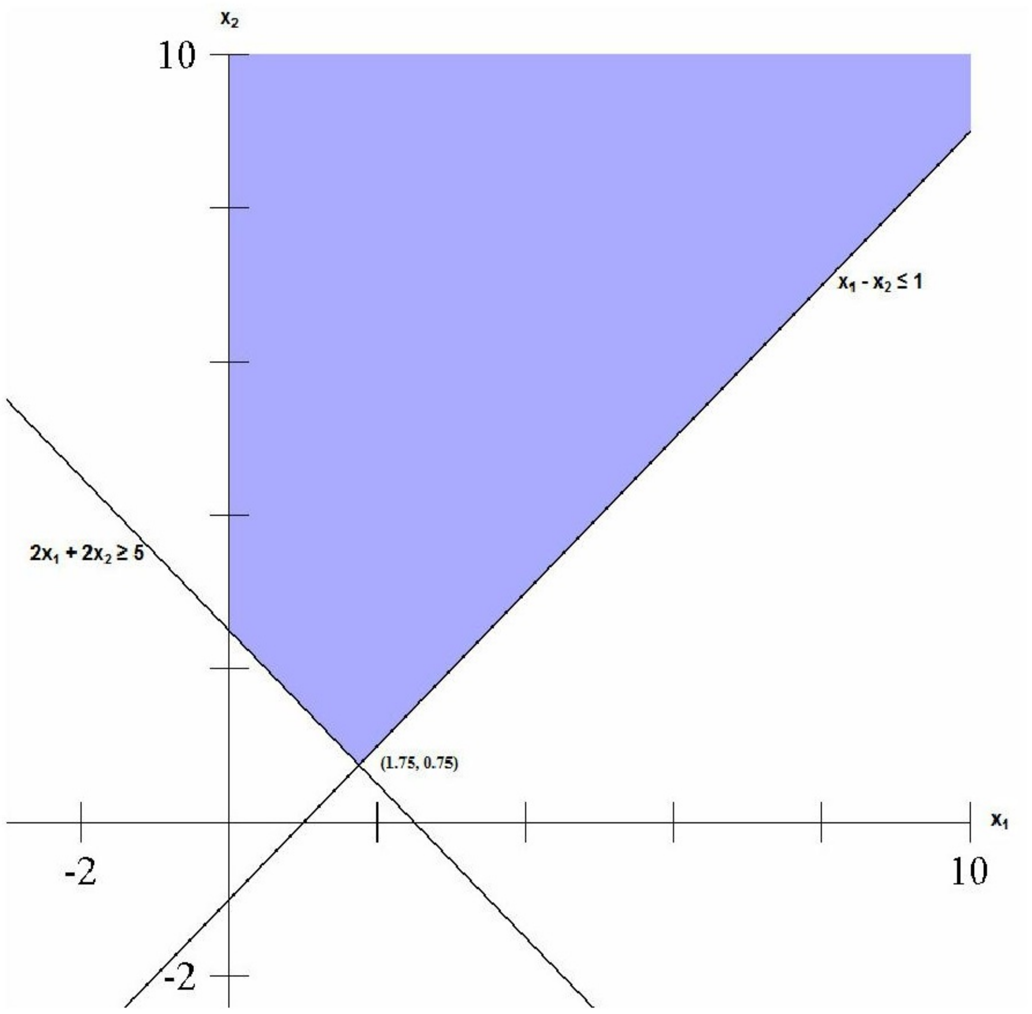


Figure 1.2: Feasible region of the LP relaxation.

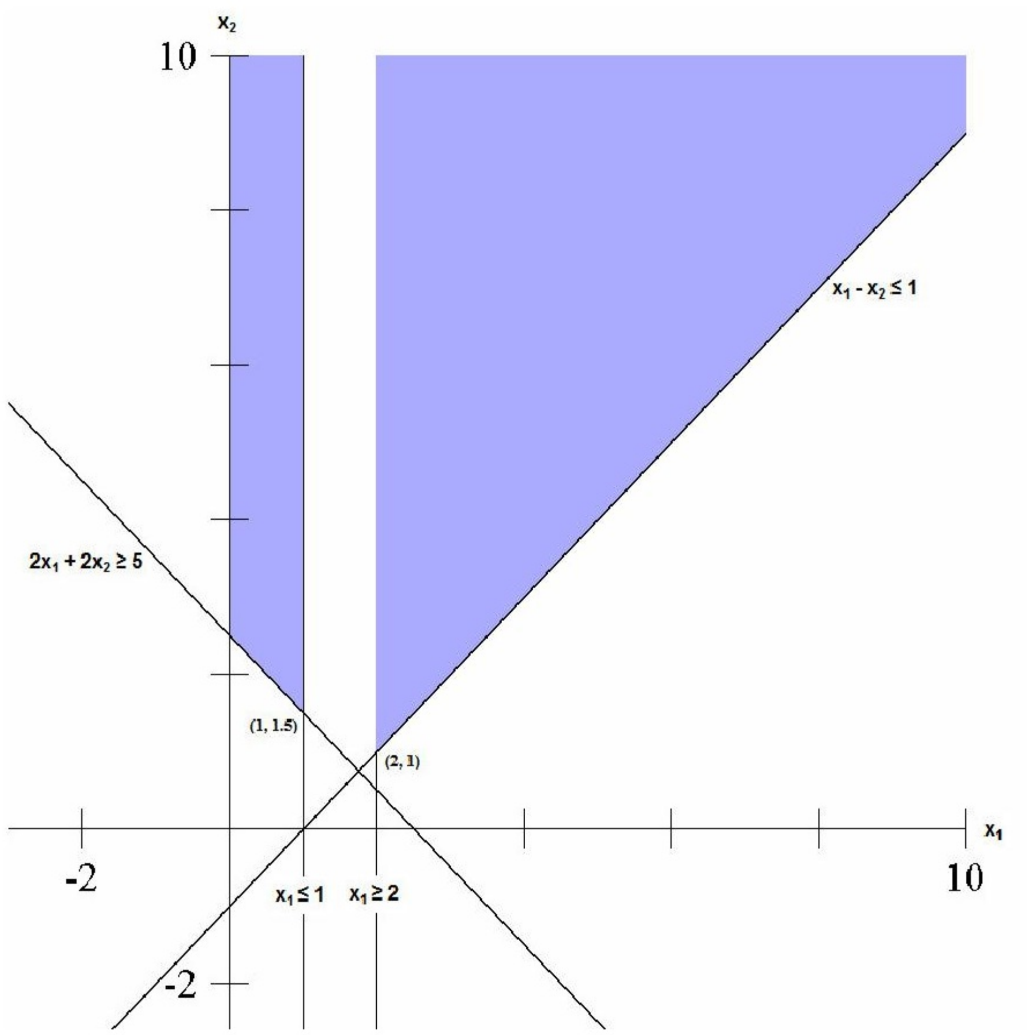


Figure 1.3: Feasible regions of the subproblems.

We recursively divide region 1, in which $x_1 \leq 1$. Since the optimal solution in this region was (1, 1.5), we branch by creating two new subproblems: one in which both $x_1 \leq 1$ and $x_2 \leq 1$ (called region 3), and one in which both $x_1 \leq 1$ and $x_2 \geq 2$ (called region 4). Once again, all integer solutions in region 1 are contained in either region 3 or region 4.

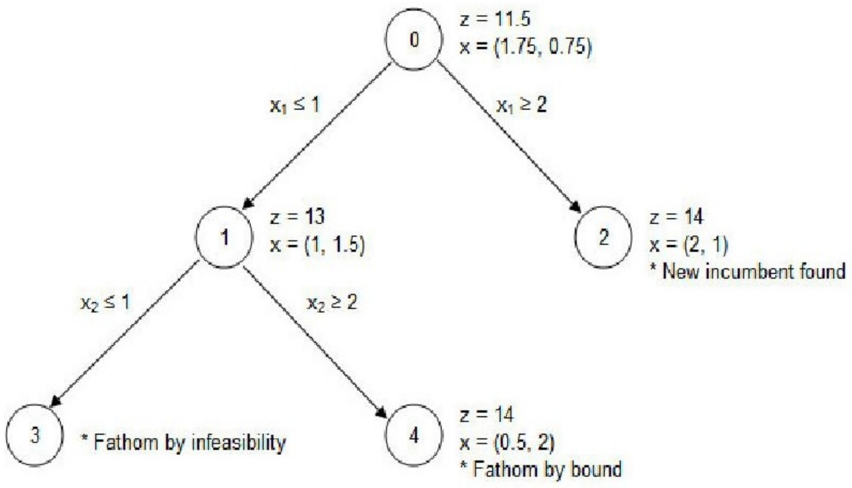


Figure 1.4: Branch-and-Bound tree.

However, note that region 3 is empty, because the stipulation that both x_1 and x_2 are no more than 1 makes it impossible to satisfy (1.20). There are therefore no integer solutions in this region either, and so we stop searching region 3. This region is said to be fathomed by infeasibility. The optimal solution to region 4‘s linear relaxation is (0.5, 2), with objective function value 14. However, our incumbent solution has an objective function value of 14. We have not found the best integer solution in region 4, but we know that the best solution in region 4 will not improve the incumbent solution we have found. Thus, we are not interested in any integer feasible solution in region 4, and we stop searching that region. (An alternative optimal integer solution can exist in that region, but we are not seeking to find all optimal solutions, just one.) Region 4 is said to be fathomed by bound.

Figure 1.4 depicts a tree representation of this search process, which is called the “branch-and-bound tree”. Each node of the tree represents a feasible region. Now, there are no more regions to be examined (no more active nodes), and the algorithm terminates with the incumbent solution, (2, 1), as an optimal solution.

A formal description of the branch-and-bound algorithm for minimization problems is given as follows.

> **Step 0** Set the incumbent objective $v = \infty$ (assuming that no initial feasible integer solution is available). Set the active node count $k = 1$ and denote the original problem as an "active" node. Go to Step 1.
>
> **Step 1** If $k = 0$, then stop: the incumbent solution is an optimal solution. (If there is no incumbent, i.e., $v = \infty$, then the original problem has no integer solution.) Else, if $k > 1$, go to Step 2.
>
> **Step 2** Choose any active node, and call it the "current" node. Solve the LP relaxation of the current node, and make it inactive. If there is no feasible solution, then go to Step 3. If the solution to the current node has objective value $z^* \geq v$, then go to Step 4. Else, if the solution is all integer (and $z^* < v$), then go to Step 5. Otherwise, go to Step 6.
>
> **Step 3** Fathom by infeasibility. Decrease k by 1 and return to Step 1.
>
> **Step 4** Fathom by bound. Decrease k by 1 and return to Step 1.
>
> **Step 5** Fathom by integrality. Replace the incumbent solution with the solution to the current node. Set $v = z^*$, decrease k by 1, and return to Step 1.
>
> **Step 6** Branch on the current node. Select any variable that is fractional in the LP solution to the current node. Denote this variable as x_s and denote its value in the optimal solution as f. Create two new active nodes: one by adding the constraint $x_s \leq |f|$ to the current node, and the other by adding $x_s \geq |f|$ to the current node. Add 1 to k (two new active nodes, minus one due to branching on the current node) and return to Step 1.

Note that in Step 0, a *heuristic procedure* could be executed to quickly obtain a good-quality solution to the MILP with no guarantees on its optimality. This solution would then become our initial incumbent solution, and could possibly help conserve branch-and-bound memory requirements by increasing the rate at which active nodes are fathomed in Step 4. In Step 2, we may have several choices of active nodes on which to branch, and in Step 6, we may have several choices on which variable to perform the branching operation. There has been much empirical research designed to establish good general rules to make these choices, and these rules are implemented in commercial solvers. However, for specific types of formulations, one can often improve the efficiency of the branch-and-bound algorithm by experimenting with node selection and variable branching rules.

The best-case scenario in solving a problem by branch-and-bound is that the original node yields an optimal LP solution that happens to be integer, and the algorithm terminates immediately. Indeed, in (1.20), by simply adding the constraint $x_1 + x_2 \geq 3$ and solve the LP relaxation, we would obtain the optimal solution (2, 1) immediately.

Thus, a classical way to reduce the presence of fractional solutions is to find *valid inequalities*, which do not cut off any integer solutions, but do cut off some fractional solutions. A *cutting plane* is a valid inequality that removes the optimal LP relaxation solution from the feasible region. The *cutting plane* method is an umbrella term for optimization methods which iteratively refine a feasible set or objective function by means of linear inequalities. Such procedures are generally used to find integer solutions to integer and mixed integer linear programming problems, and may be used also to solve other general optimization problems.

The theory of linear programming dictates that under mild assumptions (if the linear program has an optimal solution, and if the feasible region does not contain a line), one can always find a vertex that is optimal. The obtained optimal solution is tested for being integer. If it is not, there is guaranteed to exist a linear inequality that separates this LP relaxation solution from the *convex hull* of the set of integer solutions. Finding such an inequality is known as the *separation problem*, and such an inequality is a *cut*. A cut can be added to the relaxed linear program. Then, the current non-integer solution is no longer feasible to the relaxation. This process is repeated until an optimal integer solution is found.

In theory, MILP can be solved without branching either by (a) including enough valid inequalities before solving the LP relaxation, so that the LP relaxation provides an integer solution, or (b) looping between solving the LP relaxation, adding a cutting plane, and re-solving the LP relaxation, until the LP relaxation yields an integer solution.

However, using these approaches by themselves may suffer from numerical instability problems or require the solution of intractable problems. Therefore, the most effective implementations often use a combination of valid inequalities added a *priori* to the model, after which branch-and-bound is executed, with cutting planes periodically added to the nodes of the branch-and-bound tree. This approach is called “branch-and-cut”. Valid inequality and cutting-plane approaches can either be generic or problem-specific. Clearly, the second approach needs a problem-by-problem analysis, but can provide very efficient solution techniques. A large amount of research work on these subjects during the last 50 years lead to the development of many different cut types. Many classical cutting plane approaches are described in greater detail for instance in [76].

1.7 Rules based on Propositional Logic

Propositional logic, sometimes called sentential logic, may be viewed as a grammar for exploring the construction of complex sentences (propositions) from *atomic statements*, using the logical connectives. In prepositional logic we consider formulas (sentences, propositions) that are built up from atomic propositions that are unanalyzed. In a specific application, the meaning of these atomic propositions will be known.

The traditional (symbolic) approach to prepositional logic is based on a clear separation of the syntactical and semantical functions. The syntactics deals with the laws that govern the construction of logical formulas from the atomic propositions and with the structure of proofs. Semantics, on the other hand, is concerned with the interpretation and meaning associated with the syntactical objects. Prepositional calculus is based on purely syntactic and mechanical transformations of formulas leading to inference.

Propositional formulae are syntactically built by using an alphabet over the two following sets:

- The set of primary logic connectives $\{\neg, \vee, \wedge\}$, together with the brackets () to distinguish start and end of the field of a logic connective.
- The set of proposition symbols, such as $x_1, x_2, \ldots, x_n$.

The only significant sequences of the above symbols are the well-formed formulas (wffs). An inductive definition is the following:

- A propositional symbol x or its negation $\neg x$.
- Other wffs connected by binary logic connectives and surrounded, in case, by brackets.

Both propositional symbols and negated propositional symbols are called literals. Propositional symbols represent atomic (i.e. not divisible) propositions, sometimes called atoms. An example of wff is the following:

$$(\neg x_1 \vee (x_1 \wedge x_3)) \wedge ((\neg(x_2 \wedge x_1)) \vee x_3) \tag{1.21}$$

A formula is a wff if and only if there is no conflict in the definition of the fields of the connectives. Thus a string of atomic propositions and primitive connectives, punctuated with parentheses, can be recognized as a well-formed formula by a simple linear-time algorithm. We scan the string from left to right while checking to ensure that the parentheses are nested and that each field is associated with a single connective. Incidentally, in order to avoid the use of the awkward abbreviation "wffs", we will henceforth just call them propositions or formulas and assume they are well formed unless otherwise noted.

The calculus of propositional logic can be developed using only the three primary logic connectives above. However, it is often convenient to permit the use of certain additional connectives, such as $\Rightarrow$ which is called *implies*. They are essentially abbreviations that have equivalent formulas using only the primary connectives. In fact, if S_1 and S_2 are formulas, we have:

$$(S_1 \Rightarrow S_2) \quad \text{is equivalent to} \quad (\neg S_1 \vee S_2)$$

The elements of the set $B = \{T, F\}$ (or equivalently $\{1, 0\}$) are called truth values with T denoting $True$ and F denoting $False$. The truth or falsehood of a formula is a semantic interpretation that depends on the values of the atomic propositions and the structure of the formula. In order to examine the above, we need to establish a correspondence between propositional symbols and the elements of our Domain. This provides a truth assignment, which is the assignment of values T or F to all the atomic propositions. For this reason, propositions are often, although slightly improperly, called binary variables, but no connection with the concept of variable such as in the case of first-order logic exists.

To evaluate a formula we interpret the logic connectives, with their appropriate meaning of "not", "or", and "and". As an illustration, consider the formula (1.21). Let us start with an assignment of true (T) for all three atomic propositions x_1, x_2, x_3. At the next level, of subformulas, we have $\neg x_1$ evaluates to F, $(x_1 \wedge x_3)$ evaluates to T, $(x_2 \wedge x_1)$ evaluates to T, and x_3 is T. The third level has $(\neg x_1 \vee (x_1 \wedge x_3))$ evaluating to T and $((\neg(x_2 \wedge x_1)) \vee x_3)$ also evaluating to T. The entire formula is the "and" of two propositions both of which are true, leading to the conclusion that the formula evaluates to T. This process is simply the inductive application of the rules:

- S is T if and only if $\neg S$ is F.
- $(S_1 \vee S_2)$ is F if and only if both S_1 and S_2 are F.

- $(S_1 \wedge S_2)$ is T if and only if both S_1 and S_2 are T.

The assignment of truth values to atomic propositions and the evaluation of truth/falsehood of formulas is the essence of the semantics of this logic. We now introduce a variety of questions related to the truth or falsehood of propositions. The propositional logic uses symbolic valuations of propositions as either $True$ or $False$. Mathematical programming, however, works with numerical valuations.

By introducing suitable bounds on the variables, it is possible to restrict variables to only take values in the nonnegative integers or even just values of 0 and 1. This "boolean" restriction captures the semantics of propositional logic since the values of 0 and 1 may be naturally associated with $False$ and $True$. It is natural therefore to express the formulas with clauses represented by constraints and atomic propositions represented by 0-1 variables [31]. All the inequality constraints have to be satisfied simultaneously (in conjunction) by any feasible solution.

A positive atom (x_i) corresponds to a binary variable (x_i), and a negative atom $(\neg x_i)$ corresponds to the complement of a binary variable $(1 - x_i)$. Consider, for example, the single clause

$$x_2 \vee \neg x_3 \vee x_4$$

This clause can be converted in the following inequality over (0,1) variables.

$$x_2 + (1 - x_3) + x_4 \geq 1$$

Similarly the formula

$$(x_1) \wedge (x_2 \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee x_5)$$

is equivalent to the following system of linear inequalities:

$$\begin{aligned} x_1 \geq {} & 1 \\ x_2 + (1 - x_3) + x_4 \geq {} & 1 \\ (1 - x_1) + (1 - x_4) \geq {} & 1 \\ (1 - x_2) + x_3 + x_5 \geq {} & 1 \\ x_1, \ldots, x_5 \in {} & \{0, 1\} \end{aligned}$$

It is conventional in mathematical programming to clear all the constants to the right-hand side of a constraint. Thus a clause C_j is represented by $a_j x \geq b_j$, where for each i, a_{ji} is +1 if x_i is a positive literal in C_j, is -1 if $\neg x_i$ is a negative literal in C_j, and is 0 otherwise. Also, b_j equals $1 - n(C_j)$, where $n(C_j)$ is the number of negative literals in C_j. We shall refer to such inequalities as clausal. So the linear inequalities converted to clausal form are given by

$$\begin{aligned} x_1 &\geq 1 \\ x_2 - x_3 + x_4 &\geq 0 \\ -x_1 - x_4 &\geq -1 \\ -x_2 + x_3 + x_5 &\geq 0 \\ x_1, \ldots, x_5 &\in \{0, 1\} \end{aligned}$$

The rules used for checking data correctness (see Section 1.4) can be expressed by using propositional logic. Therefore, in general, the set of such checking rules is equivalent to the following system of linear inequalities

$$Ax \geq b, \qquad x \in \{0,1\}^n$$

where the inequalities of $Ax \geq b$ are clausal. Notice that A is a matrix over $\{0, \pm 1\}$, and each b_j equals 1 minus the number of -1's in row j of the matrix A. Further details on the conversion into linear inequalities are given in Chapter 5.

Chapter 2

Classification based on Discretization and MILP

2.1 The Problem of Data Classification

Given a set of *data* grouped into *classes*, the problem of predicting which class new data should receive is called *classification* problem. The first set of data is called *training set*, while the set of new data is called *test set*. Instances from the training set should have the same structure and the same nature than those of the test set. Classification is of fundamental significance in the fields of data analysis and data mining, and several important practical decision problems are actually classification problems.

Many approaches to this problem have been proposed, based on different considerations and data models. Established ones include: Neural Networks, Support Vector Machines, k-Nearest Neighbors, Bayesian approaches, Decision Trees, Logistic regression, Boolean approaches (see for references [60, 62, 70, 73, 80, 91, 100]). Each approach has several variants, and algorithms can also be designed by mixing approaches. Specific approaches may fit to specific classification contexts, but one approach that is considered quite effective for many practical applications are Support Vector Machines (SVM). They are based on finding a separating hyperplane that maximizes the margin between the extreme training data of opposite classes, possibly after a mapping in an higher dimensional space, see also [32, 37]. However, no single algorithm is currently able of providing the best performance on all datasets, and this seems to be inevitable [96]. Predicting which algorithm will perform best on a specific dataset has become a learning task on its own, belonging to the area called meta-learning [67]. Therefore, techniques based on the aggregation of a set of different (and hopefully complementary) classifiers have been investigated. They are called *ensemble techniques*, and they include Boosting [86, 50] and Bagging [22]. Roughly speaking, those techniques generate many weak learners and combine their outputs in order to obtain a classification that is both accurate and robust. Ensemble techniques based on tree classifiers, such as Random Forest [23], are often able to provide a good performance. A Random Forest, in particular, is a combination of many tree classifiers, each of which grown on a subset of data randomly sampled independently, and the classification output is a combination (i.e. the mode) of the outputs of the individual trees.

On the other hand, one interesting Boolean approach is the *Logical Analysis of Data* (LAD), developed since the 1990's by Hammer *et al.* (see [18, 19, 21, 29, 40]). It is inspired by the mental processes that a human being applies when learning from examples a *classifier*. In the LAD methodology, data should be encoded into binary form by means of a discretization process called *binarization*. This is done by using the training set for computing specific values for each field, called *cut-points* in the case of numerical fields, that split each field into *binary attributes*. Discretization is also adopted in other classification methodologies, such as decision trees, and several ways for selecting cut-points exists, such as entropy based ones (see e.g. [46, 70]). The selected binary attributes, constituting a *support set*, are then combined for generating logical rules called *patterns*. Patterns are used to classify each unclassified record, on the basis of the sign of a weighted sum of the patterns *activated* by that record. A main feature of such approach is that patterns constitute also a *compact description* of the data, i.e. a generally understandable set of rules that describes the classification rationale (see e.g. [39]). LAD methodology is closely related to decision trees and nearest neighbor methods, and constitutes an extension of those two approaches, as shown in [21].

In this Chapter the following original enhancements to the LAD methodology are proposed. First, the idea of evaluating the quality of each cut-point for numerical fields and of each binary attribute for numerical fields, and a criterion for doing so. Such quality values are computed by using information extracted from the training set, and are taken into account for improving the selection of the support set. The support set selection can therefore be modeled as a *weighted set covering* problem, and also as a *binary knapsack* problem (see e.g. [76, 88]). In a related work, Boros *et al.* [20] consider the problem of finding essential attributes in binary data, which again reduces to finding a small support set with a good separation power. They give alternative formulations of such problem and propose three types of heuristic algorithm for solving them. An analysis of the smallest support set selection problem within the framework of the probably approximately correct learning theory, and algorithms for its solution, is also in [2].

