Integer Optimization in Data Mining

by

Romy Shioda

S.B. Management, Massachusetts Institute of Technology, 1999
S.B. Mathematics with Computer Science, Massachusetts Institute of Technology, 1999
S.M. Operations Research, Massachusetts Institute of Technology, 2002

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2003

Author .....
Sloan School of Management
May 19, 2003

Certified by .....
Dimitris Bertsimas
Boeing Professor of Operations Research
Thesis Supervisor

Accepted by .....
James Orlin
Co-director and Edward Pennell Brooks Professor of Operations Research

Integer Optimization in Data Mining

by

Romy Shioda

Submitted to the Sloan School of Management
on May 19, 2003, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

While continuous optimization methods have been widely used in statistics and data mining over the last thirty years, integer optimization has had very limited impact in statistical computation. Thus, our objective is to develop a methodology utilizing state of the art integer optimization methods to exploit the discrete character of data mining problems. The thesis consists of two parts: The first part illustrates a mixed-integer optimization method for classification and regression that we call Classification and Regression via Integer Optimization (CRIO). CRIO separates data points in different polyhedral regions. In classification each region is assigned a class, while in regression each region has its own distinct regression coefficients. Computational experimentation with real data sets shows that CRIO is comparable to and often outperforms the current leading methods in classification and regression. The second part describes our cardinality-constrained quadratic mixed-integer optimization algorithm, used to solve subset selection in regression and portfolio selection in asset allocation. We take advantage of the special structures of these problems by implementing a combination of implicit branch-and-bound, Lemke's pivoting method, variable deletion and problem reformulation. Testing against popular heuristic methods and CPLEX 8.0's quadratic mixed-integer solver, we see that our tailored approach to these quadratic variable selection problems have significant advantages over simple heuristics and generalized solvers.

Thesis Supervisor: Dimitris Bertsimas
Title: Boeing Professor of Operations Research

Acknowledgments

To all of you who have been so wonderful and supportive all of these years at MIT – thank you. I can't imagine the past eight years without you. In particular (in some what arbitrary order): my advisor, Dimitris Bertsimas, for all his support and encouragement over the years; my thesis committee members, Rob Freund and Jim Orlin, for their invaluable input; Georgia Perakis, for inspiration and all of her great advice; Nitin Patel, for his insights and warm encouragements; the OR Center students (past and present), for their generosity and support – both in terms of research and life in general; the MIT music department and all the incredible musicians I have had the pleasure of performing with – you have enriched my life and would be terribly missed; Finally, my close friends and family, for their patience and encouragement, and being there in both the worst and best of times. Thank you!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last twenty years, the availability of massive amounts of data in electronic form and the spectacular advances in computational power have led to the development of the field of data mining (or knowledge discovery). A simple definition for data mining is the process of extracting meaningful information – i.e., patterns and relations – from the data set, commonly using statistical and computer science techniques, in order to build a predictive model that can be used to accurately predict behaviors in the future.

Data mining techniques have been utilized and have been successful in a vast range of application areas, such as marketing, finance, health care, and artificial intelligence. For example, a catalog company many want to ship their promotions only to customers with high probability of response. Health care practitioners may want to develop a model to detect the potential of cancer in patients, given their physiological and genetic information. Recently, significant work has been done in text classification to detect and remove spam in emails. In all of these areas, practitioners want to utilize the enormous amounts of historical data, which may potentially be rich with hidden information, to aid them in making more focused, accurate decisions in the future.

Key differences between data mining and traditional statistical inference techniques are the scale of the data sets, computational requirements and the drivers of the prediction model. In traditional statistical inference tools, statisticians tested

their hypothesis on the data under several assumptions (e.g., distributional assumptions). Often, only a few observations are available in the data set, and thus one cannot adequately capture the underlying characteristics of the data without any assumptions. In contrast, there is an abundance of data as well as computational power in data mining problems. This marriage between the size of the data set and computation power is the key ingredient and motivation to data mining. Distributional assumptions are no longer necessary to infer relationships in the data, and there is no need to restrict the model into traditional statistical frameworks.