Moreover, the classification of the test set is not given here simply on the basis of the sign of the weighted sum of activated patterns, but by comparing that weighted sum to a suitable *classification threshold*. Indeed, we propose to compute both the values of pattern weights and the value of classification threshold in order to minimize errors, by solving a mixed integer linear programming problem. The objective of minimizing errors is pursued by (*i*) minimizing classification errors on records of the training set and by (*ii*) reproducing in the test set the class distribution of the training set. Pattern weights and classification threshold are in fact parameters for the classification procedure, and, in our opinion, this should allow obtaining the best choice of these parameters for the specific dataset, overcoming the parameter tuning or guessing phase that always represents a difficult and questionable step. The proposed approach, based on statistical considerations on the data, allows to classify with a good degree of accuracy and in short times even when the available training sets are small as in the case of rare events and uncontrollable events.

The known LAD procedure is recalled in Section 2.2. In this Chapter, the "standard" procedure, as described in [19], has been mainly considered, although other variants have been investigated in the literature ([58]). The original contributions of this work begin with Section 2.3, which explains motivations and possible criteria for evaluating the quality of cut-points. In particular, we derive procedures for dealing with cut-points on continuous fields having *normal* (Gaussian) distribution, on discrete fields having *binomial* (Bernoulli) distribution, or on general numerical fields having unknown distribution. This latter approach is used also for qualitative, or categorical, fields. The support set selection problem is then reformulated as weighted set covering and as binary knapsack in Section 2.4. After that, patterns are generated, and computation of pattern weights and classification threshold are described in Section 2.5. Results of the proposed procedure on publicly available datasets of the UCI repository [49] are analyzed and compared to those of the standard LAD methodology, and also to those of the Support Vector Machines (SVM [37, 32]) methodology in its implementation LIBSVM [33], which is currently deemed to be one of the more effective classifiers, in Section 2.6.

2.2 Classifying with the LAD Methodology

Data used for classification are organized in records. The structure of records, called *record scheme* R, consists in a set of fields f_i, with $i = 1 \dots m$. A *record instance* r, also simply called *record*, consists in a set of values v_i, one for each field. Fields are essentially of two types: *quantitative*, or *numerical*, and *qualitative*, or *categorical*. A record r is *classified* if it is assigned to an element of a set of possible classes C. In many cases, C has only two elements, and we speak of *binary classification*. This case will be considered hereinafter. Note, however, that the proposed procedure, *mutatis mutandis*, could also be used for the case of multiple classes. A positive record instance is denoted by r^+, a negative one by r^-.

For classifying, a *training set* S of classified records is given. Denote by S^+ the set of its positive records and by S^- the set of its negative ones. Sets S^+ and S^- constitute our source of information for learning a *classifier*. A set of records used for evaluating the performance of the learned classifier is called *test set* T. The *real* classification of each record $t \in T$ should be known. We compare the classification of T given by the learned classifier, also called *predicted* classification, to the real classification of T: the differences are the classification errors of our classifier. A positive training record is denoted by s^+, a negative one by s^-. A positive test record is denoted by t^+, a negative one by t^-. Very roughly speaking, the larger S is, the more information it contains, the more accurate our learned classifier will be, even if clearly there are several aspects involved. However, in many important applications, the availability of training records is scarce, and a classification methodology able to be accurate using *small* training sets would be very useful.

LAD methodology begins with encoding all fields into binary form. This process, called *binarization*, converts each (non-binary) field f_i into a set of binary *attributes* a_i^j, with $j = 1 \dots n_i$. The total number of binary attributes is $n = \sum_{i=1}^{m} n_i$. Note that the term "attribute" is not used here as a synonym of "field". A binarized record scheme R_b is therefore a set of binary attributes a_i^j, and a binarized record instance r_b is a set of binary values $b_i^j \in \{0, 1\}$ for those attributes.

$$\begin{aligned} R_b &= \{a_1^1, \dots, a_1^{n_1}, \dots, a_m^1, \dots, a_m^{n_m}\} \\ r_b &= \{b_1^1, \dots, b_1^{n_1}, \dots, b_m^1, \dots, b_m^{n_m}\} \end{aligned}$$

For each qualitative fields f_i, all values can simply be encoded by means of a logarithmic number of binary attributes a_i^j, so that n_i binary attributes can binarize a quantitative field having up to 2^{n_i} different values. For each numerical field f_i, on the contrary, we introduce n_i thresholds called *cut-*

points $\alpha_i^1, \dots, \alpha_i^{n_i} \in \mathbb{R}$, and the binarization of a value v_i is obtained by considering whether v_i lies above or below each α_i^j. Cut-points α_i^j should be set at values representing some kind of watershed, or being otherwise relevant, for the analyzed phenomenon. Generally, α_i^j are placed in the middle of specific couples of data values v_i' and v_i'':

$$\alpha_i^j = (v_i' + v_i'')/2$$

This can be done for each couple v_i' and v_i'' belonging to records from opposite classes and adjacent on f_i, i.e.:

- $v_i' \in r^+ \in S^+$ and $v_i'' \in r^- \in S^-$, or *vice versa*;
- no other training record has a value v_i''' such that $v_i' < v_i''' < v_i''$ if $v_i' < v_i''$, or *vice versa* if $v_i'' < v_i'$.

Cut-points α_i^j are then used for binarizing each numerical field f_i into the binary attributes a_i^j (also called level variables). The values b_i^j of such a_i^j are

$$b_i^j = \begin{cases} 1 & \text{if } v_i \geq \alpha_i^j \\ 0 & \text{if } v_i < \alpha_i^j \end{cases}$$

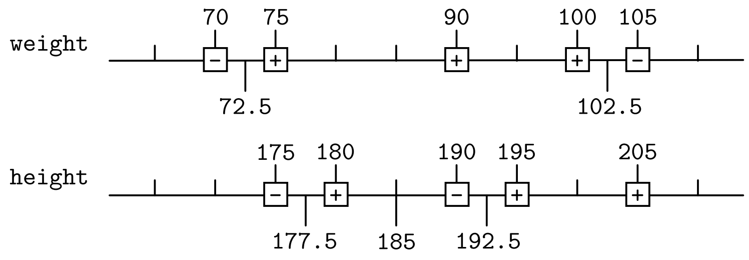
Note that α_i^j is not required to belong to D_i, but only to be comparable, by means of $\geq$ and $<$, to all values $v_i \in D_i$.

Example 2.1 Consider the following training set of records representing persons having fields `weight` (in Kg.) and `height` (in cm.), and a positive [respectively negative] classifications meaning "is [resp. is not] a professional basketball player".

	`weight`	`height`	`pro.bask.player.?`
S^+	`90`	`195`	`yes`
	`100`	`205`	`yes`
	`75`	`180`	`yes`
S^-	`105`	`190`	`no`
	`70`	`175`	`no`

We now plot values belonging to positive [resp. negative] records by using a framed + [resp. −]. Cut-points obtainable from this set S are $\alpha^1_{\texttt{weight}}$=`72.5`,

$\alpha^2_{\texttt{weight}}$=`102.5`, $\alpha^1_{\texttt{height}}$=`177.5`, $\alpha^2_{\texttt{height}}$=`185`, $\alpha^3_{\texttt{height}}$=`192.5`. Corresponding binary attributes obtainable are $a^1_{\texttt{weight}}$, meaning: `weight` $\geq$ `72.5` Kg., $a^2_{\texttt{weight}}$, meaning: `weight` $\geq$ `102.5` Kg., $a^1_{\texttt{height}}$, meaning: `height` $\geq$ `177.5` cm., $a^2_{\texttt{height}}$, meaning: `height` $\geq$ `185` cm., $a^3_{\texttt{height}}$, meaning: `height` $\geq$ `192.5`.



A set of binary attributes $\{a^j_i\}$ used for binarizing a dataset S is called *support set* U. A support set is exactly separating if no pair of positive and negative records of S have the same binary encoding. A single data-set may have several possible exactly separating support sets. Since the number of binary attributes obtainable in practical problems is often very large, and many of them may be not needed to explain the analyzed phenomenon, we are interested in selecting a small (or even the smallest) exactly separating support set. By using a binary variable x^j_i for each a^j_i, such that

$$x^j_i = \begin{cases} 1 & \text{if } a^j_i \text{ is retained in the support set} \\ 0 & \text{if } a^j_i \text{ is excluded from the support set} \end{cases}$$

the integer programming problem (2.1) should be solved. For every pair of positive and negative records s^+, s^- we define $I(s^+_b, s^-_b)$ to be the set of couples of indices (i, j) where the binary representations of s^+ and s^- differ, except, under special conditions [19], for the indices that involve monotone values. This problem has a peculiar mathematical form called *set covering* [76, 88]: the objective (sum of all the binary variables) minimizes the cardinality of the support set; the constraints (sums of binary variables ≥ 1) impose retaining at least one binary attributes for each set of them producing different binarizations for any pair of positive and negative records.

$$\begin{cases} \min \sum_{i=1}^{m} \sum_{j=1}^{n_i} x_i^j \\ \text{s.t.} \sum_{(i,j)\in I(s_b^+, s_b^-)} x_i^j \geq 1 \quad \forall I(s_b^+, s_b^-),\ s^+ \in S^+,\ s^- \in S^- \\ x_i^j \in \{0,1\} \end{cases} \tag{2.1}$$

Note that this selection does not have the aim of improving the classification power, and actually "the smaller the chosen support set, the less information we keep, and, therefore, the less classification power we may have" [19]. Instead, it is necessary for reducing the computational complexity of the remaining part of the procedure, which may otherwise become impracticable. Indeed, a non-optimal solution to such problem would not necessarily worsen the classification power [19, 20]. Since different support sets correspond to different alternative binarizations, hence to actually different binarized record, the *support set selection* constitutes a key point.

Example 2.2. Continuing Example 2.1, by solving to optimality the above set covering problem (2.1), we have the alternative support sets $U_1 = \{a^2_{\texttt{weight}}, a^1_{\texttt{height}}\}$ and $U_2 = \{a^1_{\texttt{weight}}\, a^2_{\texttt{weight}}\}$. Moreover, an approximate solution is $U_3 = \{a^1_{\texttt{weight}}, a^2_{\texttt{weight}}, a^1_{\texttt{height}}, \}$. The corresponding alternative binarizations of the records in S are:

	U_1		U_2		U_3		
	$b^2_{\texttt{we.}}$	$b^1_{\texttt{he.}}$	$b^1_{\texttt{we.}}$	$b^2_{\texttt{we.}}$	$b^1_{\texttt{we.}}$	$b^2_{\texttt{we.}}$	$b^1_{\texttt{he.}}$
	0	1	1	0	1	0	1
S^+	0	1	1	0	1	0	1
	0	1	1	0	1	0	1
S^-	1	1	1	1	1	1	1
	0	0	0	0	0	0	0

The selected support set U is then used to create patterns. A *pattern* P is a conjunction ($\wedge$) of literals, which are binary attributes $a_i^j \in U$ or negated binary attributes $\neg a_i^j$. Given a binarized record r_b, that is a set of binary values $\{b_i^j\}$ for the above binary attributes, each literal of P receives a value: $b_i^j \in \{0,1\}$ for literal a_i^j; $(1-b_i^j) \in \{0,1\}$ for literal $\neg a_i^j$. We have that $P = 1$ if all literals of P are 1, $P = 0$ otherwise. We say that a pattern P *covers* a record r if the set of values $r_b = \{b_i^j\}$ makes $P = 1$. A *positive* pattern P^+ is a pattern covering at least one positive record r^+ but no negative ones. A negative pattern P^- is defined symmetrically. Patterns admit an interpretation as rules governing the analyzed phenomenon. We write $P(r)$ for denoting the value of pattern P applied to record r :

$$P(r) = \begin{cases} 1 & \text{if } P \text{ covers } r \\ 0 & \text{if } P \text{ does not cover } r \end{cases}$$

Example 2.3. By continuing Example 2.2, a positive pattern obtained using the support set U_1 is $P_1 = \neg a^2_{\texttt{weight}} \wedge a^1_{\texttt{height}}$. This means `weight` $<$ `102.5` Kg. and `height` $\geq$ `177.5` cm. Recall P_1 is a pattern if $P_1(s^+) = 1$ for at least some $s^+ \in S$ and $P_1(s^-) = 0$ for all $s^- \in S$. Indeed, we have $P_1(s^+) = 1$ for all $s^+ \in S$ and $P_1(s^-) = 0$ for all $s^- \in S$. Another pattern, obtained using support set U_3, is $P_2 = a^1_{\texttt{weight}} \wedge \neg a^2_{\texttt{weight}} \wedge a^1_{\texttt{height}}$. P_2 appears to be even more appropriate than P_1, since it means "one is a professional basketball player if has a medium weight (`weight` $\geq$ `72.5` Kg. and `weight` $<$ `102.5` Kg.) and height above a certain value (`height` $\geq$ `177.5` cm.)". $P_2(s^+) = 1$ for all $s^+ \in S$ and $P_2(s^-) = 0$ for all $s^- \in S$.

Positive patterns can be generated by means of two types of approaches: top-down, i.e. by removing one by one literals from the conjunction of literals covering a single positive record until no negative records are covered, or bottom-up, i.e. by conjoining one by one single literals until obtaining a conjunction covering only positive records. Negative patterns can be generated symmetrically. Also the number of generated patterns may be too large, so a pattern selection step can be performed. This is done in [19] by solving another set covering problem, whose solution gives the set of the indices H^+ of selected positive patterns and that of the indices H^- of selected negative patterns, with $H = H^+ \cup H^-$. Weights w_h are now assigned to all patterns in H, with $w_h \geq 0$ for $h \in H^+$ and $w_h \leq 0$ for $h \in H^-$, by using criteria described in [19]. We skip detail here since we will discuss this again and propose a new approach in Section 2.5. Finally, each new record r is classified according to the positive or negative value of the following weighted sum, called *discriminant* and denoted by $\Delta(r)$.

$$\Delta(r) = \sum_{h \in H^+} w_h P_h(r) + \sum_{h \in H^-} w_h P_h(r) = \sum_{h \in H} w_h P_h(r)$$

2.3 Evaluation of Binary Attributes

We remarked that selecting a small support set is computationally necessary, but that excluding attributes means losing information. Therefore, we propose to evaluate the *quality* (the separating power) of each attribute and to perform such a selection taking into account this evaluation. In the following Figure 2.1, we give three examples of numerical fields (a,b,c). In each case we draw (in the area above the horizontal line) "qualitative" distributions densities of a consistent number of values of positive and negative records, and report (on the same line) a smaller sample of positive and negative records having the above distributions. Very intuitively, cut-points obtainable in case a) are the worst ones (they do not appear very useful for separating the two classes), while the cut-point of case c) is the best one (it has a good "separating power"). Moreover, the different cut-points of case b) do not have the same quality. We now need to formalize this evaluation. The approach is based on how α_i^j splits the data, i.e. it divides the two classes, *even if* the real classification step will be performed by using patterns. Different estimators could of course be designed, however results show that using the proposed one is able to improve accuracy with respect to the standard LAD procedure (that do not use estimators).

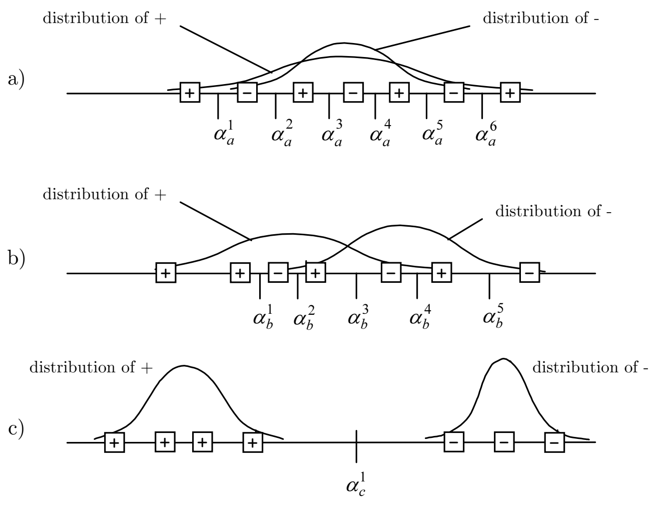

distribution of +
distribution of -
a)
α_a^1 α_a^2 α_a^3 α_a^4 α_a^5 α_a^6
b)
α_b^1 α_b^2 α_b^3 α_b^4 α_b^5
c)
α_c^1

Figure 2.1: Examples of cut-points in different conditions.

Given a single cut-point α_i^j and a record r, denote by $+$ the fact that r is actually positive, and by $-$ the opposite situation. Moreover, denote by $_{class}+(\alpha_i^j)$ the fact that r is classified as positive by α_i^j, i.e. stays on the positive side of cut-point α_i^j, and by $_{class}-(\alpha_i^j)$ the fact that r is in the opposite situation. Given a generic set of records N, let A_+ be the set of the records which are $_{class}+(\alpha_i^j)$, and A_- be the set of records which are $_{class}-(\alpha_i^j)$. Denote instead by N^+ and N^- the (possibly unknown) real positive and negative sets. Errors occur when a negative record is classified as positive, and vice versa. The first kind of errors, called *false positive*, are $N_- \cap A_+$; the second kind of errors, called *false negative*, are $N_+ \cap A_-$. The *confusion matrix* is given in Table 2.1 below.

		Real	
		$+$	$-$
Predicted by α_i^j	$+$	$N_+ \cap A_+$	$N_- \cap A_+$
	$-$	$N_+ \cap A_-$	$N_- \cap A_-$

Table 2.1: Confusion matrix.

Since the described support set selection problem is a non-trivial decision problem, it seems reasonable to model it as a binary linear programming problem. For doing so, we need to use a criterion for evaluating the quality of each binary attribute such that the overall quality value of a set of binary attributes can be given by the sum of their individual quality values. We obtain this as follows. A basic measure of the accuracy of the positive classification obtained from α_i^j can be the probability of producing a true positive divided by the probability of producing a false positive.

$$o^+(\alpha_i^j) = \frac{Pr(+ \cap {}_{class}+(\alpha_i^j))}{Pr(- \cap {}_{class}+(\alpha_i^j))}$$

A similar measure can evaluate the accuracy of the negative classification obtained from α_i^j.

$$o^-(\alpha_i^j) = \frac{Pr(- \cap {}_{class}-(\alpha_i^j))}{Pr(+ \cap {}_{class}-(\alpha_i^j))}$$

Clearly, $o^+(\alpha_i^j) \in [0, +\infty)$ and $o^-(\alpha_i^j) \in [0, +\infty)$. The higher the value, the better positive [resp. negative] classification α_i^j provides. In order to have a complete evaluation of α_i^j, we consider the product $o^+(\alpha_i^j) \times o^-(\alpha_i^j) \in [0, +\infty)$.

Moreover, rather than the numerical value of such evaluation, we are interested in the relative differences among the values obtained for the different cut-points. Therefore, we can sum 1 to such product, obtaining a value in $[1, +\infty)$.

$$1 + \frac{Pr(+ \cap \textit{class} + (\alpha_i^j))}{Pr(- \cap \textit{class} + (\alpha_i^j))} \cdot \frac{Pr(- \cap \textit{class} - (\alpha_i^j))}{Pr(+ \cap \textit{class} - (\alpha_i^j))}$$

Denote now by A the set of couples of indices (i, j) of a generic set of cut-points: $\{\alpha_i^j \ : (i, j) \in A\}$. The overall accuracy of a classification using the cut-points in A is now related to the product of the individual evaluations:

$$\prod_{(i,j)\in A} \left[1 + \frac{Pr(+ \cap \textit{class} + (\alpha_i^j))}{Pr(- \cap \textit{class} + (\alpha_i^j))} \cdot \frac{Pr(- \cap \textit{class} - (\alpha_i^j))}{Pr(+ \cap \textit{class} - (\alpha_i^j))}\right]$$

As noted above, more than the numerical values, we are interested in producing, for each set of cut-points, values that can be compared. Therefore, we can apply a scale conversion and take the logarithm of the above value.

$$\ln \prod_{(i,j)\in A} \left[1 + \frac{Pr(+ \cap \textit{class} + (\alpha_i^j))}{Pr(- \cap \textit{class} + (\alpha_i^j))} \cdot \frac{Pr(- \cap \textit{class} - (\alpha_i^j))}{Pr(+ \cap \textit{class} - (\alpha_i^j))}\right]$$

This allows to convert it in a sum, as requested, obtaining:

$$\sum_{(i,j)\in A} \ln \left[1 + \frac{Pr(+ \cap \textit{class} + (\alpha_i^j))}{Pr(- \cap \textit{class} + (\alpha_i^j))} \cdot \frac{Pr(- \cap \textit{class} - (\alpha_i^j))}{Pr(+ \cap \textit{class} - (\alpha_i^j))}\right]$$

In conclusion, the quality q_i^j of a single cut-point α_i^j can be evaluated as follows (so that the quality of a set of cut-points results in the sum of their individual quality values).

$$q_i^j = \ln \left[1 + \frac{Pr(+ \cap \textit{class} + (\alpha_i^j))}{Pr(- \cap \textit{class} + (\alpha_i^j))} \cdot \frac{Pr(- \cap \textit{class} - (\alpha_i^j))}{Pr(+ \cap \textit{class} - (\alpha_i^j))}\right]$$

Clearly, $q_i^j \in [0, +\infty)$. Computing the above probabilities by counting instances (and denoting by $|\cdot|$ the cardinality of a set), we have:

$$q_i^j = \ln \left[1 + \frac{\frac{|N_+ \cap A_+|}{|N^+|}}{\frac{|N_- \cap A_+|}{|N^+|}} \cdot \frac{\frac{|N_- \cap A_-|}{|N^-|}}{\frac{|N_+ \cap A_-|}{|N^-|}}\right] = \ln \left[1 + \frac{|N_+ \cap A_+|}{|N_- \cap A_+|} \cdot \frac{|N_- \cap A_-|}{|N_+ \cap A_-|}\right]$$

However, this evaluation needs the correct classification $\{N^+, N^-\}$ of the dataset N. We obviously prefer an *a priori* quality evaluation, i.e. computable by knowing only the correct classification of the training set S. We can do this by using a non-parametric method for fields having unknown distribution, and a parametric one for fields having known distribution.

In the case of fields having unknown distribution, q_i^j is simply obtained by considering the training set S instead of the generic N, while for each cut-point α_i^j sets A_+ and A_- are clearly known (they respectively are the sets or records that are $\mathit{class} + (\alpha_i^j)$ and $\mathit{class} - (\alpha_i^j)$). Now, the quality of each attribute a_i^j over a numerical field f_i is that of its corresponding cut-point α_i^j, that is the defined q_i^j.

In the case of fields where the hypothesis of a known distribution is satisfactory, their positive and negative density functions can be computed using the training set S, and the above quantities $|N_+ \cap A_+|$, etc. can be evaluated by using such density functions. In other words, we just know data from the training set S, but we may infer where other data will be, and compute how useful α_i^j would be for all of them. In particular, for any continuous-valued field f_i, we make the hypothesis of a *normal* (Gaussian) distribution. Such distribution can indeed model the majority of real-world values, as a consequence of the central limit theorem [48]. Denote now by m_{i+} the *mean value* that positive records have for f_i and by σ_{i+} their (population) *standard deviation* (defined as $\sqrt{\frac{\sum_{s\in S^+}(v_i^s - m_{i+})^2}{|S^+|}}$), denote by m_{i-} and σ_{i-} the same quantities for the negative records, and suppose w.l.o.g. that cut-point α_i^j represents a transition from $-$ to $+$. By computing the above parameters from the training set S, our evaluation of quality q_i^j becomes:

$$q_i^j = \ln\left[1 + \frac{\int_{\alpha_i^j}^{+\infty} \frac{1}{\sqrt{2\pi(\sigma_{i+})^2}}\, e^{-\frac{(t-m_{i+})^2}{2(\sigma_{i+})^2}}\, dt}{\int_{\alpha_i^j}^{+\infty} \frac{1}{\sqrt{2\pi(\sigma_{i-})^2}}\, e^{-\frac{(t-m_{i-})^2}{2(\sigma_{i-})^2}}\, dt} \cdot \frac{\int_{-\infty}^{\alpha_i^j} \frac{1}{\sqrt{2\pi(\sigma_{i-})^2}}\, e^{-\frac{(t-m_{i-})^2}{2(\sigma_{i-})^2}}\, dt}{\int_{-\infty}^{\alpha_i^j} \frac{1}{\sqrt{2\pi(\sigma_{i+})^2}}\, e^{-\frac{(t-m_{i+})^2}{2(\sigma_{i+})^2}}\, dt}\right]$$

In case of a discrete-valued field f_i, on the contrary, we make the hypothesis of *binomial* (Bernoulli) distribution. This should indeed describe many discrete real-world quantities [48]. Denote now by m_{i+} and M_{i+} the *minimum* and the *maximum* values on field i for positive records, and by m_{i-} and M_{i-} the same quantities for the negative records. Denote also by

$n_{i+} = M_{i+} - m_{i+}$ the *number* of possible positive values for f_i, and by p_+ the characteristic positive *probability of success* (also called Bernoulli probability parameter, estimated as $|S^+|/n_{i+}$). Denote by $n_{i-} = M_{i-} - m_{i-}$ and by p_- the same quantities for negative records. Suppose, again, that α_i^j is a transition from $-$ to $+$. By computing the above parameters from S, our evaluation of quality q_i^j becomes now:

$$q_i^j = \ln\left[1+\frac{\sum_{t=\alpha_i^j-m_{i+}}^{n_{i+}} \binom{n_{i+}}{t}(p_{i+})^t(1-p_{i+})^{n_{i+}-t}}{\sum_{t=\alpha_i^j-m_{i+}}^{n_{i+}} \binom{n_{i-}}{t}(p_{i-})^t(1-p_{i-})^{n_{i-}-t}} \cdot \frac{\sum_{t=0}^{\alpha_i^j-m_{i-}-1} \binom{n_{i-}}{t}(p_{i-})^t(1-p_{i-})^{n_{i-}-t}}{\sum_{t=0}^{\alpha_i^j-m_{i-}-1} \binom{n_{i+}}{t}(p_{i+})^t(1-p_{i+})^{n_{i+}-t}}\right]$$

Moreover, we modify the above q_i^j in order to reduce possible *overfitting* and to avoid selecting attributes in an unbalanced manner (e.g. all from the same fields).

We penalize each attribute a_i^j corresponding to a cut-point α_i^j originated by a few isolated points of one class laying near many points of the opposite class. More precisely, we set two thresholds ν_1 and ν_2 and put $q_i^j := q_i^j/2$ for each a_i^j such that: *i*) a number of training records $\leq \nu_1$ lie on one side of α_i^j, and *ii*) a number of training records $\geq \nu_2$ (of the opposite class) lie on the other side of α_i^j.

We also penalize the binary attributes over a field f_i from which other binary attributes have already been selected. Clearly, this can be applied only during a sequential solution (see Section 2.4) of the support set selection problem. More precisely, each time an attribute from f_i is selected, we put $q_i^j := q_i^j/2$ for each still unselected attributes of f_i.

Note, finally, that for fields having a considerable overlapping between the two classes, cut-points cannot be generated when inverting the class, because almost every region of the field contains both classes. On the contrary, they are generated when inverting the *class predominance*, i.e. when passing from a region with positive predominance to one with negative predominance and *vice versa*. By considering the fraction of negative records in the training $\frac{|S^-|}{|S|}$, a region has positive predominance when its percentage of negative records is $\leq g\frac{|S^-|}{|S|}\%$. Values for g was set at 70.

2.4 Reformulations of the Support Set Selection Problem

When the quality value of each attribute have been computed, the exactly separating support set selection problem can be modeled as follows. We would like minimizing a weighted sum (and not only the number) of selected attributes, where the weights are the reciprocal $1/q_i^j$ of the quality q_i^j, while selecting at least an attribute for each of the above defined sets $I(r_b^+, r_b^-)$. Note that $1/q_i^j$ can be viewed as a measure of the *uselessness* of a_i^j. By using the binary variables x_i^j already introduced in Section 2.2, the following *weighted* set covering problem should be solved, using the non-negative weights $1/q_i^j$.

$$\begin{cases} \min \displaystyle\sum_{i=1}^{m} \sum_{j=1}^{n_i} \frac{1}{q_i^j}\; x_i^j \\ \text{s.t.} \displaystyle\sum_{(i,j)\in I(r_b^+, r_b^-)} x_i^j \geq 1 \quad \forall I(s_b^+, s_b^-),\ s^+ \in S^+,\ s^- \in S^- \\ x_i^j \in \{0,1\} \end{cases} \tag{2.2}$$

This formulation takes now into account the individual qualities of the attributes. One may observe that this would discard attributes that have a poor isolated effect but may have important effect when combined with other attributes during the pattern generation step. However, a selection is necessary for the computational viability of the entire procedure, and the proposed approach aims at discarding the attributes that appear more suitable to be discarded.

Moreover, such weighted set covering formulation (2.2) has strong computational advantages on a non-weighted one (2.1). Although still NP-hard [76], solution algorithms become considerably faster when the model variables receive different weight coefficients in the objective function. Depending on the size of the model and on available computational time, such weighted set covering problem may be either solved to optimality or by searching for an approximate solution. In the former case, it is guaranteed that the pattern generation step is performed by using a set of attributes U which is a minimal set for which no positive and negative records have the same binary encoding. In the latter case, if the approximate solution is feasible but non-optimal, it is not guaranteed that U is minimal, i.e. it may exist also a proper subset $U' \subset U$ such that no positive and negative records have the same binary encoding. This could have the effect of increasing the computational burden of the pattern generation step, but not of worsening the classification accuracy. If, on the contrary, the approximate solution is (slightly) infeasible, U is such that (few) positive and negative records have the same binary encoding. This could have the effect of accelerating the pattern generation step, but of decreasing the classification accuracy.

In the cases when the above model still remains computationally demanding, e.g. for large datasets, or when there are very tight time requirements, e.g. real time applications, the support set selection problem can be modeled differently. The computational burden added to the whole classification procedure could be evaluated by retaining each single attribute a_i^j, and call it its *size* s_i^j. When no specific evaluations can be done, those sizes could be set all at 1. Moreover, a maximum affordable computational burden b can be established, for instance on the basis of the time available for preforming the classification, or of the available computing hardware, etc. Note that such requirement may be independent from the minimum size of an exactly separating support set: the available resources are limited, and, if they allow obtaining an exactly separating support set, the better, but this cannot be imposed. By using the same binary variables x_i^j, the support set selection problem can now be modeled as *binary knapsack* problem:

$$\begin{cases} \max \displaystyle\sum_{i=1}^{m} \sum_{j=1}^{n_i} \; q_i^j \; x_i^j \\ \text{s.t.} \; \displaystyle\sum_{i=1}^{m} \sum_{j=1}^{n_i} \; s_i^j \; x_i^j \leq b \\ x_i^j \in \{0,1\} \end{cases} \tag{2.3}$$

Solving the above model is again NP-hard [76], so it may in general be as hard as (2.2). However, in the case when all sizes s_i^j are 1, it becomes polynomially solvable by just sorting the q_i^j values and by taking the best b of them. Note that, in this case, attributes can be selected sequentially, and the weights be modified after each single attribute selection, in order to incorporate penalty techniques such as the one described in the end of previous Section. The above selections are performed independently on positive and negative attribute, so as to find the set U^+ of selected positive attributes and the set U^- of selected negative ones.

2.5 Pattern Generation and Use

A pattern P is a logic function of attributes a_i^j, typically a conjunction of literals, which are binary attributes $a_i^j \in U$ or negated binary attributes $\neg a_i^j$. Given a binarized record r_b, that is a set of binary values $\{b_i^j\}$, each literal of a generic pattern P receives a value, and so P itself receives a value, denoted by $P(r) \in \{0, 1\}$ (see also Section 2.2). We say that a pattern P *covers* a record r if $P(r) = 1$, and that pattern P is *activated* by r. In the standard LAD procedure, a *positive* pattern P^+ has to cover at least one positive record r^+ but no negative ones, and a *negative* pattern P^- is defined symmetrically. This, however, can lead to improper pattern generation in the case of noisy or otherwise difficult datasets. In the procedure presented in this Chapter, patterns are produced in a bottom-up fashion. For obtaining a positive pattern, we generate every possible logic conjunction grouping up to p literals, using one after another all literals obtainable from U^+. When a conjunction $\bar{P}$ verifies the following *coverage conditions*

- $\bar{P}$ covers at least η^c positive records of S
- $\bar{P}$ covers at most η^e negative records of S

we save $\bar{P}$ as a pattern and never repeat $\bar{P}$ as part of other conjunctions. A negative pattern is generated symmetrically. This simple generalization of the original covering condition can generate patterns being more robust, since patterns not covering any element of the opposite class may be rare in the mentioned cases. Thresholds η^c and η^e should be tuned on the specific dataset. However, in general, we use $\eta^c \geq 2\eta^e$, with η^c proportional to data density and η^e proportional to the noise contained in the data.

In order to produce a complete classifier, each test record should be covered by at least one pattern. However, generating bottom-up patterns could leave uncovered some regions of the data space. Therefore, an additional pattern generation step is required, in a top-down fashion: patterns describing single training records covering the still uncovered regions of the data space are taken, and then simplified, by iteratively removing literals from them in all possible ways, until they satisfy other two coverage thresholds η_a^c and η_a^e. Their meaning is respectively analogous to η^c and η^e, but the requirements should in general be more relaxed.

Now, unclassified records can be classified by examining which patterns cover them. Clearly, a record activating only positive patterns should be classified as positive, and vice versa. A positive pattern is indeed a (partial) compact description of positive records. However, in the majority of the cases, unclassified records activate both positive and negative patterns. Some kind of "voting" criterion is needed. LAD methodology uses a weighted sum of the activated patterns, also called discriminant Δ. The weight given to pattern P_h in this sum is denoted by w_h, with $h \in H$. The discriminant must be compared to a *classification threshold* δ for classifying record r:

$$\sum_{h \in H} w_h P_h(r) = \Delta(r) > \delta \quad \Leftrightarrow \quad r \in R^+$$
$$\sum_{h \in H} w_h P_h(s) = \Delta(r) \leq \delta \quad \Leftrightarrow \quad r \in R^-$$

Using patterns can also be seen as *Boosting* [50, 86]: learning weak classifiers (the patterns) and combining them by means of weights in order to obtain a strong classifier. Evaluating the mentioned weights, i.e. the "power" of each pattern in the classification process, can be done using different criteria. A first criterion can be based on the coverage values of each pattern, as in the original LAD [19]. For instance, if u_h is the number of positive records covered by a positive pattern P_h, its weight is $w_h = u_h^2$, and symmetrically for a negative one (squared pattern coverage). However, in the case of patterns covering overlapping sets of records, this criterion could be misleading.

A more ambitious criterion is assigning weights and classification threshold in order to minimize classification errors. Since the only classification errors that can be detected at this stage are those on the training set, we try to minimize them. We assume, in absence of further information, that this would produce a similar effect on the test set, being such data of the same nature of the training set. For doing so, denote by $c(r)$ the value 1 if r is a positive record, 0 otherwise (the real classification). Clearly, $c(s)$ is known for each training record $s \in S$.

On the other hand, applying the learned classifier on the training set S produces a predicted classification for each $s \in S$. By comparing real and predicted classification of a training record $s \in S$, we obtain $e_s \in \{0, 1\}$, that we call classification error for the training record s.

$$e_s = \begin{cases} 1 & \text{if } \ \Delta(s^+) \leq \delta \ \text{ or } \ \Delta(s^-) > \delta \\ 0 & \text{otherwise} \end{cases}$$

Values e_s clearly depend on all elements of the procedure: cut-point selection, pattern generation, pattern weights, classification threshold, so they are not easily expressible. However, when knowing whether each pattern P_h, with $h \in H$, covers or not each record $s \in S$, the above e_s are simple functions of pattern weights $\{w_h\}$ and classification threshold δ. Therefore, given the set of generated pattern $\{P_h\}$, we compute the coverages $P_h(s) \in \{0,1\}$ for all $h \in H$ and $s \in S$, obtaining a $|H| \times |S|$ matrix PS having binary elements d_{hs}:

$$PS = [d_{hs}] \text{ with } d_{hs} = P_h(s)$$

The same can be done for each pattern P_h, with $h \in H$ and each test record $t \in T$, obtaining a $|H| \times |T|$ matrix PT having binary elements d_{ht}:

$$PT = [d_{ht}] \text{ with } d_{ht} = P_h(t)$$

On the other hand, for each test record $t \in T$, we only know (at this stage) the classification given by the learned classifier, which is again function of $\{w_h\}$ and δ.

$$c_t = \begin{cases} 1 & \text{if } \sum_{h \in H} w_h P_h(r) > \delta \\ 0 & \text{if } \sum_{h \in H} w_h P_h(r) \leq \delta \end{cases}$$

Moreover, we want to learn from the training set the *class distribution*, that is the fraction of positive $\frac{|S^+|}{|S|}$ (or of negative $\frac{|S^-|}{|S|}$) records contained in the training set (clearly, given one of the two, the other is also fixed). We therefore introduce a value, called *tolerance* and denoted by γ, measuring the "difference" from the class distribution of the training set and that of the test set. Hence, we have a bi-objective: minimizing the number of errors on the training set and minimizing the tolerance γ. By introducing a scalarization parameter $G > 0$, our objective becomes:

$$\min \sum_{s \in S} e_s + G\gamma$$

A reasonable choice for G is $|S|/10$, so that the second term of the objective cannot override the first one (whose theoretical maximum is $|S|$, but with typical values between $0.01|S|$ and $0.4|S|$). The description of constraints is following.

We need to impose that the classification error e_s is 1 for each record $s \in S$ such that the classification that s would receive using $\{w_h\}$ and δ does not match its real class $c(s)$.

$$\sum_{h \in H} w_h d_{hs} - \delta \leq M(c(s) + e_s) \qquad \forall s \in S \tag{2.4}$$

$$\sum_{h \in H} w_h d_{hs} - \delta > -M(1 - c(s) + e_s) \qquad \forall s \in S \tag{2.5}$$

M is a positive constant greater than any possible value of the first member (see also [93]). The mathematical behavior of those constraints is the following. When $\sum_{h \in H} w_h d_{hs} - \delta > 0$, record s is predicted positive. In this case, the second member of (2.4) must be $\geq$ than a positive number, so it must be positive, while the second member of (2.5) must be $<$ than the same positive number, so it can be either 0 or negative: if $c(s) = 0$ (= the prediction is an error), e_s is forced to be 1 by the (2.4), while it is free for the (2.5); if $c(s) = 1$ (= the prediction is not an error), e_s is free for both constraints.

On the other hand, when $\sum_{h \in H} w_h d_{hs} - \delta \leq 0$, record s is predicted negative. In this case, the second member of (2.4) must be $\geq$ than a number ≤ 0, so it can be either 0 or positive, while the second member of (2.5) must be $<$ than the same number, so it must be negative: if $c(s) = 1$ (= the prediction is an error), e_s is forced to be 1 by the (2.5), while it is free for the (2.4); if $c(s) = 0$ (= the prediction is not an error), e_s is free for both constraints. Note that, when e_s is free for both constraints, the minimization of the objective will make it 0. In order to have a closed feasible region, (2.5) is converted into $\geq$ by introducing a small $\epsilon > 0$

$$\sum_{h \in H} w_h d_{hs} - \delta \geq -M(1 - c(s) + e_s) + \epsilon \qquad \forall s \in S$$

In order to evaluate the class distribution that $\{w_h\}$ and δ would produce in T, we need to compute the predicted classification of its records. We therefore need constraints connecting values $\{w_h\}$ and δ to the class c_t that would be predicted for each record $t \in T$. The machinery is similar to that of the above analyzed constraints, but note that we do not use at all the real class of the test records, that must obviously remain unknown during classification process.

$$\sum_{h \in H} w_h d_{ht} - \delta \leq M c_t \qquad \forall t \in T \tag{2.6}$$

$$\sum_{h \in H} w_h d_{ht} - \delta > -M(1 - c_t) \qquad \forall t \in T \tag{2.7}$$

Constraint (2.7) is converted into $\geq$ by using again a small $\epsilon > 0$

$$\sum_{h \in H} w_h d_{ht} - \delta \geq -M(1 - c_t) + \epsilon \qquad \forall t \in T$$

Finally, we need constraints imposing that $\{w_h\}$ and δ reproduce in T the class distribution of S, so $|T^+|$ should be as similar as possible to $|S^+| \cdot \dfrac{|T|}{|S|}$, and connecting the difference to the introduced γ.

$$\sum_{t \in T} c_t \leq \sum_{s \in S} c(s) \cdot \frac{|T|}{|S|} + |T|\gamma + \rho \tag{2.8}$$

$$\sum_{t \in T} c_t \geq \sum_{s \in S} c(s) \cdot \frac{|T|}{|S|} - |T|\gamma - \rho \tag{2.9}$$

Note that, when we need to classify just one or a few records, obtaining the same class distribution of S could be impossible. For example, if we need to classify two records, and the fraction of positive $\frac{|S^+|}{|S|}$ is 0.2, targeting at that class distribution for the two records is clearly useless. Hence, (2.8-2.9) should have no effect when T is very small. This is obtained by using value ρ, that, when set for instance at 3, relaxes constraints (2.8-2.9) of 3 units. For large $|T|$ this relaxation is negligible (so it is not considered in the tests of Section 2.6), while for small $|T|$ the problem gradually reduces to minimizing only the classification error on S.

The overall mixed integer linear model for finding optimal pattern weights w_h and classification threshold δ is now the following. Weight values are bounded by a value W, in order to avoid giving excessive importance to any single pattern, since that could be a source of overfitting.

$$\begin{cases} \min \sum_{s \in S} e_s + G\gamma \\ \\ \sum_{h \in H} w_h d_{hs} - \delta \leq M(c(s) + e_s) & \forall s \in S \\ \sum_{h \in H} w_h d_{hs} - \delta \geq -M(1 - c(s) + e_s) + \epsilon & \forall s \in S \\ \sum_{h \in H} w_h d_{ht} - \delta \leq M c_t & \forall t \in T \\ \sum_{h \in H} w_h d_{ht} - \delta \geq -M(1 - c_t) + \epsilon & \forall t \in T \\ \sum_{t \in T} c_t \leq \sum_{s \in S} c(s) \cdot \frac{|T|}{|S|} + |T|\gamma + \rho \\ \sum_{t \in T} c_t \geq \sum_{s \in S} c(s) \cdot \frac{|T|}{|S|} - |T|\gamma - \rho \\ -W \leq w_h \leq W & \forall h \in H \\ \\ e_s \in \{0,1\} & \forall s \in S \\ c_t \in \{0,1\} & \forall t \in T \\ w_h \in \mathbb{R} & \forall h \in H \\ \delta \in \mathbb{R} \\ \gamma \in \mathbb{R}_+ \end{cases} \tag{2.10}$$

2.6 Implementation and Computational Results

Tests are carried out on an Intel Pentium IV PC with 3GHz processor and 3.24 Gb RAM. The proposed methodology has been implemented in C++ using MS Visual Studio. The quality values q_i^j are numerically approximated by using C functions described in [79]. Pattern are generated by grouping up to p literals (see Tables), using $\nu_1 = 5$, $\nu_2 = 50$, $\eta^c = 2$, $\eta^e = 1$, $\eta_a^c = 1$, and $\eta_a^e = 0$. The support set selection problem, when modeled as knapsack (2.3) with all $s_i^j = 1$, is solved by simply ordering by quality values the binary attributes. When modeled differently, as in (2.1) or (2.2), is solved by means of IBM Cplex [65], a state-of-the-art implementation of branch-and-cut procedure (e.g. [76, 88]). The same solver is used for solving the pattern weights and classification threshold selection problem (2.10). Datasets used for the experiments are "Ionosphere", "Spambase", "Pima Indians Diabetes", "Statlog Heart", "Mushroom", "Adult" and "Poker Hand", publicly available from the UCI Repository of machine learning problems [49]. They were chosen in order to have a test bed containing different types of datasets (with many or few records, many or few fields, numerical or categorical, easy or difficult, etc.), so as to analyze the classifiers behavior under all conditions.

The first set, Ionosphere, is composed by 351 instances, each having 34 fields (plus the class). In particular, there are 32 real-valued fields and 2 binary ones. All 32 real-valued fields could be considered having normal distribution, one binary field could be considered having binomial distribution, the other is always 0. They are “data collected by a radar system in Goose Bay, Labrador. The targets were free electrons in the ionosphere. Good radar returns are those showing evidence of some type of structure in the ionosphere. Bad returns are those that do not; their signals pass through the ionosphere”, from UCI Repository [49].

The second set, Spambase, is composed by 4,601 instances, each having 57 fields (plus the class), all numerical. In particular, 55 are real-valued and 2 are integer; however they are the frequencies of “particular words or characters” and numbers of capital letters in an email, so all 57 were considered having normal distribution. Records of this dataset correspond to received emails, and the class “denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail”, from the UCI Repository [49].

The third set, Pima Indians Diabetes, is composed by 768 instances, each having 8 fields (plus the class). In particular, there are 2 real-valued fields and 6 integer ones. However, since 3 integer fields have a number of possible values high enough, 5 field could be considered having normal distribution, while 3 could be considered having binomial distribution. Fields are medical informations about “females patients of Pima Indian heritage living near Phoenix, Arizona, the class is whether the patient shows signs of diabetes”, from the UCI Repository [49].

The forth set, Statlog Heart, is composed by 270 instances, each having 13 fields (plus the class). In particular, there are 7 real-valued fields and 6 categorical or binary. The first 7 were considered having normal distribution. The last 6 could not be considered having binomial distribution, so they were treated as those with normal distribution but generating cut-points when inverting the class predominance due to the few number of possible values (see Section 2.3). Fields are several medical informations about patients, the class is whether the patient “has absence or presence of heart disease”, from the UCI Repository [49].

The fifth set, Mushroom, is composed by 8,124 instances, each having

22 fields (plus the class). All fields are categorical with very few possible values (no more than 12, some are just 2), so they were treated as those with binomial distribution but generating cut-points when inverting the class predominance (see Section 2.3). The records correspond to "mushrooms described in terms of physical characteristics, from Audobon Society Field Guide", and the "classification is poisonous or edible", from the UCI Repository [49].

The sixth set, Adult, is composed by 48,842 instances, each having 14 fields (plus the class). In particular, there are 6 real-valued fields that could be considered having normal distribution, the other 8 are categorical and were treated as fields with unknown distribution. They are "a set of reasonably clean person records extracted from the 1994 US Census database", from the UCI Repository [49]. The class is whether that person earns more than 50,000 USD per year or not.

The seventh set, Poker Hand, is composed by 1,025,010 instances, each having 10 fields (plus the class). For this dataset, the training set is predefined and composed by 25,010 records, corresponding to 2.44% of the total. Records have 5 categorical and 5 integer fields, but such integer fields should be considered categorical, since they are values of cards: just labels. All fields were treated as fields with unknown distribution, but the first 5 did not produce any cut-points, so they were excluded also for LIBSVM in oder to make the classifiers work in the same conditions (note that including them would produce worse results for LIBSVM). Each record is suit and rank of the 5 cards received in the poker game, drawn from a standard deck of 52. The class is the ranking of the entire poker hand, from 0 to 9. In order to obtain a binary classification problem, we assigned 0 to "nothing in hand" and 1 to any other point (pair, two pairs, etc.). The above data-sets have been classified using the following procedures:

1. The proposed one, called SLAD (Statistical and Logical Analysis of Data), solving the knapsack version (2.3) of the Support set selection problem and the described Pattern weights and classification threshold selection problem (2.10).