In addition, the main criterion of data mining practitioners is prediction accuracy. Due to the abundance of data, a small random fraction of the data set can be set aside for model validation. After a prediction model is constructed, what is important is its prediction accuracy on the validation data set. Regardless of the hypothesis and assumptions, the prevailing model is the model that has superior prediction in future data points. Thus, there is less emphasis on statistical and geometric theory, and more importance on the accuracy and robustness of the prediction.

Clearly, with this shift in focus from statistical frameworks to optimal predictions, most data mining problems lend themselves to be modelled as optimization problems. For example, in classification problems, we minimize the total misclassification error. In regression, we minimize the total squared or absolute error. The problems often exhibit discrete characteristics as well. For example, in binary classification, data are often separated into disjoint regions, which can be modelled as a complex assignment problem. In variable/feature selection, a proper subset of the variables are chosen that gives the best predictive fit to the data. This property makes these problems good candidates for modelling as discrete optimization problems.

Continuous optimization has had quite a presence in data mining, as well as traditional statistics. Some notable applications are support vector machines in machine learning, linear least square regression, logistic regression and maximum likelihood estimators. However, in comparison, integer optimization has had very limited impact. The statistics community has long recognized that many data mining problems can be formulated as integer optimization problems [1], however, the belief was formed

in the early 1970's that these methods are not tractable in practical computational settings. As a result, the applicability of integer optimization methods to statistical problems has not been seriously investigated.

The objective in this thesis is to develop a methodology utilizing state of the art integer optimization methods to exploit the discrete characteristics of data mining problems. Chapters 2 and 3 describe new techniques for solving classification and regression problems, which we call Classification and Regression via Integer Optimization (CRIO). CRIO solves these problems by merging current methodologies with integer optimization. CRIO is able to capture complex, discrete behaviors of the data, and thus improve on prediction accuracy due to the modelling power of integer optimization. It further alleviates the misconception of the intractability of integer optimization, by solving these problems in reasonable and comparable times to the existing popular methods.

Chapter 4 introduces new solution algorithms for solving variable selection problems – a crucial issue in data mining. One of the difficulties of having such large amounts of data, is that often we are given variables or features that are not essential in the development of an accurate prediction model. Not only is using all the available variables computationally costly, it also increases the variance of the predicted value (i.e., reduces robustness in its prediction). The more variables the model depends on, the more noisy its estimation. Thus, clearly we want to be able to select a subset of the variables that are critical for the prediction model. However, this makes it a very difficult combinatorial problem. Currently, data mining practitioners use simple greedy heuristics to make these variable selections, but often such a myopic selection strategy gives us a subset of variables that are far from optimal. In the last chapter, we develop an algorithm tailored towards variable selection problems with quadratic costs – a criteria function that occurs frequently in the context of data mining (such as total squared error and expected variance). From computational experimentations, our methods are successful in finding good feasible solutions within practical time. In certain cases, they also solve the problems faster than generic commercial quadratic mixed-integer optimization software.

The structure of the thesis is as follows:

Chapter 2: Classification via Integer Optimization We present the binary classification problem and the current state of the art methodologies for solving it. We then elaborate on our approach that combines the use of mixed-integer optimization, clustering and continuous optimization to effectively solve classification problems. Finally, we compare the performance of CRIO against the current methods and illustrate some computational results.

Chapter 3: Regression via Integer Optimization First we introduce the problem of multivariate regression, along with the current leading methods. We then discuss the regression component of CRIO that solves regression problems using mixed-integer optimization, clustering and nonlinear transformations. Finally, we compare the predictive accuracy of CRIO against the current methods and illustrate some computational results.

Chapter 4: Quadratic Variable Selection via Integer Optimization We present the quadratic variable selection problem and discuss notable works in this area. We then elaborate on our method that formulates this problem as a quadratic mixed-integer optimization problem with cardinality constraints, and our algorithm that efficiently solves these problems using a combination of implicit branch-and-bound, Lemke's method, variable deletion and re-formulation. Finally, we introduce two applications, variable selection in regression and portfolio selection in asset management, and illustrate computational results.

We view our contributions as follows:

1. To show that CRIO is a promising method for classification and regression that matches and often outperforms other state of the art methods on widely circulated real data sets.