2. The standard LAD (Logical Analysis of Data) procedure, obtained from the former by not assigning values to binary attributes and solving an unweighted set covering problem (2.1) for the Support set selection, and using pattern weights w_h based on squared pattern coverage (see Section 2.5) and classification threshold $\delta = 0$.

3. A simplified version of SLAD, called RLAD (Reduced Logical Analysis of Data), solving the knapsack version (2.3) of the Support set selection problem and simply using the binary attributes for performing the classifi-

cation. In other words, patterns are composed by only one literal, and use pattern weights w_h based on squared pattern coverage (see Section 2.5) and classification threshold $\delta = 0$.

4. The publicly available LIBSVM 3.17 (Library for Support Vector Machines [33]), a very good C++ implementation of the Support Vector Machines methodology [32, 62], developed by Chih-Chung Chang and Chih-Jen Lin, possibly working on dataset previously scaled to a restricted range by means of svm-scale [33] (a preprocessing for improving accuracy).

Tests reported in Tables 2.2-2.8 are conducted as follows. A small number of record instances, representing respectively 5% and 10%, of the total, are randomly extracted from each data-set, and used as training set (except for Poker Hand, which is already subdivided into training and test). The rest of the data-set constitutes the test set. Such extraction is performed 10 times, and the results reported are averaged on the 10 trials. Tests are conducted on a best-against-best basis: we selected, for each dataset and training percentage, the classifier parameters producing the best accuracy. Tables report classification accuracy, computational times required by the whole classification procedure (in seconds, with time limit of 3600), parameters used for obtaining those results.

Algorithm	Training 5% (18/351)		
	Accuracy	Time	Parameters
LAD	69.60 %	0.60	standard, $\eta^c = 1, \eta^e = 0$
RLAD	70.98 %	0.04	$b = 35$
SLAD	85.41 %	0.04	$b = 10, p = 3, G = 1.8, W = 10^4$
LIBSVM	82.66 %	0.58	unscaled, -s 0 -t 2 -g 0.25 -c 1.6818 -e 0.001

Algorithm	Training 10% (36/351)		
	Accuracy	Time	Parameters
LAD	71.13 %	0.75	standard, $\eta^c = 1, \eta^e = 0$
RLAD	75.10 %	0.10	$b = 46$
SLAD	89.58 %	0.18	$b = 11, p = 3, G = 3.6, W = 10^4$
LIBSVM	86.64 %	0.63	unscaled, -s 0 -t 2 -g 0.1486 -c 1 -e 0.001

Table 2.2: Ionosphere (351 records, 34 fields) average on 10 trials.

As a general outcome, our experiment show that the effort invested in evaluating the quality of the different binary attributes returns a superior classification accuracy with respect to the standard LAD procedure. In the

Algorithm	Training 5% (230/4601)		
	Accuracy	Time	Parameters
LAD	83.10 %	2.14	standard, $\eta^c = 1, \eta^e = 0$
RLAD	75.70 %	0.18	$b = 240$
SLAD	88.90 %	0.92	$b = 100, p = 3, G = 23, W = 10^4$
LIBSVM	85.74 %	1.05	scaled, -s 0 -t 2 -g 0.5 -c 8.0 -e 0.001

Algorithm	Training 10% (460/4601)		
	Accuracy	Time	Parameters
LAD	84.32 %	2.21	standard, $\eta^c = 1, \eta^e = 0$
RLAD	78.55 %	0.50	$b = 370$
SLAD	89.51 %	1.61	$b = 200, p = 3, G = 46, W = 10^4$
LIBSVM	89.12 %	1.36	scaled, -s 0 -t 2 -g 0.5 -c 8.0 -e 0.001

Table 2.3: Spambase (4601 records, 57 fields) average on 10 trials.

Algorithm	Training 5% (38/768)		
	Accuracy	Time	Parameters
LAD	70.48 %	0.88	standard, $\eta^c = 1, \eta^e = 0$
RLAD	66.10 %	0.03	$b = 60$
SLAD	72.77 %	0.40	$b = 60, p = 3, G = 3.8, W = 10^4$
LIBSVM	70.54 %	0.80	scaled, -s 0 -t 2 -g 0.000173 -c 4096 -e 0.001

Algorithm	Training 10% (76/768)		
	Accuracy	Time	Parameters
LAD	72.36 %	1.15	standard, $\eta^c = 1, \eta^e = 0$
RLAD	67.90 %	0.06	$b = 64$
SLAD	73.96 %	0.62	$b = 64, p = 3, G = 7.6, W = 10^4$
LIBSVM	72.38 %	2.84	scaled, -s 0 -t 2 -g 0.25 -c 11.31 -e 0.001

Table 2.4: Pima Indians Diabetes (768 records, 8 fields) average on 10 trials.

totality of the analyzed cases, indeed, SLAD is more accurate than LAD. That additional effort clearly required an additional computational time, but that was almost negligible, and moreover, in the solution of the support set selection problem, weighted set covering problems can generally be solved in times which are much shorter than those needed for the corresponding non-weighted ones, so performing the above quality evaluation appears in

Algorithm	Training 5% (14/270)		
	Accuracy	Time	Parameters
LAD	62.40 %	0.78	standard, $\eta^c = 1, \eta^e = 0$
RLAD	64.25 %	0.02	$b = 54$
SLAD	77.23 %	0.02	$b = 14, p = 4, G = 1.4, W = 10^4$
LIBSVM	76.44 %	0.20	scaled, -s 0 -t 2 -g 0.00035 -c 512 -e 0.001

Algorithm	Training 10% (27/270)		
	Accuracy	Time	Parameters
LAD	68.80 %	0.78	standard, $\eta^c = 1, \eta^e = 0$
RLAD	69.00 %	0.04	$b = 64$
SLAD	80.24 %	0.04	$b = 22, p = 4, G = 2.7, W = 10^4$
LIBSVM	77.52 %	0.34	scaled, -s 0 -t 2 -g 0.125 -c 512 -e 0.001

Table 2.5: Statlog Heart (270 records, 13 fields) average on 10 trials.

Algorithm	Training 5% (406/8124)		
	Accuracy	Time	Parameters
LAD	76.90 %	2.67	standard, $\eta^c = 1, \eta^e = 0$
RLAD	76.39 %	0.32	$b = 176$
SLAD	98.79 %	0.69	$b = 20, p = 4, G = 40, W = 10^4$
LIBSVM	98.34 %	1.12	unscaled, -s 0 -t 2 -g 0.03125 -c 8.0 -e 0.001

Algorithm	Training 10% (812/8124)		
	Accuracy	Time	Parameters
LAD	80.15 %	3.85	standard, $\eta^c = 1, \eta^e = 0$
RLAD	78.48 %	0.45	$b = 200$
SLAD	99.72 %	0.83	$b = 25, p = 4, G = 81, W = 10^4$
LIBSVM	99.13 %	1.25	unscaled, -s 0 -t 2 -g 0.03125 -c 8.0 -e 0.001

Table 2.6: Mushroom (8124 records, 22 fields) average on 10 trials.

any case convenient. Furthermore, the solution of the support set selection problem as binary knapsack (2.3) using the above quality evaluation and all $s_i^j = 1$ is even faster and produces a very good classification accuracy.
The comparison with LIBSVM, which is currently deemed to be one of the most effective classifiers, show the effectiveness of the proposed approach. Indeed, SLAD obtains a classification accuracy that is always comparable

Algorithm	Training 5% (2442/48842)		
	Accuracy	Time	Parameters
LAD	76.32 %	35.10	standard, $\eta^c = 1, \eta^e = 0$
RLAD	75.39 %	0.78	$b = 150$
SLAD	82.16 %	7.10	$b = 80, p{=}3, G{=}244, W = 10^5$
LIBSVM	83.60 %	10.02	scaled, -s 0 -t 2 -g 0.125 -c 8.0 -e 0.001

Algorithm	Training 10% (4884/48842)		
	Accuracy	Time	Parameters
LAD	75.72 %	75.30	standard, $\eta^c = 1, \eta^e = 0$
RLAD	77.48 %	0.95	$b = 200$
SLAD	83.68 %	8.64	$b{=}100, p{=}3, G{=}488, W{=}10^5$
LIBSVM	84.28 %	12.58	scaled, -s 0 -t 2 -g 0.125 -c 8.0 -e 0.001

Table 2.7: Adult (48842 records, 14 fields) average on 10 trials.

or better than that of LIBSVM, in times that are often comparable and become much shorter on the larger data-sets, so the scalability of the proposed approach appears fully satisfactory.

Algorithm	Training 2.44% (25010/1025010)		
	Accuracy	Time	Parameters
LAD	-	>3600	standard, $\eta^c = 1, \eta^e = 0$
RLAD	63.75 %	48.6	$b = 80$
SLAD	83.86 %	1497.0	$b{=}10,p{=}4,G{=}2501,W{=}10^6$
LIBSVM	64.74 %	2640.0	scaled, -s 0 -t 2 -g 0.125 -c 8.0 -e 0.001

Table 2.8: Poker Hand (1025010 records, 10 fields) one trial.

Moreover, the simple classifier RLAD is considerable faster than any other of the considered procedures because no time consuming problems must be solved in the different steps of this procedure. Its accuracy is clearly inferior to that of the other classifiers, but is sometimes comparable. Therefore, it could be useful in practical applications where a timely (or even a real-time) classification is needed, or when very large datasets should be treated.

2.7 Summary and remarks

Classify with a good degree of accuracy and in short times on the basis of small training sets is required in a variety of practical applications. Unfortunately, obtaining these three desirable features together can be very difficult. We consider here the framework of the Logical Analysis of Data (LAD), and propose several enhancements to this methodology based on statistical considerations on the data. In particular, we use more information extracted from the training set to guide the support set selection step, and propose two reformulations of such a problem having several advantages. Moreover, we consider the problem of selecting the best parameters for the procedure (pattern weights and classification threshold), and formulate it as an optimization problem, overcoming this way the parameter tuning step. The proposed methodology, called Statistical and Logical Analysis of Data (SLAD), is tested on a test bed of publicly available datasets from the UCI repository, and compared to the standard LAD methodology and to the Support Vector Machines (SVM) methodology. Experiments show that the presented enhancements are able to increase the classification accuracy and reduce computation times with respect to standard LAD methodology. The comparison with SVM methodology, currently deemed one of the best classification technique, proves that SLAD provides good accuracy and timing results using very small training sets. Moreover, a simplified version of SLAD, called Reduced Logical Analysis of Data (RLAD), is proposed. All steps of this latter procedure can be solved in very short times, allowing a sensible speed-up of the whole classification procedure. As a consequence, real-time classification, or fast classification of massive datasets, can be undertaken.

Chapter 3

Balancing Problem in Agriculture

3.1 The Problem of Matrix Balancing

A Census of Agriculture is a very complex, important and expensive operation for a National Statistic Office. It is an essential activity, periodically performed for monitoring the agricultural sector (see e.g. [44]). Data collected in such a process have therefore a great intrinsic economic value, and moreover, in the case of EU countries, constitute a basis for assigning financial resources, planning production, and for several other economical European policies. As in any other large-scale survey, however, those data may contain errors or missing values, due to a variety of reasons. Nonetheless, the correct information must be published and provided to the EU level, also considering that large financial resources are allocated to the sector. Therefore, error *detection* and *correction* become crucial tasks. This kind of activity is generally called *Information Reconstruction*, or also *Data Cleaning*, within the field of Data Mining (see also [61, 70]), or *Data Editing and Imputation* within the field of Statistics (see also [41, 94]). Note that, in contexts different from the Census, the possibility of reconstructing exact values could be useful also for counteracting possible opportunistic behaviors (e.g. willingly erroneous declarations), and knowing that exact values can be reconstructed could indeed prevent such opportunistic behaviors.

Data are generally organized into conceptual units called records (see also [82]). In the case of a Census of Agriculture, data are typically constituted by farm codes, cultivation codes, size of cultivation areas and other amounts, years, etc., so we restrict our attention to numerical data. Agriculture is a rich source of large data mining problems, and a recent overview on the use of data mining techniques in this field is in [74]. The above Information Reconstruction tasks, in particular, can be performed by following different approaches, each of which having its own features. A main approach is based on the use of rules, called edits, that each data record must respect in order to be declared exact (see e.g. [7, 71]). Records not respecting such rules are declared erroneous. A seminal paper on the subject is [47]. However, satisfactory rules accuracy and computational efficiency often appear to be at odds. For this reason, rules are often converted into mathematical expressions, e.g. inequalities (see also [12]), and finding within a record the most probably erroneous fields or the most suitable values correcting those fields become nontrivial optimization problems (see e.g. [52] for an introduction to computational complexity). This allows to overcome the computational limits of other techniques (see e.g. [8, 71, 94]) based on the Fellegi Holt approach. Such a methodology has been adopted within the data Editing and Imputation software system DIESIS [26, 27] and in other works such as [41, 84].

In the described Census, each farm specifies the *cultivation area* used for each cultivation and *number of livestock* for each type of animal, divided in some cases also by year. Moreover, they specify total areas and total numbers of livestock. However, those totals may be inconsistent with the mentioned detailed information, and a classical problem is restoring data consistency by correcting errors. These errors should be corrected by mathematically “guessing” the correct values, since it is clearly impossible to contact again the farm or inspect it somehow. The main issue is doing this on large datasets both efficiently and in order to obtain corrected data as similar as possible to the exact (but unknown) data.

This Chapter presents an innovative procedure for solving this problem based on optimization. In particular, Section 3.2 describes possible alternative models for the above problem and explains the development of the proposed mixed integer linear programming model. Section 3.3 reports computational results in the case of the Italian Census of Agriculture 2010 (“Censimento Generale dell’Agricoltura 2010”), both for plants cultivations and for livestock. Clearly, the proposed model is not limited to the case of an Agricultural Census, but can be used for any other problem sharing the same characteristics, in particular the presence of balancing conditions. Note that, to the best of our knowledge, no previous attempt to treat this large-size Census problem with a discrete optimization approach was made, and only *ad hoc* procedures, designed by experts after an analysis of the specific available data, were used. This work has been published in [13].

3.2 Formulating the MILP Model

Data obtained from each *farm* during the described Census contain *detail* information about the *cultivation area* used by that farm for each cultivation and the *number of livestock* for each type of animal. Those data may sometimes be erroneous or missing, due to a variety of reasons. In such cases, errors should be automatically detected and corrected, i.e. the information that was corrupted and lost should be "reconstructed" in order to be as similar as possible to the unknown exact value. Moreover, each farm also declares other information (called *macrodata*, information about totals): the total cultivation area and the total number of livestock, and in some cases those totals are also divided into *subtotals* by year of planting. Clearly, balancing conditions must hold between all the above microdata and the corresponding macrodata: each total (or year subtotal) must be equal to the sum of those details concerning its parts. When such conditions do not hold, data are inconsistent.

Records incurring in this problem are detected by checking the balancing conditions, which are called *balance edits*. However, when a balance edit is violated, the error could be either on the detail side or on the total side of the equation. The less *reliable* information should now be changed in order to restore consistency. It is generally assumed, in similar cases, that details constitute the less reliable information, since totals have already been confirmed from other sources. This mathematical problem of adjusting the entries (here the microdata) of a large matrix to satisfy prior consistency requirements (here given by the macrodata) is called *matrix balancing* [87] and occurs in several fields, such as economics, urban planning, statistics, demography, etc. The problem is also related to the matrix rounding problem [5], consisting in rounding off the elements of a matrix consistently with its row and column sums, often arising in economic statistics, and belongs to the broad category of *matrix scaling* problems [6].

In some cases of matrix balancing problems the only aim is restoring balancing without further objectives, and iterative scaling algorithms can be used, e.g. the RAS algorithm [69]. In other cases, on the contrary, the variations introduced for balancing the matrix should pursue an objective that typically depends on the specific application. In the case of Census data, the choice of the objective is a delicate issue for avoiding data distortions, and makes this problem different from other types of balancing problems. Errors in microdata could broadly be divided into systematic errors and random errors [51]. Systematic errors are those caused by specific (and often traceable) mechanisms, e.g. usage of a wrong unit of measurement, OCR error, etc., and are generally treated during a preliminary correction phase [42]. Our central problem is therefore correcting microdata values affected by random errors. In this case, changes from the available microdata values should be minimized, according to specific distance criteria, since it is generally deemed that this should produce data as similar as possible to the unknown exact data (Fellegi-Holt paradigm [47, 71]). An optimization approach is therefore required.

The models proposed for the above problem will be hereinafter explained by referring to the specific case of *vineyards*. This is one of the most important cases: dozens of grapes varieties exist, and they determine type and quality of wines produced. The case has great economic relevance and, due to its large dimension, is also computationally demanding. Moreover, those data are used when allocating European financial resources and when reorganizing wine production. However, the proposed models are clearly not limited to that case, but can be used for any other similar problem. Each farm could have several vine types, and each of them could have been planted in a different time period (e.g. a specific year). Denote by

$I = \{1,\ldots,$ n$\}$ the set of indices of all possible vine types; with $n = 442$;

$K = \{1,\ldots,$ m$\}$ the set of indices of all possible time periods; with $m = 6$.

For each farm, denote by

$a_{\mathtt{ik}}$ (real valued ≥ 0) the area of vine type i planted in period k declared by the farm, with $i \in I$ and $k \in K$;

$a_{\mathtt{i0}}$ (real valued ≥ 0) the total area of vine type i (planted during any of the periods) declared by the farm, with $i \in I$;

T_k (real valued ≥ 0) the total vine area planted in period k declared by the farm, with $k \in K$;

T (real valued ≥ 0) the total vine area owned by the farm.

In order to reconstruct the erroneous information, we need the following set of decision variables:

$x_{\mathtt{ik}}$ (real valued ≥ 0, $\leq$ S) = the area of vine type i that, according to our reconstruction, has been planted in period k by the farm, with $i \in I$ and $k \in K$;

$x_{\texttt{i0}}$ (real valued ≥ 0, $\leq$ mS) = the total area of vine type i that, according to our recon-struction, has been planted (during any of the periods) by the farm, with $i \in I$.

In other words, $x_{\texttt{ik}}$ is the correct value for $a_{\texttt{ik}}$. When reconstructing information for a Census, as in the case of other large-scaled surveys, it is generally assumed that the changes introduced in the data should be somehow minimized. This because, in absence of further information, being as similar as possible to the exact (unknown) data corresponds to being as similar as possible to the available (even if possibly erroneous) data. By following this minimum change paradigm, two basic alternatives exist: one is minimizing the number of changes, the other minimizing the amount of those changes.

If we need to distinguish when our reconstruction provides a result which is different form the available declaration (i.e. a change), we need the following set of binary variables:

$$y_{\texttt{ik}} = \begin{cases} 1 & \text{if } \ x_{\texttt{ik}} \text{ is different from } x_{\texttt{ik}} \qquad \forall i = 1, \ldots, n \ \ \forall k = 0, \ldots, m \\ 0 & \text{otherwise} \end{cases}$$

The presence of binary variables clearly has its impact on the complexity of the model: by adding the other constraints needed for this problem, which are linear, we obtain an integer linear program. Minimizing the total number of changes corresponds to the following objective function

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} \ y_{\texttt{ij}} \tag{3.1}$$

When variables y are used, they should be linked to the x variables by constraints imposing that $y_{\texttt{ik}}$ takes value 1 when $x_{\texttt{ik}} <> a_{\texttt{ik}}$ (using a certain numerical precision), otherwise those variables could be inconsistent. There is no need for constraints imposing $y_{\texttt{ik}} = 0$ when $x_{\texttt{ik}} = a_{\texttt{ik}}$ because the objective (3.1) itself would do that. Value M is a real number greater than all possible values of the left-hand side of the following inequalities.

$$a_{\texttt{ik}} - x_{\texttt{ik}} \leq M y_{\texttt{ik}} \qquad \forall i = 1, \ldots, n \ \ \forall k = 0, \ldots, m \tag{3.2}$$

$$x_{\texttt{ik}} - a_{\texttt{ik}} \leq M y_{\texttt{ik}} \qquad \forall i = 1, \ldots, n \ \ \forall k = 0, \ldots, m \tag{3.3}$$

When, on the other hand, we are interested in measuring the difference between our reconstruction $x_{\texttt{ik}}$ and the available declaration $a_{\texttt{ik}}$, we should consider a generic norm of this difference:

$$\| a_{\mathtt{ik}} - x_{\mathtt{ik}} \|_p \tag{3.4}$$

Several norm types and norm-induced functions exist [10]. We consider more suitable to our reconstruction problems the following three:

- The squared Euclidean norm, defined as $(\| u - v \|_2)^2 = \sum_{h=1}^{q}(u_h - v_h)^2$;
- the so-called Manhattan norm, defined as $\| u - v \|_1 = \sum_{h=1}^{q} |u_h - v_h|$;
- the so-called Chebyshev norm, defined as $\| u - v \|_\infty = max_h\{|u_h - v_h|\}$.

Clearly, the structure of the optimization model that we must solve depends now on this choice. In the first case (squared Euclidean norm), minimizing the total amount of the changes corresponds now to the following objective function, containing quadratic terms.

$$\min \sum_{i=1}^{n} \sum_{k=0}^{m} (a_{\mathtt{ik}} - x_{\mathtt{ik}})^2 = \min \sum_{i=1}^{n} \sum_{k=0}^{m} (a_{\mathtt{ik}}^2 - 2a_{\mathtt{ik}}x_{\mathtt{ik}} + x_{\mathtt{ik}}^2) \tag{3.5}$$

However, all of them are simply squared variables $(x_{\mathtt{ik}})^2$, so they are strictly convex, and a conic combination of those strictly convex terms produces a separable strictly convex function [17]. By adding to that the linear terms of (3.5) and the constraints needed for this problem (described later), which are linear, the problem remains efficiently solvable (see e.g. [15, 64]).

In the second case (Manhattan norm), there are absolute values in the objective. However, they can be easily linearized by introducing additional variables:

$$s_{\mathtt{ik}} \ (real\ valued \geq 0) = the\ value\ of\ |a_{\mathtt{ik}} - x_{\mathtt{ik}}|, \quad \forall i = 1, \ldots, n \ \forall k = 0, \ldots, m$$

and linear constraints enforcing their meaning

$$s_{\mathtt{ik}} \geq a_{\mathtt{ik}} - x_{\mathtt{ik}}, \quad s_{\mathtt{ik}} \geq x_{\mathtt{ik}} - a_{\mathtt{ik}} \quad \forall i = 1, \ldots, n \ \ \forall k = 0, \ldots, m \tag{3.6}$$

We can now minimize the linear function $\sum_{i=1}^{n} \sum_{k=0}^{m} s_{\mathtt{ik}}$. When adding the other constraints needed for this problem, which are linear, the problem becomes an easily solvable linear program.

In the third case (Chebyshev norm), we have a min-max objective in the problem that again can be easily linearized by introducing one additional variable

$$t(real\ valued \geq 0) = value\ of\ max_{\mathtt{ik}}\{|a_{\mathtt{ik}} - x_{\mathtt{ik}}|\}, \ \forall i = 1, \ldots, n \ \forall k = 0, \ldots, m$$

and linear constraints enforcing the above meaning

$$t \geq a_{\mathtt{ik}} - x_{\mathtt{ik}}, \quad t \geq x_{\mathtt{ik}} - a_{\mathtt{ik}} \qquad \forall i = 1, \ldots, n \ \ \forall k = 0, \ldots, m \tag{3.7}$$

We now simply minimize t. When adding the other constraints needed for this problem, which are linear, the problem becomes again an easily solvable linear program.

Clearly, also a combination of the above alternatives can be considered. The characteristics of the specific real problem will determine, from case to case, the choice of the objective among the described ones or their possible combinations. In our case, we consider more representative of the real problem's aim the minimization of the total number of changes, and, in second place, the minimization of the amount of those changes. This because a change with respect to a value that has been deliberately declared has intrinsically a very high cost. Therefore, we prefer maintaining the maximum number of those declared values, even if this may result in a greater amount of the changes that we are forced to introduce. The objective function becomes:

$$\min \, (M' \sum_{i=1}^{n} \sum_{k=0}^{m} y_{\mathtt{ik}} + \sum_{i=1}^{n} \sum_{k=0}^{m} s_{\mathtt{ik}}) \tag{3.8}$$

where the first sums are multiplied by a numerical value M' weighting the relative importance of the first part with respect to the second one. We chose $M' = S$, so that a single change weights as much as the maximum amount of a change. We now describe the balancing conditions that should be respected in our case. The sum of vine areas of any type planted in period k must be equal to the total vine area planted in period k (called balancing over vine types)

$$\sum_{i=1}^{n} x_{\mathtt{ik}} = T_k \qquad \forall k \in K \tag{3.9}$$

The sum of the areas of vine type i planted in periods from 1 to m must be equal to the area of the same vine type planted along all the periods (called balancing over time periods)

$$x_{\mathtt{i0}} = \sum_{k=1}^{m} x_{\mathtt{ik}} \qquad \forall i \in I \tag{3.10}$$

The sum of vine areas of any type planted in any period must be equal to the total vine area owned by the farm (called overall balancing)

$$\sum_{i=1}^{n} \sum_{k=1}^{m} x_{\mathtt{ik}} = T \tag{3.11}$$

Clearly, any other type of balancing condition could be expressed as other linear constraints. Note that the structure of balancing constraints (3.9) and (3.10) could be considered as defining a transportation problem (see e.g. [10, 89]) with a set of origins I and a set of destinations K, values $a_{\mathtt{i0}}$ being the supply at origin i, values T_k being the demand at destination k, variables $x_{\mathtt{ik}}$ being the amount to be shipped from source i to destination k. However, the values $a_{\mathtt{i0}}$ are in our case declared values that we may change (using the $x_{\mathtt{i0}}$ variables), and moreover there is no guarantee that the following condition, essential for the feasibility of a transportation problem, is respected:

$$\sum_{i=1}^{n} a_{\mathtt{i0}} = \sum_{k=1}^{m} T_k \tag{3.12}$$

The complete mixed integer linear programming model is therefore the following:

$$\left\{\begin{array}{ll}
\min(M' \sum_{i=1}^{n}\sum_{k=0}^{m} y_{\mathtt{ik}} + \sum_{i=1}^{n}\sum_{k=0}^{m} s_{\mathtt{ik}}) & \\
 & \\
\sum_{i=1}^{n} x_{\mathtt{ik}} = T_k & \forall k \in K \\
x_{\mathtt{i0}} = \sum_{k=1}^{m} x_{\mathtt{ik}} & \forall i \in I \\
\sum_{i=1}^{n}\sum_{k=1}^{m} x_{\mathtt{ik}} = T & \\
a_{\mathtt{ik}} - x_{\mathtt{ik}} \le M y_{\mathtt{ik}} & \forall i = 1,\dots,n \;\; \forall k = 0,\dots,m \\
x_{\mathtt{ik}} - a_{\mathtt{ik}} \le M y_{\mathtt{ik}} & \forall i = 1,\dots,n \;\; \forall k = 0,\dots,m \\
s_{\mathtt{ik}} \ge a_{\mathtt{ik}} - x_{\mathtt{ik}} & \forall i = 1,\dots,n \;\; \forall k = 0,\dots,m \\
s_{\mathtt{ik}} \ge x_{\mathtt{ik}} - a_{\mathtt{ik}} & \forall i = 1,\dots,n \;\; \forall k = 0,\dots,m \\
0 \le x_{\mathtt{ik}} \le S & \forall i \in I \qquad \forall k \in K \\
0 \le x_{\mathtt{i0}} \le mS & \forall i \in I \\
s_{\mathtt{ik}} \ge 0 & \forall i = 1,\dots,n \;\; \forall k = 0,\dots,m \\
x_{\mathtt{ik}}, s_{\mathtt{ik}} \in \mathbb{R} & \forall i = 1,\dots,n \;\; \forall k = 0,\dots,m \\
y_{\mathtt{ik}} \in \{0,1\} & \forall i = 1,\dots,n \;\; \forall k = 0,\dots,m
\end{array}\right. \tag{3.13}$$

3.3 Computational Analysis

By sequentially solving the above model for each farm, we perform the requested Information Reconstruction process. This procedure was implemented in C++, using ILOG Concert Technology [66] in order to express the described optimization models. The models themselves are solved by means of the state-of-the-art branch-and-cut (see e.g. [10, 76]) procedure implemented by the solver ILOG Cplex [65], running on a 16 cores server having 128Gb of RAM and Linux Operating System. An analysis of the behavior of open source solvers was published in [11]. The resulting software system has been applied for the treatment of data from the Italian Census of Agriculture 2010 ("Censimento Generale dell'Agricoltura 2010"), with specific respect to the cases of:

1. Vineyards suitable for "controlled origin" wine, considered in Table 3.1;
2. Vineyards not suitable for "controlled origin" wine, considered in Table 3.2;
3. Generic cultivations, considered in Table 3.3;
4. Livestock, considered in Table 3.4.

Note that, in the last two cases, microdata are not subdivided by year of planting but by geographical area. In the above four cases, we report results for each Italian region and for all Italy (1st column); the total number of farms not respecting the balancing conditions (2nd column); the total number of records involved in those unsatisfied balancing conditions (3rd column); the total number of changes operated by the reconstruction process (4th column).

Moreover, we analyze in greater detail those changes: we report the percentages of area (or heads) modified by the procedure, computed with respect to the total area involved in that case (or to the total number of animals). Such modifications can be done by adding (5th column) and/or by subtracting (6th column), and note that those quantities are not bounded to be equal, since errors are not so. Finally, we report the total processing time in seconds (7th column).

The practical behavior of the proposed procedure should now be evaluated both from the computational and from the data quality points of view. As observable, the procedure is very fast: each single model is solved to optimality in extremely short times (generally about 0.02 sec.) so that the processing of all the Italian farms requires, for the 4 cases together, only about 50 minutes.

The quality of the obtained data has been evaluated by considering: (i) the ability to restore balancing; and (ii) the variation produced in the data

Region	Farms	Records	Changes	Added Area	Subtracted Area	Time
Piemonte	696	2055	1866	0.15%	-1.70%	23.5
Valle d'Aosta	22	55	46	0.00%	-0.00%	0.6
Lombardia	468	1488	1423	0.16%	-0.75%	17.0
Veneto	3817	6916	5602	4.23%	-0.96%	79.2
Friuli-Venezia Giulia	286	1528	945	0.13%	-0.38%	17.5
Liguria	124	257	493	0.01%	-0.04%	2.9
Emilia-Romagna	336	940	640	0.07%	-0.74%	10.8
Toscana	3392	6099	4332	3.20%	-1.01%	69.8
Umbria	73	248	173	0.05%	-0.13%	2.8
Marche	1359	2243	1541	0.79%	-0.21%	25.7
Lazio	432	925	1455	0.13%	-0.55%	10.6
Abruzzo	197	324	300	0.04%	-0.36%	3.7
Molise	407	479	428	0.12%	-0.01%	5.5
Campania	420	953	1212	0.13%	-0.41%	10.9
Puglia	6954	7895	8016	3.04%	-0.72%	90.4
Basilicata	61	78	104	0.02%	-0.06%	0.9
Calabria	168	223	369	0.04%	-0.12%	2.6
Sicilia	620	973	1764	0.43%	-0.94%	11.1
Sardegna	221	394	668	0.05%	-0.45%	4.5
Bolzano	125	446	241	0.01%	-0.35%	5.1
Trento	268	899	482	0.04%	-0.34%	10.3
Italy total	20346	35418	32100	12.85%	-10.21%	405.5

Table 3.1: Vineyards suitable for controlled origin wine.

by the reconstruction process.

As for the first aspect, data obtained by the procedure were able to satisfy the balancing conditions in the totality of the cases (100%). As for the second aspect, a positive feature for a general information reconstruction procedure is satisfying requirements while not changing the data exceedingly. In the analyzed cases, in addition to the theoretical guarantee that the number of changes is minimal, we observe that the amount of the variations is always a small percentage. This means that the procedure was able to reconstruct information without distorting the data.

The accuracy of the reconstructed information has been further evaluated by setting up a specific experiment. A large dataset of 274,687 records representing all vineyards obtained from about 126,000 farms, all exact, were perturbed by introducing random errors with uniform distribution at 3 different intensities, so that respectively about 1%, 5% and 10% of the microdata values have been changed. This was performed 20 times, in order to obtain statistically significant results, so 60 different large erroneous datasets were obtained. After this, the reconstruction procedure was applied on all of them, and the 60 obtained (corrected) datasets were compared to

Region	Farms	Records	Changes	Added Area	Subtracted Area	Time
Piemonte	528	866	1961	0.05%	-0.28%	9.9
Valle d'Aosta	168	254	173	0.01%	-0.01%	2.9
Lombardia	2312	3761	3172	0.38%	-0.25%	43.1
Veneto	1346	3811	2593	0.33%	-1.05%	43.7
Friuli-Venezia Giulia	1489	2931	2223	0.19%	-0.19%	33.6
Liguria	868	1465	1321	0.06%	-0.04%	16.8
Emilia-Romagna	436	918	814	0.10%	-0.62%	10.5
Toscana	4558	11481	6747	0.83%	-4.88%	131.6
Umbria	2200	4924	2466	0.18%	-0.17%	56.4
Marche	3005	5850	3472	0.33%	-0.09%	67.0
Lazio	4333	7363	7768	0.43%	-0.80%	84.4
Abruzzo	5392	10664	5673	0.71%	-0.94%	122.2
Molise	1732	3392	1841	0.24%	-0.05%	38.9
Campania	10904	18092	13728	0.92%	-0.67%	207.3
Puglia	10897	15251	13383	2.86%	-1.48%	174.8
Basilicata	370	486	659	0.04%	-0.15%	5.6
Calabria	2455	3391	4513	0.41%	-0.89%	38.9
Sicilia	4224	7497	9630	2.00%	-3.54%	85.9
Sardegna	642	1385	2528	0.07%	-0.72%	15.9
Bolzano	11	21	19	0.00%	-0.01%	0.2
Trento	1936	2400	1977	0.13%	-0.05%	27.5
Italy total	59806	106203	86661	10.25%	-16.87%	1217.0

Table 3.2: Vineyards not suitable for controlled origin wine.

the original exact one.

Statistical indicators commonly used for measuring the differences between real and predicted values, such as Relative Root Mean Square Error (RRMSE), are practically 0 ($< 10^{-5}$) for all the corrected datasets. This means that the quality of the reconstruction is satisfactory. However, in order to obtain more insight, we analyzed the reconstruction with an even greater detail: we compared each single reconstructed value to its original value, and checked whether it was exactly identical or not. Note that such test is extremely strict, probably beyond the requirements of a similar reconstruction process. The results are presented in Table (3.5). The percentage of reconstructed values that are exactly equal to the original values has been computed by subdividing the datasets on the basis of the number of errors actually introduced in each farm. Clearly, those percentages lower when the number of errors introduced in the farm increases, but accuracy is anyway extremely high. Even when the farm data contain a considerable number of errors (from 4 to 10, that is often more than what happens in usual practice), the reconstructed values are exactly equal to the original ones in a very high percentage of the cases.

Region	Farms	Records	Changes	Added Area	Subtracted Area	Time
Piemonte	624	1682	1191	0.037%	-0.016%	19.2
Valle d'Aosta	84	221	157	0.000%	0.000%	2.5
Lombardia	1436	3784	2162	0.030%	-0.069%	43.3
Veneto	5715	13604	11033	0.383%	-0.670%	155.8
Friuli-Venezia Giulia	734	1955	1123	0.010%	-0.002%	22.4
Liguria	264	645	458	0.001%	0.000%	7.3
Emilia-Romagna	252	742	976	0.168%	-0.042%	8.5
Toscana	3006	6571	6274	0.486%	-0.319%	75.2
Umbria	513	1160	931	0.007%	-0.003%	13.2
Marche	2210	5347	4615	0.229%	-0.088%	61.2
Lazio	912	2023	1553	0.009%	-0.002%	23.1
Abruzzo	1960	4540	2650	0.038%	-0.103%	52.0
Molise	2006	4649	3817	0.047%	-0.035%	53.2
Campania	2854	6538	4403	0.015%	-0.003%	74.9
Puglia	18205	41924	34881	0.561%	-0.527%	480.3
Basilicata	433	996	710	0.011%	-0.005%	11.4
Calabria	506	1166	851	0.007%	-0.008%	13.3
Sicilia	1117	2546	1772	0.035%	-0.002%	29.1
Sardegna	223	512	407	0.006%	-0.004%	5.8
Bolzano	23	96	83	0.012%	-0.037%	1.1
Trento	889	2280	1344	0.005%	-0.003%	26.1
Italy total	43966	102981	81391	2.098%	-1.938%	1180.0

Table 3.3: Other cultivations.

Region	Farms	Records	Changes	Added Area	Subtracted Area	Time
Piemonte	704	947	719	0.317%	-0.107%	9.7
Valle d'Aosta	38	51	38	0.000%	0.000%	0.5
Lombardia	528	784	574	0.390%	-0.130%	8.0
Veneto	3797	4449	4489	7.348%	-2.763%	45.4
Friuli-Venezia Giulia	164	210	169	0.017%	-0.285%	2.1
Liguria	71	92	71	0.000%	0.000%	0.9
Emilia-Romagna	333	593	500	5.585%	-8.403%	6.1
Toscana	1554	1878	1973	0.265%	-0.037%	19.2
Umbria	124	160	124	0.000%	-0.554%	1.6
Marche	1171	1488	1592	0.823%	-0.428%	15.2
Lazio	302	380	305	0.000%	0.000%	3.9
Abruzzo	215	286	217	1.634%	0.000%	2.9
Molise	1003	1303	1281	0.437%	-0.477%	13.3
Campania	387	471	394	0.001%	0.000%	4.8
Puglia	3734	4144	4556	0.595%	-0.570%	42.3
Basilicata	123	173	126	0.001%	0.000%	1.8
Calabria	266	320	270	0.000%	0.000%	3.3
Sicilia	338	441	340	0.000%	0.000%	4.5
Sardegna	276	489	281	0.004%	0.000%	5.0
Bolzano	127	181	127	0.000%	0.000%	1.8
Trento	97	111	97	0.000%	0.000%	1.1
Italy total	15352	18951	18243	17.418%	-13.754%	193.5

Table 3.4: Livestock.

	Percentage of Exactly Reconstructed Values		
Errors per Farms	Pertubation at 1%	Pertubation at 5%	Pertubation at 10%
1	99.9%	99.9%	99.9%
2	98.1%	98.2%	98.2%
3	86.0%	86.6%	83.4%
4÷10	81.8%	56.3%	49.1%

Table 3.5: Accuracy of the reconstruction process.

3.4 Summary and Remarks

Information Reconstruction is a crucial task in the case of large surveys, such as a Census of Agriculture, as well as for other applications of database processing. A typical problem arising in the described Census consists in checking, and correcting when needed, the areas declared by each farm for each cultivation. This type of balancing problem is extremely important and has a great economical relevance. Moreover, in contexts different from the Census, the possibility of reconstructing exact values could be useful for counteracting opportunistic behaviors, e.g. willingly erroneous declarations for influencing resources allocation or production plans.

Similar problems could be formulated in different manners. This particular Census problem has very specific aims and requirements, and it was deemed that they were better represented by the proposed mixed integer linear model (3.13). This problem has not been previously solved by using discrete optimization or some other model similar to the one proposed in this Chapter. Before the development of this approach, data were examined by statistical experts and the changes that appeared necessary were introduced, generally in a deterministic way and often interactively.

The procedure has been applied in the case of the Italian Census of Agriculture 2010 with specific respect to the 4 most important cases. Clearly, the proposed class of models is not limited to the case of an Agricultural Census, but can be used for other problems sharing the same characteristics, in particular the presence of balance requirements and minimum change objective. Results are very encouraging both form the computational and from the data quality point of view. The sequence of arisen mixed integer problems can be solved to optimality by using a state-of-the-art implementation of branch-and-cut procedures. Each single model is solved in extremely short times. In the totality of the cases the reconstructed information was able to satisfy the balancing conditions without excessively distorting the data, as resulted from the analysis of the variations introduced in the whole datasets.

Moreover, a specific experiment proves that the reconstructed information was exactly equal to the original uncorrupted one in an exceedingly high percentage of the cases.

Chapter 4

Reconstruction of Cultivation Data in Agriculture

4.1 Reconstruction of Cultivation Data

Data collected in a Census of Agriculture, as in any other large-scale survey, may contain errors or missing values, and their automatic *detection* and *correction* are crucial tasks. This kind of activity is generally called *Information Reconstruction*, or also *Data Cleaning*, within the field of Data Mining [61], or *Data Editing and Imputation* within the field of Statistics [41, 94].

Several approaches to these problems have been developed, dating back to the work of Fellegi-Holt [47]. An approach to verify the accuracy of these informations and to restore data consistency is based on the use of rules (edits), that each data record must respect in order to be declared exact [7, 26]. Rules can be converted into linear inequalities, and the problem of finding within a record the most probably erroneous fields or the most suitable values correcting those fields can be modeled as nontrivial optimization problems. This approach allows to overcome the computational limits of the Fellegi Holt methodology and it has been implemented within the data Editing and Imputation software system DIESIS [26, 27].

Agricultural Census data typically are numerical values, such as farm codes, cultivation codes, size of cultivation areas, etc. Similar data are generally organized into very large numerical records. In the described Census, each *farm* specifies the cultivation area used for each *cultivation.* A classical problem is verifying the accuracy of this information. In the event that farm declarations are considered unreliable, it's necessary to assign the correct cultivations to the area for which it's not known. Errors should be detected and corrected, by mathematically "guessing" the correct values, since it is clearly impossible to contact again the farm or inspect somehow the cultivations.

Farms can extend on one or more districts, and the area owned by each farm in each district is known. Therefore, the compatibility of each cultivation with each district can also be evaluated (some cultivation can grow only on specific types of soils, or need specific climatic conditions, latitude, altitude, etc.). In the specific case of vineyards, considered in this Chapter, there may be cultivation areas with missing or erroneous vineyard codes. For these areas, it is required to assign a code, according to a set of consistency constraints, taking also into account the compatibility between each district and the type of vineyard. Theoretically, the elements for solving the above problem are available, but the problem is doing this on large datasets both efficiently and in an unbiased manner.

This Chapter presents an innovative automatic procedure for solving this problem based on a discrete mathematical optimization model. In particular, Section 4.2 describes problem details and the proposed mixed integer linear programming model [10, 76], explaining its features. Section 4.3 reports computational results in the case of the Italian Census of Agriculture 2010, with specific respect to the case of vineyards. This is probably the most important case for the considered problem, since dozens of vine varieties exist, and they determine the type and the quality of wines produced. Relevant economic aspects are therefore involved, and vine cultivation and use are also regulated by legislation. Clearly, the proposed model is not limited to the case of a Census of Agriculture, but can be used for any other problem sharing the same characteristics. This work has been published in [14].

4.2 A Discrete Mathematical Model

Data obtained from each farm during the described Census contain information about the area used by that farm for each cultivation. Those data may sometimes be erroneous or missing, due to a variety of reasons. In such cases, errors should be automatically detected and corrected, i.e. the information that was corrupted and lost should be "reconstructed" in order to be as similar as possible to the (unknown) exact value.

Farms are located over the state territory. This territory is subdivided into many districts. Each farm can extend on one or more district. Denote by

$I = \{1, \ldots, \mathrm{n}\}$ the set of all possible cultivations;

$J = \{1, \ldots, \mathrm{m}\}$ the set of all possible districts.

Focusing on a single farm, and denoting by f its total area, all the cultivations declared by that farm are checked. Some of them verify a set of rules and conditions prepared for this aim and are therefore considered reliable, while some other do not. This may happen either because some of the declarations appear erroneous, or because there is a discordance between the total area declared and the sum of the areas declared for each cultivation. Denote by a the total farm area reliably assigned, i.e. the area for which the farm declaration are considered reliable. On the contrary, by grouping all the *unreliable declarations*, a nonempty area often remains for which the cultivation is not known. That area will be called unassigned area and denoted by u. Clearly, $f = a + u$. The central problem of our Information Reconstruction process consist now in assigning the cultivations to the mentioned unassigned area. In this Section a discrete mathematical model for this problem is proposed. For each farm, denote by

s_i (real value ≥ 0) the total area that the farm uses for cultivation i, with $i \in I$. Note that this area may span on one or more districts, and the farm does not declare, nor generally even consider, such subdivision. These values are only the ones, among all the cultivation data declared by farms, that can be considered reliable, so $\sum_i (s_i) = a$.

d_j (real value ≥ 0) the total area owned by the farm in district j, with $j \in J$. These values are not surveyed during the considered Census but are already available and are reliable.

$p_{\mathtt{ij}}$ (real value $\in [0, 1]$) the likelihood of having cultivation i in district j, with $i \in I$ and $j \in J$. Values near to 1 means high likelihood, near to 0 means very low likelihood. This values are estimated on the basis of agricultural registrations and studies, not surveyed during the considered Census.

Moreover, there are areas where specific cultivations may be used to produce foods having “controlled origin” (in Italian DOC). In particular, for the unassigned area u, it is possible to partition it into a portion that is suitable for "controlled origin" and a portion that is not suitable for that. Denote by

> C (reale value ≥ 0) the total unassigned area owned by the farm in cultivations suitable for "controlled origin";
>
> N (real value ≥ 0) the total unassigned area owned by the farm in cultivations not suitable for"controlled origin", so that $C + N = u$.

Those areas C and N should be assigned in order to maximize the likelihood of the assignment. Note that it is not known which district the unassigned area u is located into. On the other hand, the likelihood values depend on the districts. As a consequence, we need to locate the unassigned area u on the districts. This is apparently hard to obtain. A way of doing so is locating on the districts each of the reliable cultivation areas s_i, and then obtaining the location of u as the portion of farm area f not covered by a. In order to model the described problem, we need to introduce the following sets of decision variables:

> $x_{\mathtt{ij}}$ (real value ≥ 0) the area of cultivation i that, according to our reconstruction, is localized in district j, with $i \in I$ and $j \in J$;
>
> $v_{\mathtt{ij}}$ (real value ≥ 0) the portion of C that, according to our reconstruction, is used for cultivation i and localized in district j, with $i \in I$ and $j \in J$;
>
> $w_{\mathtt{ij}}$ (real value ≥ 0) the portion of N that, according to our reconstruction, is used for cultivation i and localized in district j, with $i \in I$ and $j \in J$.

Moreover, each of the farm unassigned areas C and N generally contains only a specific cultivation, and not a mixture of different cultivations. We therefore want to assign all C to one single type of cultivation, and not to fragment it among all the cultivations compatible with that area. A similar requirement holds for N. This requires the use of additional binary decision variables

$$y_i = \begin{cases} 1 & \text{if } C \text{ is assigned in our reconstruction to cultivation } i, \text{with } i \in I \\ 0 & \text{otherwise} \end{cases}$$

$$z_i = \begin{cases} 1 & \text{if } N \text{ is assigned in our reconstruction to cultivation } i, \text{with } i \in I \\ 0 & \text{otherwise} \end{cases}$$

Now it's possible to formulate a mixed integer linear programming model for each farm. Cultivation assignment to areas should be done in order to maximize the likelihood. Our objective function is therefore

$$\max \sum_{\forall i,j} p_{\mathtt{ij}} x_{\mathtt{ij}} + \sum_{\forall i,j} p_{\mathtt{ij}} v_{\mathtt{ij}} + \sum_{\forall i,j} p_{\mathtt{ij}} w_{\mathtt{ij}} \tag{4.1}$$

This assignment should obviously verify a set of constraints. First of all, the sum of the areas assigned to the different cultivations in each district j must be equal to the area owned by the farm in district j:

$$\sum_{\forall i} x_{\mathtt{ij}} + \sum_{\forall i} v_{\mathtt{ij}} + \sum_{\forall i} w_{\mathtt{ij}} \ = \ d_j \qquad \forall j = 1, \ldots, m \tag{4.2}$$

The sum of the areas used by the farm for cultivation i over all the districts must be equal to the total area used by the farm for cultivation i:

$$\sum_{\forall j} x_{\mathtt{ij}} = s_i \qquad \forall i = 1, \ldots, n \tag{4.3}$$

The sum of the portions of C assigned to all cultivations in all districts must be equal to C. A similar condition must hold for N.

$$\sum_{\forall i,j} v_{\mathtt{ij}} = C \qquad \sum_{\forall i,j} w_{\mathtt{ij}} = N \tag{4.4}$$

In order to connect the y variables to v, we need to impose that it is not possible assigning a portion of C to cultivation i (regardless to the district) when the corresponding variable y_i is 0. A similar condition must hold for to connect the z variables to w. Note that M is a constant value greater than all possible left-hand-side values.

$$v_{\mathtt{ij}} \leq M y_i \qquad \forall i = 1, \ldots, n \ \ \forall j = 1, \ldots, m \tag{4.5}$$

$$w_{\mathtt{ij}} \leq M z_i \qquad \forall i = 1, \ldots, n \ \ \forall j = 1, \ldots, m \tag{4.6}$$

The whole C must be assigned to only one cultivation. A similar condition must hold for N.

$$\sum_{\forall i} y_i = 1 \qquad \sum_{\forall i} z_i = 1 \tag{4.7}$$

The above constraints have the effect of letting only one y (only one z) be 1, and so the previous constraints can only assigning C (respectively N) to a unique cultivation.

Finally, an assignment for C or for N cannot be accepted when the likelihood of that assignment, although being the greatest possible for the current problem, is too low. In such a case, indeed, that assignment cannot be considered reliable. For this reason we introduce, in the following constraints, two thresholds, denoted by SC and SN, respectively for the assignments made on C and N.

$$\sum_{\forall j} v_{\mathtt{ij}} - \sum_{\forall j} p_{\mathtt{ij}} v_{\mathtt{ij}} - SC \leq M(1 - y_i) \qquad \forall i = 1, \ldots, n \tag{4.8}$$

$$\sum_{\forall j} w_{\mathtt{ij}} - \sum_{\forall j} p_{\mathtt{ij}} w_{\mathtt{ij}} - SN \leq M(1 - z_i) \qquad \forall i = 1, \ldots, n \tag{4.9}$$

In the constraints (4.8), the assignment of C can be possible if the likelihood of assigning C to cultivation i is good (= near to 1) for the different districts where C have been located, that means y_i can assume value 1. On the other hand, when that likelihood is not good (= near to 0), that assignment is not allowed, that means y_i must be forced to value 0. Note that, if no assignment has a sufficient likelihood, those constraints cannot be satisfied and the model correctly becomes infeasible.

The above is obtained because, for assignments having good likelihood, the value of $\sum_{\forall j} p_{\mathtt{ij}} v_{\mathtt{ij}}$ is only a bit smaller than the value of $\sum_{\forall j} v_{\mathtt{ij}}$, and by subtracting SC the left-hand-side of the inequality (4.8) becomes smaller than or equal to 0, leaving y_i free.

When on the contrary the likelihood is not good, the value of $\sum_{\forall j} p_{\mathtt{ij}} v_{\mathtt{ij}}$ is much smaller than the value of $\sum_{\forall j} v_{\mathtt{ij}}$, and even subtracting SC (whose reasonable value is therefore just a fraction of $\sum_{\forall j} v_{\mathtt{ij}}$, for instance one half) the left-hand-side of the inequality (4.8) becomes positive. As a consequence, $M(1 - y_i)$ must have a strictly positive value, and so y_i must have value 0.

In the constraints (4.9) the logic of the assignment of N is the same as that previously described for C in the the constraints (4.9).

On the whole, the complete mixed integer linear programming model for assigning the unassigned area of a single farm is the following:

$$\left\{\begin{array}{lll}
\max \sum_{\forall i,j} p_{\mathtt{ij}} x_{\mathtt{ij}} + \sum_{\forall i,j} p_{\mathtt{ij}} v_{\mathtt{ij}} + \sum_{\forall i,j} p_{\mathtt{ij}} w_{\mathtt{ij}} & & \\
\sum_{\forall i} x_{\mathtt{ij}} + \sum_{\forall i} v_{\mathtt{ij}} + \sum_{\forall i} w_{\mathtt{ij}} \ = \ d_j & \forall j = 1, \ldots, m & \\
\sum_{\forall j} x_{\mathtt{ij}} = s_i & \forall i = 1, \ldots, n & \\
\sum_{\forall i,j} v_{\mathtt{ij}} = C \qquad \sum_{\forall i,j} w_{\mathtt{ij}} = N & & \\
v_{\mathtt{ij}} \leq M y_i & \forall i = 1, \ldots, n & \forall j = 1, \ldots, m \\
w_{\mathtt{ij}} \leq M z_i & \forall i = 1, \ldots, n & \forall j = 1, \ldots, m \\
\sum_{\forall i} y_i = 1 \qquad \sum_{\forall i} z_i = 1 & & \\
\sum_{\forall j} v_{\mathtt{ij}} - \sum_{\forall j} p_{\mathtt{ij}} v_{\mathtt{ij}} - SC \leq M(1 - y_i) & \forall i = 1, \ldots, n & \\
\sum_{\forall j} w_{\mathtt{ij}} - \sum_{\forall j} p_{\mathtt{ij}} w_{\mathtt{ij}} - SN \leq M(1 - z_i) & \forall i = 1, \ldots, n & \\
x_{\mathtt{ij}} \geq 0 & \forall i = 1, \ldots, n & \forall j = 1, \ldots, m \\
v_{\mathtt{ij}} \geq 0 & \forall i = 1, \ldots, n & \forall j = 1, \ldots, m \\
w_{\mathtt{ij}} \geq 0 & \forall i = 1, \ldots, n & \forall j = 1, \ldots, m \\
x_{\mathtt{ij}} \in \mathbb{R} & \forall i = 1, \ldots, n & \forall j = 1, \ldots, m \\
v_{\mathtt{ij}} \in \mathbb{R} & \forall i = 1, \ldots, n & \forall j = 1, \ldots, m \\
w_{\mathtt{ij}} \in \mathbb{R} & \forall i = 1, \ldots, n & \forall j = 1, \ldots, m \\
y_i \in \{0, 1\} & \forall i = 1, \ldots, n & \\
z_i \in \{0, 1\} & \forall i = 1, \ldots, n &
\end{array}\right. \tag{4.10}$$

4.3 Computational Results

The requested Information Reconstruction process has been performed by solving the above model for each farm sequentially. This procedure was implemented in C++, using ILOG Concert Technology [66] in order to express the described optimization models. The models themselves are solved by means of the state-of-the-art branch-and-cut [10, 76] procedure implemented by the solver ILOG Cplex [65], running on a 16 cores server having 128Gb of RAM and Linux Operating System. The resulting software system has been applied for the treatment of data from the Italian Census of Agriculture 2010, with specific respect to the case of vineyards. This is probably the most important case for the considered problem, since dozens of vine varieties exist, and they determine the type and the quality of wines produced. The case has therefore great economic relevance and, due to its large dimension, is also computationally demanding. Moreover, the total area that a region uses for each vine type determines the European Community funding obtained for that region. Note that, to the best of our knowledge, no previous attempt to treat this problem with a discrete optimization approach was practically successful. Clearly, the proposed model is not limited to the case of vineyards, but can be used for any other problem sharing the same characteristics.

The practical behavior of the proposed procedure has been evaluated both from the computational and from the data quality points of view, as follows. One large dataset including all farms producing vine from all Italian regions has been assembled. This dataset included 388,487 farms, and the total number of vineyard areas declared was 804,930, corresponding to a total area of 625,700 ha (hectares, 1 hectare being 10,000 m^2). Each vineyard declaration constitutes a record, so the dataset is considerably large. After this, the cultivation declarations where checked by means of rules, and an unassigned area resulted for 18,263 farms. The total unassigned area was 34,783 ha, with 30,226 ha suitable for "controlled origin" and 4,557 ha not suitable for "controlled origin". As remarked in Section 4.1, thresholds were assigned in order to reject low likelihood cultivation assignments, in particular $SC = 1/2 \sum_{\forall j} v_{\mathtt{ij}}$ and $SN = 1/2 \sum_{\forall j} w_{\mathtt{ij}}$. In order to evaluate the computational behavior, Table 4.1 reports regional detail, that are: the total unassigned area for each region, and the corresponding computational times for processing the whole region. As observable, the procedure is very fast, and the processing of all the Italian farms having an unassigned area requires only about 17 minutes.

The quality of the reconstructed information has been evaluated by considering: (i) the ability to assign the unassigned area; and (ii) the variation produced in the data by the reconstruction process. As for the first aspect, the procedure was able to assign the unassigned area with likelihood values high enough to satisfy the thresholds SC and SN in the totality of the cases (100%). As for the second aspect, Table 4.2 reports an analysis of the percentages of the two big groups of vine cultivations (red and white) on the initial data, i.e. before applying the described reconstruction process, and on the final data, i.e. after the reconstruction process, followed by the computation of the variation introduced by this process. Note that, in the general case of the correction of a survey, a small variation of the frequency distributions of the data means that the reconstruction procedure was able

Region	Total unassigned area (ha)	Times (sec.)
Piemonte	42.89	0.82
Valle d'Aosta	10.08	1.98
Lombardia	271.19	34.61
Veneto	12510.09	165.75
Friuli-Venezia Giulia	87.94	13.95
Liguria	52.87	15.60
Emilia-Romagna	20.24	0.44
Toscana	8293.34	160.14
Umbria	100.75	14.50
Marche	2517.57	65.76
Lazio	263.03	38.02
Abruzzo	259.62	34.39
Molise	397.68	22.80
Campania	511.53	86.31
Puglia	8939.25	303.37
Basilicata	13.09	0.66
Calabria	331.33	24.45
Sicilia	49.41	1.15
Sardegna	8.35	0.49
Bolzano	0.88	0.05
Trento	101.47	22.74
Italy total	34782.60	1008.00

Table 4.1: Total correction times for each region and for all Italy

to reconstruct information without distorting the data, so it is a positive feature. In this case, as observable from the Table, the variation value has been very small, so the quality of the reconstructed information is extremely satisfactory. Moreover, in the same Table the described percentages computed only over the farms not having unassigned areas ("Exact only") is reported. Those percentages are again very similar to the percentages after reconstruction, confirming the high quality of the reconstructed information.

4.4 Summary and Remarks

A typical problem arising in an Agricultural Census is assigning the correct cultivations to the area for which the farm declarations are considered unreliable. This assignment should take into account the compatibility between each district and the type of cultivation. The problem is extremely important due to its economical and normative aspects, and is also computationally demanding due to its large dimension. This Chapter presents

	Before Reconstruction	After Reconstruction	Variation	Exact only
White	43.55%	42.84%	-0.71%	43.95%
Red	56.45%	57.16%	+0.71%	56.05%

Table 4.2: Analysis of frequency distributions before and after the reconstruction process

an automatic approach to this information reconstruction problem based on the formulation of mixed integer linear programming models. The proposed procedure has been applied in the specific case of vineyards of the Italian Census of Agriculture 2010. In this important case there may be areas with missing or erroneous vineyard codes. For these areas, it is required to assign a code, according to a set of consistency constraints. Results are very encouraging both form the computational and from the data quality point of view. The sequence of arisen mixed integer programming problems can be solved to optimality by using state-of-the-art implementation of branch-and-cut procedures. Each single model is solved to optimality in extremely short times (generally about 0.1 sec.). In the totality of the cases the reconstructed information was able to satisfy the thresholds introduced to reject low likelihood cultivation assignments. Moreover, the reconstruction process did not distort data, as resulted from the analysis of the variations introduced in the frequency distributions of the whole dataset. The methodology described in this Chapter can again be used to solve other problems of different origin but having the same mathematical structure.

Chapter 5

A Formal Procedure for Finding Contradictions into a Set of Rules

5.1 The Problem of Contradictions Localization

In several fields of knowledge, many tasks are accomplished by using sets of expressions called *rules* (see e.g. [47]). Rules are typically used do detect, among a possibly large set of elements, the ones verifying some condition. This happens for example in Data Mining, in Database Theory, in Statistics, but also in less mathematical fields such as Normative or Regulation. The condition may be of any nature, for instance "being correct", "being wrong", "being convenient", "respecting the laws", "being compliant with a standard", etc. The set of rules may have several origins: it could be automatically generated, for instance learned by some dataset, or be written by human experts, or also be the result of an updating or a merging of other sets of rules. A major issue is the presence of contradictions into the set of rules itself. This can frequently arise, in particular when the set of rules has been assembled from different sources. Generally, the presence of contradictions makes such a set not usable anymore. Each contradiction should therefore be located and removed, either by deleting or by slightly changing some of the rules. This is however a very difficult problem in general: a contradiction can be quite hidden, or involve many rules, or there can be several contradictions. Moreover, this difficulty rapidly increases with the size of the set of rules [52]. See also [25, 28] for related work on inconsistencies selection.

This Chapter presents an automatic procedure for finding a contradiction into a set of rules. The procedure can be iterated until all contradictions are removed from a set. A main advantage of the proposed approach is that this procedure works only at the formal level, so it can be performed without the need of going into the semantic meaning of the rules under analysis and can be applied to rules arising from any field. In particular, Section 5.2 explains how several kind of rules can be formally represented into linear inequalities. After this, Section 5.3 presents a theoretical condition, based on a variant of Farkas' lemma (see e.g. [88]), used to detect a single contradiction. All contradictions are detected by iterating this procedure, and the structure of the set of all contradictions, together with the relationships among themselves, are also studied. Finally, Section 5.4 gives a detailed explanation of the operations performed by the proposed procedure on a realistic set of rules. This work has been published in [12].

5.2 Encoding Rules into Linear Inequalities

In Database theory, a *record schema* is a set of fields f_i, with $i = 1 \dots m$, and a *record instance* is a set of values v_i, one for each of the above fields. In order to help exposition, we will focus on records representing *persons*. Note, however, that the proposed procedure is not influenced by the meaning of processed data. The record scheme will be denoted by P, whereas a generic record instance corresponding to P will be denoted by p.

$$P = \{f_1, \dots, f_m\} \qquad p = \{v_1, \dots, v_m\}$$

Example 5.1. For records representing persons, fields are for instance `age` or `marital status`, and corresponding examples of values are `18` or `single`.

Each field f_i, with $i = 1 \dots m$, has its *domain* D_i, which is the set of every possible value for that field. A distinction is usually made between *quantitative*, or *numerical*, fields, and *qualitative*, or *categorical*, fields. The proposed approach is able to deal with both qualitative and quantitative values.

In several applications, records verifying some condition are selected by using *rules*. As known in Section 1.4, each rule can be seen as a mathematical function r_k from the Cartesian product of all the domains to the Boolean set {0,1}, as follows.

$$\begin{array}{rccc} r_k : & D_1 \times \cdots \times D_m & \to & \{0,1\} \\ & p & \mapsto & 0,1 \end{array}$$

We call *logical rules* the rules expressed only with logical conditions, *mathematical rules* the rules expressed only with mathematical conditions, and *logic-mathematical rules* the rules expressed using both types of condition. See [26] for further details on different kind of rules.

Values appearing in the rules are called *breakpoints*, or *cut points*, for the domains. They represent the logical *watershed* between values of the domain, and will be indicated with b_i^j. Such breakpoints are used to split every domain D_i into n_i subsets S_i^j representing values of the domain which are *equivalent* from the rules' point of view. We congruently have $D_i = \bigcup_{j=1}^{n_i} S_i^j$.

Example 5.2. Suppose that, by scanning a given set of rules R, the following breakpoints are obtained for the field `age` of a person.

$$b^1_{\texttt{age}} = \texttt{0},\ b^2_{\texttt{age}} = \texttt{14},\ b^3_{\texttt{age}} = \texttt{18},\ b^4_{\texttt{age}} = \texttt{26},\ b^5_{\texttt{age}} = \texttt{110},\ b^6_{\texttt{age}} = \texttt{blank}$$

and, by using the breakpoints and the rules to cut $D_{\texttt{age}}$, we have the $n_{\texttt{age}} = 5$ subsets. The last subset is the out-of-range one.

$$\begin{gathered} S_{\texttt{age}\in\{0...13\}} = \{\texttt{0},\dots,\texttt{13}\},\ S_{\texttt{age}\in\{14...17\}} = \{\texttt{14},\dots,\texttt{17}\}, \\ S_{\texttt{age}\in\{18...25\}} = \{\texttt{18},\dots,\texttt{25}\},\ S_{\texttt{age}\in\{26...110\}} = \{\texttt{26},\dots,\texttt{110}\}, \\ S_{\texttt{age} = \texttt{out\_of\_range}} = \{\dots,-\texttt{1}\} \cup \{\texttt{111},\dots\} \cup \{\texttt{blank}\} \end{gathered}$$

Now, the *variables* for the announced linear inequalities can be introduced: a set of m real variables $z_i \in [0, U]$, one for each domain D_i, and a set of $n = n_1 + \cdots + n_m$ binary variables $x_{ij} \in \{0, 1\}$, one for each subset S_{ij}. We represent each value v_i of p with a real variable z_i, by defining a mapping φ between values of the domain and real numbers between 0 and an upper value U. Note that, occasionally, it could be convenient to bound some of the z_i variables to be integer, as described in [26], with obvious specific modifications in the rest of the procedure. However, in this description we consider the general case of real z variables.

The membership of a value v_i to the subset S_{ij} is encoded by using the binary variables x_{ij}.

$$x_{ij} = \begin{cases} 1 & \text{when} \quad v_i \in S_{ij} \\ 0 & \text{when} \quad v_i \notin S_{ij} \end{cases}$$

Binary and real variables are linked by using a set of linear inequalities called *bridge constraints*. They impose that, when z_i has a value such that v_i belongs to subset S_{ij}, the corresponding x_{ij} is 1 and all others binary variables $\{x_{i1}, \ldots, x_{ij-1}, x_{ij+1}, \ldots, x_{in_i}\}$ of field f_i are 0. By using these variables, all the above types of rule can be expressed. For further details see [26, 27].

Example 5.3. Consider the following logical rule.

$$\neg(\texttt{marital status} = \texttt{married}) \vee \neg(\texttt{age} < \texttt{14})$$

By substituting the logical conditions, it becomes the linear inequality:

$$(1 - x_{\texttt{marital\_status = married}}) + (1 - x_{\texttt{age}\in\{0\ldots13\}}) \geq 1$$

Consider, instead, the following logic-mathematical rule.

$$\neg(\texttt{marital status} \;=\; \texttt{married}) \vee (\texttt{age} - \texttt{years married} \geq \texttt{14})$$

By substituting the logical and mathematical conditions, we have

$$(1 - x_{\texttt{marital status = married}}) \vee (z_{\texttt{age}} - z_{\texttt{years married}} \geq 14)$$

which becomes the following linear inequality

$$U(1 - x_{\texttt{marital status = married}}) + z_{\texttt{age}} - z_{\texttt{years married}} \geq 14$$

Altogether, from the set of rules R, a set of linear inequalities is obtained. Each record p determines an assignment of values for the introduced variables x_{ij} and z_i. By denoting with x and z the vectors respectively made of all the components x_{ij} and z_i, $i = 1 \ldots m$, $j = 1 \ldots n_i$, as follows,

$$x = (x_{11}, \ldots, x_{1n_1}, \ldots, x_{m1}, \ldots, x_{mn_m})^T \quad z = (z_1, \ldots, z_m)^T$$

the set of rules R becomes a system of linear inequalities, expressed in compact notation as follows.

$$\begin{cases} B \begin{bmatrix} x \\ z \end{bmatrix} \geq b \\ 0 \leq z_i \leq U \qquad i = 1 \ldots m \\ x \;\in\; \{0,1\}^n \\ z \;\in\; \mathbb{R}^m \end{cases} \tag{5.1}$$

Since x has $n = n_1 + ... + n_m$ components and z has m components, and letting l be the total number of inequalities, B is in general a $l \times (n + m)$ real matrix, and b a real l-vector.

5.3 Locating Contradictions by Means of Linear Programming

A contradiction in the set of rules corresponds to an unsatisfiable set of inequalities within the above described system of linear inequalities. Such an unsatisfiable set is called *Infeasible Subsystem* (IS). When an IS is minimal, i.e. becomes satisfiable by removing anyone of its inequalities, is called *Irreducible Infeasible Subsystem*(IIS) [3, 35, 90]. In the case of systems of linear inequalities having real variables, the problem has been approached both by means of heuristics [34] and exact algorithms [55]. In the case of systems of linear inequalities having integer variables (more computationally demanding), the problem has been approached by means of additive or subtractive heuristics [56]. This Chapter presents a procedure for solving this problem based on a variant of well known Farkas' lemma adapted from the continuous to the discrete case.

Theorem 5.1. `(Farkas' lemma)` Let A be an $s \times t$ real matrix and let a be a real s-vector. Then there exists a real t-vector $x \geq 0$ with $Ax = a$ if and only if $y^T a \geq 0$ for each real s-vector y with $y^T A \geq 0$.
Geometrically, this means that if an s-vector γ does not belong to the cone generated by the s-vectors $a_1, \dots, a_t$ (columns of A), there exists a linear hyperplane separating γ from $a_1, \dots, a_t$. There are several equivalent forms of Farkas' lemma. The following variant is more suitable to our purposes. Given a matrix $A \in \mathbb{R}^{s \times t}$ and a vector $a \in \mathbb{R}^s$, consider the system:

$$\begin{cases} Ax & \leq & a \\ x & \in & \mathbb{R}^t \end{cases} \tag{5.2}$$

and the new system of linear inequalities obtained from the former one:

$$\begin{cases} y^T A & = & \mathbf{0} \\ y^T a & < & 0 \\ y & \geq & \mathbf{0} \\ y & \in & \mathbb{R}^s \end{cases} \tag{5.3}$$

We have that exactly one of the two following possibilities holds:

- (5.2) is feasible, i.e. there exists $x \in \mathbb{R}^t$ verifying all its inequalities.
- (5.3) is feasible, i.e. there exists $y \in \mathbb{R}^s$ verifying all its inequalities.

An IIS can be selected within (5.2) by solving the following new system [55]:

$$\begin{cases} y^T A &= \mathbf{0} \\ y^T a &\le -1 \\ y &\ge \mathbf{0} \\ y &\in \mathbb{R}^s \end{cases} \tag{5.4}$$

The *support* of a vertex denotes the indices of its non-zero components; **0**, **1** and **U** respectively denote vectors of zeros, ones and Us of appropriate dimension.

Theorem 5.2. `(Gleeson and Ryan)` Consider two systems of linear inequalities respectively in form (5.2) and (5.4). If (5.4) is infeasible, (5.2) is feasible. On the contrary, if (5.4) is feasible, (5.2) is infeasible, and, moreover, each IIS of (5.2) is given by the support of each vertex of the polyhedron (5.4).

The proof is based on polyhedral arguments using properties of extreme rays, see [55]. Therefore, checking the feasibility of (5.2), and, if infeasible, identifying one of its IIS, becomes the problem of finding a vertex of a polyhedron, that can be easily solved (e.g. with the simplex algorithm [10, 88]).

However, in the case of (5.1), we have a systems of linear inequalities were we are interested in mixed integer solutions. In order to use the results given for the linear case, let us consider the linear relaxation of such system (5.1).

$$\begin{cases} -B\begin{bmatrix} x \\ z \end{bmatrix} &\le -b \\ \begin{bmatrix} x \\ z \end{bmatrix} &\le \begin{bmatrix} \mathbf{1} \\ \mathbf{U} \end{bmatrix} \\ -\begin{bmatrix} x \\ z \end{bmatrix} &\le \mathbf{0} \\ \begin{bmatrix} x \\ z \end{bmatrix} &\in \mathbb{R}^{n+m} \end{cases} \tag{5.5}$$

The above system (5.5) is now in the form of (5.2). The l inequalities from the first group will be called *rules inequalities*, even if, for some of them, there can be no one-to-one correspondence with rules (see Sect. 5.4). By denoting with I the identity matrix, the $[l + 2(n + m)] \times (n + m)$ matrix A and the $[l + 2(n + m)]$-vector a are composed as follows. Number of rows for each block is reported on the left.

$$A = \begin{array}{c} \left[\begin{array}{c} -B \\ I \\ -I \end{array}\right] \end{array} \begin{array}{c} l \\ n+m \\ n+m \end{array} \qquad a = \left[\begin{array}{c} -b \\ \mathbf{1} \\ \mathbf{U} \\ \mathbf{0} \end{array}\right] \begin{array}{c} l \\ n \\ m \\ n+m \end{array}$$

Therefore, a system which plays the role of (5.4) can now be written.

$$\left\{ \begin{array}{lcl} y^T \left[\begin{array}{c} -B \\ I \\ -I \end{array}\right] & = & \mathbf{0} \\ y^T \left[\begin{array}{c} -b \\ \mathbf{1} \\ \mathbf{U} \\ \mathbf{0} \end{array}\right] & \leq & -1 \\ y \ \geq \ \mathbf{0}, \quad y \in \ \mathbb{R}^{[l+2(n+m)]} \end{array} \right. \tag{5.6}$$

So far, the following result on the pair of systems (5.1) and (5.6) holds. The *restriction* of the support of a vertex to rules inequalities will denote the indices of its non-zero components among those corresponding to rules inequalities.

Theorem 5.3. Consider two systems of linear inequalities respectively in form (5.1) and (5.6). In this case, if (5.6) is feasible, (5.1) is infeasible, and the restriction of the support of each vertex of the polyhedron (5.6) to rules inequalities contains an IIS of (5.1). On the contrary, if (5.6) is infeasible, (5.5) is feasible, but it cannot be decided whether (5.1) is feasible or not.

Proof: We first prove that the restriction of the support of a vertex of (5.6) to rule inequalities contains an integer IIS of (5.1). Assume (5.6) is feasible, and let v_1 be the vertex found. Therefore, (5.5) is infeasible (from Theorem 5.1), and an IIS in (5.5), called in this Chapter IIS_1, is given by the support of v_1. Such IIS_1 is in general composed by a set RI_1 of *rules inequalities* and a set BC_1 (possibly empty) of *box constraints* (the ones imposing $0 \leq x_{ij} \leq 1, 0 \leq z_i \leq U$). The set of inequalities RI_1 has no integer solutions, since removing the BC_1 from IIS_1, while imposing the more strict *integer constraints* IC_1 (the ones imposing $x_{ij} \in \{0,1\}$), keeps IIS_1 unsatisfiable. Therefore, an integer IIS is contained into RI_1. The integer IIS may also be a subset of the inequalities of RI_1, because, though $IIS_1 = RI_1 \cup BC_1$ is minimally infeasible, $RI_1 \cup IC_1$ may be not minimal: we are imposing the more strict integer constraints instead of the box constraints. Therefore, the procedure produces an integrally infeasible subsystem containing an integer IIS for (5.1).

On the other hand, not all integer IIS in (5.2) can be obtained by such procedure. This because, if (5.6) is infeasible, (5.5) is feasible (by Theorem 5.1). When imposing the more strict integer constraints instead of the box constraints, however, nothing can be said on the feasibility of (5.1).

Example 5.4. Consider a set of rules R on two conditions α_1, α_2, as follows. One may already note that R contains an inconsistency.

$$r_1 = (\alpha_1),\ r_2 = (\alpha_2),\ r_3 = (\neg\alpha_1 \vee \neg\alpha_2),\ r_4 = (\alpha_1 \vee \neg\alpha_2)$$

In this case, $n = 2$ and m can be considered 0, since no z variables are needed to express the above rules. A and a can easily be obtained, as follows.

$$A = \left[\begin{array}{rr} -1 & 0 \\ 0 & -1 \\ 1 & 1 \\ -1 & 1 \\ \hline 1 & 0 \\ 0 & 1 \\ \hline -1 & 0 \\ 0 & -1 \end{array}\right] \qquad a = \left[\begin{array}{r} -1 \\ -1 \\ 1 \\ 0 \\ \hline 1 \\ 1 \\ \hline 0 \\ 0 \end{array}\right]$$

Therefore, the system to be solved, in the form of (5.6), is the following.

$$\left\{\begin{array}{rcl} -y_1 + y_3 - y_4 + y_5 - y_7 & = & 0 \\ -y_2 + y_3 + y_4 + y_6 - y_8 & = & 0 \\ -y_1 - y_2 + y_3 + y_5 + y_6 & \leq & -1 \\ y_1,\ y_2,\ y_3,\ y_4,\ y_5,\ y_6,\ y_7,\ y_8 & \geq & 0 \\ y & \in & \mathbb{R}^8 \end{array}\right.$$

Solving such system yields the vertex (1, 1, 1, 0, 0, 0, 0, 0). Therefore, R contains an inconsistency, and the set of conflicting rules is $\{r_1, r_2, r_3\}$.

More than one IIS can be contained in an infeasible system. Some of them can overlap, in the sense that they can share some inequalities, although they cannot be fully contained one in another. Formally, the collection of all IIS of a given infeasible system is a *clutter* (see e.g. [3]). However, from the practical point of view, we are interested in IIS composed by a small number of rules inequalities. Moreover, it may happen that not all of them are equally preferable for the composition of the IIS that we are selecting.

Hence, a cost c_k for taking each of the $[l + 2(n + m)]$ inequalities into our IIS can be assigned. Such costs c_k for the inequalities of (5.5) corresponds to costs for the variables of system (5.6). A cost $[l + 2(n + m)]$-vector c is therefore computed, and the solution of the following linear program produces now an IIS having the desired inequality composition.

$$\begin{cases} \min\ c^T y \\ y^T \begin{bmatrix} -B \\ I \\ -I \end{bmatrix} = \mathbf{0} \\ y^T \begin{bmatrix} -b \\ \mathbf{1} \\ \mathbf{U} \\ \mathbf{0} \end{bmatrix} \leq -1 \\ y \geq \mathbf{0}, \quad y \in \mathbb{R}^{[l+2(n+m)]} \end{cases} \tag{5.7}$$

The result of Theorem 5.3 is not completely analogous to the linear case. In order to obtain more analogy, let us define the following property.

Integral-point property. A class of polyhedra which, if non-empty, contain at least one integral point (i.e. a point respecting integrality constraints) has the integral-point (IP) property.

Theorem 5.4. If the polyhedron (5.5), which is the linear relaxation of (5.1), has the integral-point property, the following holds. If (5.6) is infeasible, (5.1) is feasible. On the contrary, if (5.6) is feasible, (5.1) is infeasible and each integer IIS is given by the restriction of the support of each vertex of polyhedron (5.6) to rules inequalities.

Proof: If (5.6) is infeasible, (5.5) is feasible by Theorem 5.1. Since we assumed that the IP-property holds for (5.5), it contains at least one integral point. Since the box constraints hold for (5.5), this integer point must be such that $x \in \{0, 1\}^n$, hence (5.1) is feasible. On the contrary, if (5.6) is feasible, the restriction of the support of a vertex in (5.6) to rule inequalities, that is a set of inequalities denoted by RI_1, has no integer solutions by Theorem 5.3. We now prove by contradiction that RI_1 is minimally infeasible, hence it is an integer IIS. Suppose RI_1 not minimal; then there exists a smaller set RI'_1 such that $RI'_1 \cup IC_1$ has no integer solutions. On the other hand, by Theorem 5.2, $RI_1 \cup BC_1$ is minimal, so $RI'_1 \cup BC_1$ must be feasible, and since it has the IP-property, it has an integer solution, which is the contradiction. The thesis follows.

So far, when the IP property holds, solving a linear programming problem solves our inconsistency selection problem. There are several cases in which the linear relaxation (5.5) defines a polyhedron having the integral-point property (see e.g. [30, 36, 76]). Note that, imposing some syntactic restrictions, rules could be written in order to obtain one of such cases.

5.4 Applying the Proposed Procedure

Assume that each individual is described by a data record (a set of values for a set of fields). Let the fields be either categorical, e.g. *name, profession, tax1* (= if the individual has to pay a tax called tax1), *tax2*, *tax3*, or numerical, e.g. *age, length_of_career, income.*

Let the domain of *profession* be a set of strings (e.g. pr1, pr2, pr3); *blank* being an admissible value, e.g. for non-working people); the domain of *tax1, tax2, tax3* be $\{yes, no\}$; the domain of *age* be a suitable subset of the set of real non-negative numbers $\mathbb{R}_+$ (or of Z_+, with obvious modifications); the domain of *length_of_career* be a suitable subset of $\mathbb{R}_+ \cup blank$ (*blank* being an admissible value, e.g. for non-working people); the domain of *income* be a suitable subset of R_+ (being 0 for non-working people).

Assume there is a set of rules for economical regulation (something similar to laws), as follows. Clearly, the focus is not on numerical values appearing in the rules, that may be unrealistic, but on the structure of the set. Note that, in order to test the consistency of this set, we need to consider also rules that a human would consider obvious, but not a machine, called *unexpressed* rules.

- *Logical* rules

  Some taxes must be paid for some professions

L1 if *profession* = (*pr1* or *pr2*) then *tax1* must be *yes*

L2 if *profession* = *pr3* then *tax2* must be *yes*

Some taxes must be paid for some income values

L3 if *income* $\geq$ 1000 then *tax3* must be *yes*

For poor people taxes cannot exceed 100

L4 if *income* ≤ 200 then *total_tax* must be ≤ 100

- *Mathematical* rules

  Income must be related to length of career

M1 $\mathit{income} \leq 1000 + 20 \times \mathit{length\_of\_career}$

M2 $\mathit{income} \geq 200 + 30 \times \mathit{length\_of\_career}$

Taxes must be at least one third of the income

M3 $\mathit{total\_tax} \geq 0.33 \times \mathit{income}$

Taxes cannot exceed income

M4 $\mathit{total\_tax} \leq \mathit{income}$

- *Logico-mathematical* rules

  If income is too high for the career, tax 3 must me paid

LM1 if $\mathit{income} - 30 \times \mathit{length\_of\_career} \geq 400$ then *tax3* must be *yes*

- *Unexpressed* rules

  Professions are mutually exclusive

U1 $\mathit{Pr1} \oplus \mathit{Pr2} \oplus \mathit{Pr3}$

There are relations implied by the meaning of the words

U2 $\mathit{total\_tax} = \mathit{tax1} + \mathit{tax2} + \mathit{tax3}$

Some Fields are naturally limited

U3 *age* ≥ 0 and ≤ 110

U4 *length_of_career* ≥ 0 and ≤ 92

U5 $\varepsilon \geq 0$ and ≤ 0.001

U6 *total_tax* ≥ 0 and ≤ 2000

U7 *income* ≥ 0 and ≤ 5000

From the above rules we can identify some *variables*. Some of them are logical, and are also called propositions, and some are real-valued.

1) XPRO1 (binary)

2) XPRO2 (binary)

3) XPRO3 (binary)

4) XTAX1 (binary)

5) XTAX2 (binary)

6) XTAX3 (binary)

7) XTTAX0-100 (binary)

8) XINC0-200 (binary)

9) TTAX (real≥ 0)

10) INC (real≥ 0)

11) AGE (real≥ 0)

12) LEN (real≥ 0)

13) EPS (real≥ 0)

In the general case, from the rules we can identify some logical propositions, that are the elementary concepts expressed in the rules. We may have:

- *Level* propositions, e.g. $L1, L2, L3, L4$. They are conditions that become stronger as their index increases, so $L4 \Rightarrow L3, L2, L1$ and $L3 \Rightarrow L2, L1$ and $L2 \Rightarrow L1$ and $L1$ does not imply anything. Conversely, $\neg L1 \Rightarrow \neg L2, \neg L3, \neg L4$ and $\neg L2 \Rightarrow \neg L3, \neg L4$ and $\neg L3 \Rightarrow \neg L4$ and $\neg L4$ does not imply anything. A set of level propositions is *complete* when at least one of them must hold, so $L1$ is always true.

  They can represent for instance that the value of a certain field of some data records belongs to some sets $S1, S2, S3, S4$ in a domain S such that $S1 \supseteq S2 \supseteq S3 \supseteq S4$ (and are complete when $S1 = S$).

- *Exclusive* propositions, e.g. $E1, E2, E3$. They are mutually exclusive: at most one of them holds, so $E1 \Rightarrow \neg E2, \neg E3$ and $E2 \Rightarrow \neg E1, \neg E3$ and $E3 \Rightarrow \neg E1, \neg E2$. Equivalently, $\neg E1 \vee \neg E2$ and $\neg E2 \vee \neg E3$ and $\neg E1 \vee \neg E3$. A set of exclusive propositions is *complete* when at least one of them must hold, so $E1 \vee E2 \vee E3$.

  They can represent for instance that the value of a certain field of some data records belongs to some sets $S1, S2, S3$ such that $S1 \cap S2 = \phi$ and $S2 \cap S3 = \phi$ and $S1 \cap S3 = \phi$ (complete when $S1 \cup S2 \cup S3 = S$).

- *Standard* propositions, e.g. F, G, H, I. They have no predefined relations among them, and any relation among them can be expressed, e.g. $F \Rightarrow G$ and $F \wedge H \Rightarrow I$.

The rules may contain one or more inconsistencies, as explained in Section 5.3. Note that inconsistencies may be either *complete*, when no record can respect the rules, or *partial*, when no record having a specific value v_i for a specific field i (value that should not be forbidden) can respect the rules. In this example we have:

- No complete inconsistency: there are records respecting all the rules.

- A partial inconsistency for *length_of_career* ≥ 67
  (M2 says *income* ≥ 2210 and M3 says *total_tax* ≥ 729.3, while U2 says *total_tax* can be at most 720 when all *tax1, tax2, tax3* are paid. Since U7 says $0 \leq$ *income* ≤ 5000, that is a contradiction).

- Another partial inconsistency for *length_of_career* ≥ 81
  (M1 says *income* ≤ 2620 while M2 says *income* ≥ 2630, that is a contradiction).

- Another partial inconsistency for *income* ≥ 2182
  (M3 says *total_tax* ≥ 720.06, while U2 says *total_tax* can be at most 720 when all *tax1, tax2, tax3* are paid. Since U7 says $0 \leq$ *income* ≤ 5000, that is a contradiction).

Partial inconsistencies can be tested with the proposed procedure by simply imposing the value activating them, for instance by adding a constraint. We now analyze the above three examples with our procedure. First we convert rules into inequalities, until putting all of them in the form $\leq$

L1 if *profession* = (*pr1* or *pr2*) then *tax1* must be *yes*

= $\neg$ XPRO1 $\vee$ XTAX1 and $\neg$ XPRO1 $\vee$ XTAX1

= XTAX1 + (1-XPRO1) $\geq$ 1 and XTAX1 + (1-XPRO2) $\geq$ 1

1) -1 XTAX1 +1 XPRO1 $\leq$ 0

2) -1 XTAX1 +1 XPRO2 $\leq$ 0

L2 if *profession* = *pr3* then *tax2* must be *yes*

= XTAX2 + (1-XPRO3) $\geq$ 1

3) -1 XTAX2 +1 XPRO3 $\leq$ 0

L3 if *income* $\geq$ 1000 then *tax3* must be *yes*

= $\neg tax3 \Rightarrow income < 1000$

= $tax3 \vee income \leq 1000 - \varepsilon$

= -M TAX3 +INC +EPS $\leq$ 1000

4) -M TAX3 +1 INC +1 EPS $\leq$ 1000

L4 if *income* $\leq$ 200 then *total_tax* must be $\leq$ 100

= (1-XINC0-200) + XTTAX0-100 $\geq$ 1

5) 1 XINC0-200 -1 XTTAX0-100 $\leq$ 0

M1 $income \leq 1000 + 20 \times length\_of\_career$

= INC -20 LEN $\leq$ 1000

6) 1 INC -20 LEN $\leq$ 1000

M2 $income \geq 200 + 30 \times length\_of\_career$

= INC -30 LEN $\geq$ 200

7) -1 INC +30 LEN ≤ -200

M3 $total\_tax \geq 0.33 \times income$

= TTAX -0.33 INC $\geq$ 0

8) -1 TTAX +0.33 INC $\leq$ 0

M4 *total_tax* $\leq$*income*

= TTAX -INC $\leq$ 0

9) 1 TTAX -1 INC $\leq$ 0

LM1 if *income* $-30\times$*length_of_career* $\geq$ 400 then *tax3* must be *yes*

= -M TAX3 + 30 LEN -INC +EPS $\leq$ 400

10) -M TAX3 +30 LEN -1 INC +1 EPS $\leq$ 400

U1 *Pr1* $\oplus$*Pr2* $\oplus$*Pr3*

= PRO1 +PRO2 $\leq$ 1 and PRO1 +PRO3 $\leq$ 1 and PRO2 +PRO3 $\leq$ 1

11) 1 PRO1 +1 PRO2 $\leq$ 1

12) 1 PRO1 +1 PRO3 $\leq$ 1

13) 1 PRO2 +1 PRO3 $\leq$ 1

U2 *total_tax* = *tax1+tax2+tax3* (with tax1=100, tax2=120, tax3=500)

= TTAX -100 XTAX1 -120 XTAX2 -500 XTAX3 = 0

14) 1 TTAX -100 XTAX1 -120 XTAX2 -500 XTAX3 $\leq$ 0

15) -1 TTAX +100 XTAX1 +120 XTAX2 +500 XTAX3 $\leq$ 0

- XTTAX0-100=1 iff TTAX$\leq$ 100

= M (1-XTTAX0-100) $\geq$ TTAX -100 and -M XTTAX0-100 $\geq$ TTAX -100 -EPS

16) M XTTAX0-100 +1 TTAX $\leq$ M+100

17) M XTTAX0-100 +1 TTAX -1 EPS$\leq$ 100

- XINC0-200=1 iff INC$\leq$ 200

= M (1-XINC0-200) $\geq$ INC -200 and M XINC0-200 $\geq$ 200 +EPS - INC

18) M XINC0-200 +1 INC $\leq$ M +200

19) -M XINC0-200 -1 INC +1 EPS $\leq$ -200

U3 *age* $\geq$ 0 and $\leq$ 110

= AGE $\geq$ 0 and AGE $\leq$ 110

20) -1 AGE $\leq$ 0	21) 1 AGE $\leq$ 110

U4 *length_of_career* $\geq$ 0 and $\leq$ 92

= LEN $\geq$ 0 and LEN $\leq$ 92

22) -1 LEN $\leq$ 0	23) 1 LEN $\leq$ 92

U5 $\varepsilon \geq 0$ and $\leq$ 0.001

= EPS $\geq$ 0 and EPS $\leq$ 0.001

24) -1 EPS $\leq$ 0	25) 1 EPS $\leq$ 0.001

U6 *total_tax* $\geq$ 0 and $\leq$ 2000

= TTAX $\geq$ 0 and $\leq$ 2000

26) -1 TTAX $\leq$ 0	27) 1 TTAX $\leq$ 2000

U7 *income* $\geq$ 0 and $\leq$ 5000

= INC $\geq$ 0 and INC $\leq$ 5000

28) -1 INC $\leq$ 0	29) 1 INC $\leq$ 5000

- XPRO1 binary

30) -1 XPRO1 $\leq$ 0	31) 1 XPRO1 $\leq$ 1

- XPRO2 (binary)

32) -1 XPRO2 $\leq$ 0	33) 1 XPRO2 $\leq$ 1

- XPRO3 (binary)

34) -1 XPRO3 $\leq$ 0	35) 1 XPRO3 $\leq$ 1

- XTAX1 (binary)

36) -1 XTAX1 $\leq$ 0	37) 1 XTAX1 $\leq$ 1

- XTAX2 (binary)

38) -1 XTAX2 $\leq$ 0	39) 1 XTAX2 $\leq$ 1

- XTAX3 (binary)

40) -1 XTAX3 ≤ 0 41) 1 XTAX3 ≤ 1

- XTTAX0-100 (binary)

42) -1 XTTAX0-100 ≤ 0 43) 1 XTTAX0-100 ≤ 1

- XINC0-200 (binary)

44) -1 XINC0-200 ≤ 0 45) 1 XINC0-200 ≤ 1

Overall, we have the following set of linear inequalities in form $\leq$

1) -1 XTAX1 +1 XPRO1 ≤ 0
2) -1 XTAX1 +1 XPRO2 ≤ 0
3) -1 XTAX2 +1 XPRO3 ≤ 0
4) -M TAX3 +1 INC +1 EPS ≤ 1000
5) 1 XINC0-200 -1 XTTAX0-100 ≤ 0
6) 1 INC -20 LEN ≤ 1000
7) -1 INC +30 LEN ≤ -200
8) -1 TTAX +0.33 INC ≤ 0
9) 1 TTAX -1 INC ≤ 0
10) -M TAX3 +30 LEN -1 INC +1 EPS ≤ 400
11) 1 PRO1 +1 PRO2 ≤ 1
12) 1 PRO1 +1 PRO3 ≤ 1
13) 1 PRO2 +1 PRO3 ≤ 1
14) 1 TTAX -100 XTAX1 -120 XTAX2 -500 XTAX3 ≤ 0
15) -1 TTAX +100 XTAX1 +120 XTAX2 +500 XTAX3 ≤ 0
16) M XTTAX0-100 +1 TTAX $\leq$ M+100
17) M XTTAX0-100 +1 TTAX -1 EPS$\leq$ 100

18) M XINC0-200 +1 INC $\leq$ M +200

19) M XINC0-200 +1 INC -1 EPS $\leq$ 200

20) -1 AGE $\leq$ 0 21) 1 AGE $\leq$ 110

22) -1 LEN $\leq$ 0 23) 1 LEN $\leq$ 92

24) -1 EPS $\leq$ 0 25) 1 EPS $\leq$ 0.001

26) -1 TTAX $\leq$ 0 27) 1 TTAX $\leq$ 2000

28) -1 INC $\leq$ 0 29) 1 INC $\leq$ 5000

30) -1 XPRO1 $\leq$ 0 31) 1 XPRO1 $\leq$ 1

32) -1 XPRO2 $\leq$ 0 33) 1 XPRO2 $\leq$ 1

34) -1 XPRO3 $\leq$ 0 35) 1 XPRO3 $\leq$ 1

36) -1 XTAX1 $\leq$ 0 37) 1 XTAX1 $\leq$ 1

38) -1 XTAX2 $\leq$ 0 39) 1 XTAX2 $\leq$ 1

40) -1 XTAX3 $\leq$ 0 41) 1 XTAX3 $\leq$ 1

42) -1 XTTAX0-100 $\leq$ 0 43) 1 XTTAX0-100 $\leq$ 1

44) -1 XINC0-200 $\leq$ 0 45) 1 XINC0-200 $\leq$ 1

By ordering the binary (x) and real variables (z), the overall matrix and the overall vector of system (5.5), corresponding to A and a of system (5.2), unless an easy reordering of some of the box inequalities, are the following:

	XPRO	XPRO	XPRO	XTAX	XTAX	XTAX	XTTAX	XINC	TTAX	INC	AGE	LEN	EPS	vector
	1	2	3	1	2	3	0-100	0-200						*a*
1 (L1)	1	0	0	-1	0	0	0	0	0	0	0	0	0	0
2 (L1)	0	1	0	-1	0	0	0	0	0	0	0	0	0	0
3 (L2)	0	0	1	0	-1	0	0	0	0	0	0	0	0	0
4 (L3)	0	0	0	0	0	-12000	0	0	0	1	0	0	1	1000
5 (L4)	0	0	0	0	0	0	-1	1	0	0	0	0	0	0
6 (M1)	0	0	0	0	0	0	0	0	0	1	0	-20	0	1000
7 (M21)	0	0	0	0	0	0	0	0	0	-1	0	30	0	-200
8 (M3)	0	0	0	0	0	0	0	0	-1	0.33	0	0	0	0
9 (M4)	0	0	0	0	0	0	0	0	1	-1	0	0	0	0
10 (LM1)	0	0	0	0	0	-12000	0	0	0	-1	0	30	1	400
11 (U1)	1	1	0	0	0	0	0	0	0	0	0	0	0	1
12 (U1)	1	0	1	0	0	0	0	0	0	0	0	0	0	1
13 (U1)	0	1	1	0	0	0	0	0	0	0	0	0	0	1
14 (U2)	0	0	0	-100	-120	-500	0	0	1	0	0	0	0	0
15 (U2)	0	0	0	100	120	500	0	0	-1	0	0	0	0	0
16 (U2)	0	0	0	0	0	0	12000	0	1	0	0	0	0	12100
17 (U2)	0	0	0	0	0	0	-12000	0	-1	0	0	0	1	-100
18 (U2)	0	0	0	0	0	0	0	12000	0	1	0	0	0	12200
19 (U2)	0	0	0	0	0	0	0	-12000	0	-1	0	0	1	-200
20 (U3)	0	0	0	0	0	0	0	0	0	0	-1	0	0	0
21 (U3)	0	0	0	0	0	0	0	0	0	0	1	0	0	110
22 (U4)	0	0	0	0	0	0	0	0	0	0	0	-1	0	0
23 (U4)	0	0	0	0	0	0	0	0	0	0	0	1	0	92
24 (U5)	0	0	0	0	0	0	0	0	0	0	0	0	-1	0
25 (U5)	0	0	0	0	0	0	0	0	0	0	0	0	1	0.001
26 (U6)	0	0	0	0	0	0	0	0	-1	0	0	0	0	0
27 (U6)	0	0	0	0	0	0	0	0	1	0	0	0	0	2000
28 (U7)	0	0	0	0	0	0	0	0	0	-1	0	0	0	0
29 (U7)	0	0	0	0	0	0	0	0	0	1	0	0	0	5000
30 (U7)	-1	0	0	0	0	0	0	0	0	0	0	0	0	0
31 (U7)	1	0	0	0	0	0	0	0	0	0	0	0	0	1
32 (U7)	0	-1	0	0	0	0	0	0	0	0	0	0	0	0
33 (U7)	0	1	0	0	0	0	0	0	0	0	0	0	0	1
34 (U7)	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
35 (U7)	0	0	1	0	0	0	0	0	0	0	0	0	0	1
36 (U7)	0	0	0	-1	0	0	0	0	0	0	0	0	0	0
37 (U7)	0	0	0	1	0	0	0	0	0	0	0	0	0	1
38 (U7)	0	0	0	0	-1	0	0	0	0	0	0	0	0	0
39 (U7)	0	0	0	0	1	0	0	0	0	0	0	0	0	1
40 (U7)	0	0	0	0	0	-1	0	0	0	0	0	0	0	0
41 (U7)	0	0	0	0	0	1	0	0	0	0	0	0	0	1
42 (U7)	0	0	0	0	0	0	-1	0	0	0	0	0	0	0
43 (U7)	0	0	0	0	0	0	1	0	0	0	0	0	0	1
44 (U7)	0	0	0	0	0	0	0	-1	0	0	0	0	0	0
45 (U7)	0	0	0	0	0	0	0	1	0	0	0	0	0	1

Now we solve the *dual* model (5.7) with objective cost vector $c = \mathbf{1}$ and using the above matrix and vector. Model (5.7) is in this case infeasible so, according to Theorem 5.4, the primal (5.1) has no complete inconsistencies. We now search for each partial inconsistency by imposing the value activating it. In practice we try to impose any possible value for each field, and every time we find a vertex for model (5.7) we have detected a partial inconsistency.

If we add the constraint that LEN ≥ 67, that corresponds to adding the following row to the above matrix,

	XPRO 1	XPRO 2	XPRO 3	XTAX 1	XTAX 2	XTAX 3	XTTAX 0-100	XINC 0-200	TTAX	INC	AGE	LEN	EPS	vector a
46	0	0	0	0	0	0	0	0	0	0	0	-1	0	-67

we obtain that (7) has a vertex solution

$$y = \{0,0,0,0,0,0,0.035,0.11,0,0,0,0,0,0.11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,10.75,0,12.9,0,53.76,0,0,0,0,1.06\}$$

where the support is given by the 7th, the 8th, the 14th, the 37th, the 38th, the 39th and the 46th. This means that the corresponding inequalities are forming an IIS. The partial contradiction is between the 6 inequalities corresponding to the 4 following rules, and it appears for LEN$\geq$ 67 (46^th inequality), as showed in the beginning of this Section.

M2 *income* $\geq 200 + 30\times$ *length_of_career*

M3 *total_tax* $\geq 0.33\times$ *income*

U2 *total_tax* = *tax1+tax2+tax3*

U7 *income* ≥ 0 and ≤ 5000

If we add the constraint that LEN ≥ 81, that corresponds to adding the following row to the above matrix,

	XPRO 1	XPRO 2	XPRO 3	XTAX 1	XTAX 2	XTAX 3	XTTAX 0-100	XINC 0-200	TTAX	INC	AGE	LEN	EPS	vector a
46	0	0	0	0	0	0	0	0	0	0	0	-1	0	-81

we obtain that (7) has a vertex solution

$$y = \{0,0,0,0,0,0.1,0.1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1\}$$

where the support is given by the 6th, the 7th and the 46th. This means that the corresponding inequalities are forming an IIS. The partial contradiction is between the 2 inequalities corresponding to the 2 following rules, and it appears for LEN$\geq$ 81 (46^th inequality), as showed in the beginning of this Section.

M1 *income* $\leq 1000 + 20 \times$ *length_of_career*

M2 *income* $\geq 200 + 30 \times$ *length_of_career*

If we add the constraint that INC $\geq$ 2182, that corresponds to adding the following row to the above matrix,

	XPRO 1	XPRO 2	XPRO 3	XTAX 1	XTAX 2	XTAX 3	XTTAX 0-100	XINC 0-200	TTAX	INC	AGE	LEN	EPS	vector a
46	0	0	0	0	0	0	0	0	0	-1	0	0	0	-2182

we obtain that (7) has a vertex solution

$$y = \{0,0,0,0,0,0,0,16.67,0,0,0,0,0,16.67,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,\\ 0,0,0,0,0,0,0,1667,0,2000,0,8333,0,0,0,0,5.5\}$$

where the support is given by the 8^{th}, the 14^{th}, the 37^{th}, the 39^{th}, the 41^{th} and the 46^{th}. This means that the corresponding inequalities are forming an IIS. The contradiction partial is between the 5 inequalities corresponding to the following 3 rules, and it appears for INC$\geq$ 2182 (46^th inequality), as showed in the beginning of this Section.

M3 *total_tax* $\geq 0.33 \times$ *income*

U2 *total_tax = tax1+tax2+tax3*

U7 *income* ≥ 0 and ≤ 5000

Therefore, the proposed procedure was able to discover the sets of conflicting rules working only at the formal level.

5.5 Summary and Remarks

The Localization of Contradictions, also known as Inconsistency Selection, is a difficult Artificial Intelligence problem that arises in several different applications. Its solution can in general be extremely difficult, not only for computational reasons. A contradiction can be quite hidden, or involve many rules, or there can be several contradiction. However, when a set of statement or rules can be converted into linear inequalities, this task can be performed with a procedure based on Alternative theorems, in particular Farkas' lemma, and can therefore be solved quite efficiently when the integral-point property holds. A main feature of the proposed approach is that the procedure works only at the formal level, without the need of domain-specific knowledge.

References

[1] G. Alexe, S. Alexe, P.L. Hammer and A. Kogan. Comprehensive vs. Comprehensible Classifiers in Logical Analysis of Data. Discrete Applied Mathematics, vol. 156, no. 6, 870-882, 2008.

[2] H. Almuallim and T.G. Dietterich. Learning Boolean Concepts in the Presence of many Irrelevant Features. Artificial Intelligence, vol. 69, no. 1, 279-306, 1994.

[3] E. Amaldi, M.E. Pfetsch and L. Trotter Jr.. Some structural and algorithmic properties of the maximum feasible subsystem problem. Proceedings. of 10th Integer Programming and Combinatorial Optimization conference. Lecture Notes in Computer Science 1610, Springer, 45–59, 1999.

[4] M. Ayel and J.P. Laurent (eds.). Validation, Verification and Testing of Knowledge-Based Systems. John Wiley & Sons Ltd., Chichester, England, 1991.

[5] M. Bacharach. Matrix rounding problems. Management Science 12(9), 732-742, 1966.

[6] M. Bacharach. Biproportional Matrices and Input-Output Change. Cambridge University Press, Cambridge, UK, 2004.

[7] Banff Support Team. Functional Description of the Banff System for Edit and Imputation System. Quality Assurance and Generalized Systems Section Tech. Rep. Statistics Canada, 2003.

[8] M. Bankier. Canadian Census Minimum change Donor imputation methodology. In Proceedings of the Workshop on Data Editing, UN/ECE, Cardiff, United Kingdom, 2000.

[9] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali. Linear Programming and Network Flows. John Wiley & Sons, second edition, New York, 1990.

[10] D. Bertsimas and J.N. Tsitsiklis. Introduction to Linear Optimization. Athena Scientific, Belmont, Massachusetts, 1997.

[11] G. Bianchi, R. Bruni and A. Reale. Open Source Integer Linear Programming Solvers for Error Localization in Numerical Data. In N. Torelli, et al. (eds.), Advances in Theoretical and Applied Statistics, Studies in Theoretical and Applied Statistics, Springer-Verlag Berlin Heidelberg, DOI 10.1007/978-3-642-35588-2_28, 2013.

[12] G. Bianchi and R. Bruni. A Formal Procedure for Finding Contradictions into a Set of Rules. Applied Mathematical Sciences 6(126), 6253-6271, 2012.

[13] G. Bianchi, R. Bruni and A Reale. Balancing of Agricultural Census Data by Using Discrete Optimization. Optimization Letters, DOI 10.1007/s11590-013-0652-3, 2013.

[14] G. Bianchi, R. Bruni and A. Reale. Information Reconstruction via Discrete Optimization for Agricultural Census Data. Applied Mathematical Sciences Vol. 6(125), 6241-6251, 2012.

[15] I.M. Bomze and M. Locatelli. Separable standard quadratic optimization problems. Optimization Letters 6, 857-866, 2012.

[16] N. Bourbaki. Topological vector spaces. Springer-Verlag, Berlin, Germany, 1987.

[17] S. Boyd and L. Vandenberghe. Convex Optimization. Cambridge University Press, Cambridge, UK, 2004.

[18] E. Boros, P.L. Hammer, T. Ibaraki and A. Kogan. Logical Analysis of Numerical Data. Mathematical Programming, vol. 79, 163-190, 1997.

[19] E. Boros, P.L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz and I. Muchnik. An Implementation of Logical Analysis of Data. IEEE Transactions on Knowledge and Data Engineering, vol. 12, no. 2, 292-306, 2000.

[20] E. Boros, T. Horiyama, T. Ibaraki, K. Makino, and M. Yagiura. Finding Essential Attributes from Binary Data. Annals of Mathematics and Artificial Intelligence, vol. 39 no. 3, 223-257, 2003.

[21] E. Boros, Y. Crama, P.L. Hammer, T. Ibaraki, A. Kogan, and K. Makino. Logical Analysis of Data: Classification with Justification. Annals of Operations Research, vol. 188, 33-61, 2011.

[22] L. Breiman. Bagging predictors. Machine Learning, vol. 24, no. 2, 123-140, 1996.

[23] L. Breiman. Random Forests. Machine Learning, vol. 45, no. 1, 532, 2001.

[24] R. Bruni and A. Sassano. Error Detection and Correction in Large Scale Data Collecting. In Advances in Intelligent Data Analysis, Lecture Notes in Computer Science 2189, Springer, 84-94, 2001.

[25] R. Bruni. Approximating Minimal Unsatisfiable Subformulae by means of Adaptive Core Search. Discrete Applied Mathematics. Vol. 130(2), 85-100, 2003.

[26] R. Bruni. Discrete Models for Data Imputation. Discrete Applied Mathematics Vol. 144/1, 59-69, 2004.

[27] R. Bruni. Error Correction for Massive Data Sets. Optimization Methods and Software, Vol. 20/2-3, 295-314, 2005.

[28] R. Bruni. On Exact Selection of Minimally Unsatisfiable Subformulae. Annals of Mathematics and Artificial Intelligence Vol. 43(1-4), 35-50, 2005.

[29] R. Bruni. Reformulation of the Support Set Selection Problem in the Logical Analysis of Data. Annals of Operations Research Vol. 150(1), 79-92, 2007.

[30] R. Chandrasekaran. Integer programming problems for which a simple rounding type of algorithm works, In W.R. Pulleyblank, ed. Progress in Combinatorial Optimization, Academic Press, 101-106, 1984.

[31] V. Chandru and J.N. Hooker. Extended Horn sets in propositional logic. J. ACM 38, 205-221, 1991.

[32] C.C. Chang and C.J. Lin. Training ν-support vector classifiers: Theory and algorithms. Neural Computation, vol. 13, no. 9, 2119-2147, 2001.

[33] C.C. Chang and C.J. Lin. LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, vol. 2, no. 3:27, 2011.

[34] J.W. Chinneck. Fast Heuristics for the Maximum Feasible Subsystem Problem. INFORMS Journal on Computing 13/3, 210-223, 2001

[35] J.W. Chinneck and E.W. Dravnieks, Locating Minimal Infeasible Constraint Sets in Linear Programs. ORSA Journal on Computing 3, 157-168, 1991.

[36] M. Conforti, G. Cornuéjols, A. Kapoor and K. Vuskovic. Recognizing balanced 0, + or - 1 matrices. In Proceedings 5th annual SIAM/ACM Symposium on Discrete Algorithms, 103-111, 1994.

[37] C. Cortes and V. Vapnik. Support-Vector Networks. Machine Learning, vol. 20, no. 3, 273-297, 1995.

[38] T.M. Cover and P.E. Hart (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory 13 (1), 2127, 1967.

[39] Y. Crama and P.L. Hammer. Boolean Functions: Theory, Algorithms, and Applications. Cambridge University Press, New York, 2011. ISBN: 9780521847513.

[40] Y. Crama, P.L. Hammer, and T. Ibaraki. Cause-effect Relationships and Partially Defined Boolean Functions. Annals of Operations Research vol. 16, 299-326, 1988.

[41] T. De Waal. Computational Results with Various Error Localization Algorithms. UNECE Statistical Data Editing Work Session, Madrid, Spain, 2003.

[42] T. De Waal, J. Pannekoek and S. Scholtus. Handbook of Statistical Data Editing and Imputation. Wiley Handbooks in Survey Methodology, John Wiley & Sons, Inc., New York, NY, 2011.

[43] T. De Waal. Processing of Erroneous and Unsafe Data. Ph.D. Thesis, ERIMPhD series in Research Management, 2003.

[44] European Council Regulation (EEC) No 357/79 of 5 February 1979 on statistical surveys. EEC Documentation, 1979.

[45] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (eds.) Advances in Knowledge Discovery and Data Mining. AAAI Press / The MIT Press, Menlo Park, CA, 1996.

[46] U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. Artificial Intelligence, vol.13, 1022-1027, 1993.

[47] I.P. Fellegi and I.P.D. Holt. A Systematic Approach to Automatic Edit and Imputation, Journal of the American Statistical Association 71, 17-35, 1976

[48] C. Forbes, M. Evans, N. Hastings, and B. Peacock, Statistical Distributions (fourth edition). Wiley series in Probability and Statistics, New York, 2010.

[49] A. Frank, A. Asuncion. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2010.

[50] Y. Freund, Boosting a Weak Learning Algorithm by Majority. Information and Computation, vol. 121, no. 2, 256-285, 1995.

[51] W.A. Fuller. Measurement Error Models. Wiley Series in Probability and Statistics, John Wiley & Sons, Inc., New York, NY, 2006.

[52] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co, San Francisco, CA, 1979.

[53] R.S. Garfinkel, A.S. Kunnathur and G.E. Liepins. Optimal Imputation of Erroneous Data: Categorical Data, General Edits. Operations Research 34, 744-751, 1986.

[54] R.S. Garfinkel, A.S. Kunnathur and G.E. Liepins. Error Localization for Erroneous Data: Continuous Data, Linear Constraints. SIAM Journal on Scientific and Statistical Computing 9, 922-931, 1988.

[55] J. Gleeson and J. Ryan. Identifying Minimally Infeasible Subsystems of Inequalities. ORSA Journal on Computing 2/1, 61-63, 1990.

[56] O. Guieu and J.W. Chinneck. Analyzing Infeasible Mixed-Integer and Integer Linear Programs. INFORMS Journal on Computing 11/1, 63-77, 1999.

[57] I. Guyon, N. Matic and V. Vapnik. Discovering Informative Patterns and Data Cleaning.

[58] P.L. Hammer, A. Kogan, B. Simeone and S. Szedmak. Pareto-Optimal Patterns in Logical Analysis of Data. Discrete Applied Mathematics, vol. 144, no. 1-2, 79-102, 2004.

[59] J. Han and M. Kamber. Data Mining: Concepts and Techniques. Elsevier, second edition, 2006.

[60] D.J. Hand, H. Mannila and P. Smyth. Principles of Data Mining. MIT Press, London, 2001.

[61] T. Hastie, R. Tibshirani and J. Friedman. The Elements of Statistical Learning: Data Mining, Inference and Prediction. Springer, New York, NY, 2001.

[62] T. Hastie, R. Tibshirani and J. Friedman. The Elements of Statistical Learning. Springer-Verlag, New York, Berlin, Heidelberg, 2002.

[63] F.S. Hillier and G.J. Lieberman. Introduction to Operations Research. McGraw-Hill, New York, NY, eighth edition, 2005.

[64] D.S. Hochbaum and J.G. Shanthikuma. Convex separable optimization is not much harder than linear optimization. Journal of the ACM 37(4), 843-862, 1990.

[65] IBM: *Ilog Cplex 12.1 Reference Manual.* International Business Machines Corporation, 2009.

[66] IBM: *Ilog Concert Technology 12.1 Reference Manual.* International Business Machines Corporation, 2009.

[67] N. Jankowski, W. Duch, K Grabczewski (eds.) Meta-Learning in Computational Intelligence. Springer-Verlag, Berlin, 2011.

[68] M. Kantardzic. Data Mining: Concepts, Models, Methods, and Algorithms. John Wiley & Sons Publishing, 2003.

[69] B. Kalantari, I. Lari, F. Ricca and B. Simeone. On the complexity of general matrix scaling and entropy minimization via the RAS algorithm. Mathematical Programming, Ser. A 112, 371401, 2008.

[70] W. Klø̈esgen and J.M. Zytkow (eds.). Handbook of Data Mining and Knowledge Discovery. Oxford University Press: Oxford, UK, 2002.

[71] L.E. Lyberg , P. Biemer, M. Collins, E.D. De Leeuw, C. Dippo, N. Schwarz and D. Trewin (eds.). Survey Measurement and Process Quality, Section C, post survey processing and operations. John Wiley & Sons, Inc.: New York, NY, 1997.

[72] T. Menzies. Knowledge Maintenance: The State of the Art. Knowledge Engineering Review, 14(1), 1-46, 1999.

[73] T.M. Mitchell, Machine Learning. McGraw-Hill, Singapore, 1997.

[74] A. Mucherino, P. Papajorgji and P.M. Pardalos. Data Mining in Agriculture. Springer: New York, NY, 2009.

[75] J.H. Myers and E.W. Forgy. The Development of numerical credit evaluation systems. Journal of the American Statistical Association, Vol.58 Issue 303 (Sept) 799-806, 1963.

[76] G.L. Nemhauser and L.A. Wolsey. Integer and Combinatorial Optimization, J. Wiley & Sons, Inc., New York, NY, 1999.

[77] F.C. Pampel FC. Logistic regression: A primer. Sage University Papers Series on Quantitative Applications in the Social Sciences, 07-132. Thousand Oaks, CA: Sage Publications, 2000.

[78] C. Poirier. A Functional Evaluation of Edit and Imputation Tools. UN/ECE Work Session on Statistical Data Editing, Working Paper n.12, Rome, Italy, 1999.

[79] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. Numerical Recipes in C: The Art of Scientific Computing, second edition. Cambridge University Press, 1992.

[80] J.R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA, 1993.

[81] C.T. Ragsdale P.G. McKeown. On Solving the Continuous Data Editing Problem. Computers and Operations Research 23, 263-273, 1996.

[82] R. Ramakrishnan, J. Gehrke. Database Management Systems (third edition). McGraw-Hill: New York, NY, 2003.

[83] N. Rescher and R. Brandom. The Logic of Inconsistency. Basil Blackwell, Oxford, 1980.

[84] J. Riera-Ledesma and J.J. Salazar-Gonzalez. New Algorithms for the Editing and Imputation Problem. UNECE Statistical Data Editing Work Session, Madrid, Spain, 2003.

[85] F. Rosenblatt. The Perceptron: A Probalistic Model For Information Storage And Organization In The Brain. Psychological Review 65 (6): 386-408, 1958.

[86] R.E. Schapire. The Strength of Weak Learnability. Machine Learning, vol. 5, no. 2, 197-227, 1990.

[87] M.H. Schneider and S.A. Zenios. A comparative study of algorithms for matrix balancing. Operations Research, 38(3), 439-455, 1990.

[88] A. Schrijver. Theory of Linear and Integer Programming. Wiley, New York, 1986.

[89] A. Schrijver. Combinatorial Optimization. Springer, Berlin, New York, 2003.

[90] M. Tamiz, S.J. Mardle and D.F. Jones. Detecting IIS in Infeasible Linear Programs using Techniques from Goal Programming, Computers and Operations Research 23, 113-191, 1996.

[91] V. Vapnik. Statistical Learning Theory. John Wiley and Sons, New York, 1998.

[92] G. Wei. Comparison Study of Genetic Algorithm and Evolutionary Programming. Proc. of the Third International Conference on Machine Learning and Cybernetics, 1(1): 204-209, 2004.

[93] H.P. Williams. Model Building in Mathematical Programming. J.Wiley, Chichester, 1993.

[94] W.E. Winkler. State of Statistical Data Editing and current Research Problems. In Proceedings of the Workshop on Data Editing, UN/ECE, Rome, Italy, 1999.

[95] W.L. Winston and M. Venkataramanan. Introduction to Mathematical Programming: Applications and Algorithms. Volume 1. Duxbury Press (fourth edition), Belmont, CA, 2002.

[96] D.H. Wolpert. The Lack of A Priori Distinctions Between Learning Algorithms. Neural Computation vol. 8, 1341-1390, 1996.

[97] Y. Yang and X. Liu. A re-examination of text categorization methods. Annual ACM Conference on Research and Development in Information Retrieval, 42-49, USA, 1999

[98] L.A.Zadeh. Fuzzy sets. Information and Control (8), 338-353, 1965.

[99] L.A.Zadeh. Calculus of fuzzy restrictions. In L. A. Zadeh, K.-S. Fu, K. Tanaka and M. Shimura (eds.), Fuzzy sets and Their Applications to Cognitive and Decision Processes. Academic Press, New York, 1975.

[100] G.P. Zhang. Neural Networks for Classification: a Survey. IEEE Transactions on Systems, Man, and Cybernetics. vol. 30, no. 4, 451-462, 2000.

[101] H. Zhang. The Optimality of Naive Bayes. FLAIRS2004 conference.

[102] H.-J. Zimmermann. Practical Applications of Fuzzy Technologies. Springer, 2000.