2. To illustrate the computational advantage of using implicit branch-and-bound and Lemke's method to solve quadratic variable selection problems. In subset

selection in regression, our methodology significantly improves the goodness of fit criterion compared to popular heuristic methods, while remaining solvable in practical time. It also had faster node to node computation time compared to an explicit branch-and-bound formulation solved by CPLEX 8.0. Also, the combination of implicit branch-and-bound and Lemke's pivoting method significantly reduces total computation time compared to using barrier method to solve continuous relaxation of the subproblems.

3. Most importantly, we bring to the attention of the statistics and data mining community the view that integer optimization can have a significant impact on statistical computation, and thus motivate researchers to revisit old statistical problems using integer optimization. Similarly, we hope that the Operations Research community would view Data Mining as a rich area for applying various Operations Research tools and concepts.

Chapter 2

Classification via Integer Optimization

In classification problems, we are given n data points $(\boldsymbol{x}_i, y_i)$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $y_i \in \{0, 1\}$ and $i = 1, \ldots, n$, where $\boldsymbol{x}_i$ are called *explanatory variables* and y_i are called *class variables.* Though we focus only on binary classification, our approach can easily be extended to multiple class classification problems. We want to partition the explanatory variable space to separate Class 1 and Class 0 points, so that given new data points with unknown classes, we can predict their class with great accuracy. The challenge is to construct a model that is sophisticated enough for good predictions, yet simple and robust enough to be generalizable for future data points.

The current leading methods for classification can be grouped into two groups: decision tree and separating hyperplane methods. CART [9] and C5.0 [36] are examples of the former. CART (Classification And Regression Trees) recursively splits the data along a variable into hyper-rectangular regions in a locally optimal manner . C5.0 is another popular method which is similar in spirit to CART. CART has been one of the most popular classification tools, especially in the business setting, due to its simplicity and fast solution time. Its main shortcoming is its fundamentally greedy approach and its limitations of partitioning the space only into rectangular regions.

Support vector machines (SVM) [20], [28], [43], are an example of the separating

hyperplane methods. In its simplest form, SVM separates points of different classes by a single hyperplane or a simple nonlinear function. More sophisticated SVM methods map the data points via a nonlinear transformation to a higher dimensional space and use continuous optimization methods to separate the points of different classes. Although this nonlinear kernel method has proven effective, it is difficult to intuitively visualize the separator in the original explanatory variable space.

The classification component of CRIO overcomes the shortcomings of CART and SVM by partitioning points into disjoint polyhedral regions in a globally optimal manner. The following gives a geometric overview of our methodology.

2.1 The Geometry of the Classification Approach

Given n data points $(\boldsymbol{x}_i, y_i)$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $y_i \in \{0, 1\}$ and $i = 1, \ldots, n$, we want to partition $\mathbb{R}^d$ into a small number of regions that only contain points of the same class. Consider for example the points in Figure 2-1. Figure 2-2 illustrates the output of CRIO.

In the first step, we use a mixed-integer optimization model to assign Class 1 points into K groups[1] (K is a user defined parameter), such that no Class 0 point belongs in the convex hull of a Class 1 group[2]. Clearly, we want to keep K small (typically less than five) to avoid over-fitting. Given the presence of outliers, it might be infeasible to use K groups (see Figure 2-3). For this purpose, we further enhance the mixed-integer optimization model by enabling it to eliminate a pre-specified number of outliers. Having defined K groups at the end of this phase, we use quadratic optimization methods to represent groups by polyhedral regions – an approach inspired by SVMs. Finally, we eliminate redundant faces of these polyhedra by iteratively solving linear optimization problems. After the final sets of polyhedra are defined, we classify a new point $\boldsymbol{x}_0$ as Class 1, if it is contained in any of the K polyhedra, and as Class 0, otherwise. In order to reduce the dimension of the mixed-integer optimization model

[1]We use the word *group* to mean a collection of points, while we use the word *region* to mean a polyhedron that contains a group.

[2]We group Class 1 points instead of Class 0 points, without loss of generality, throughout the paper.

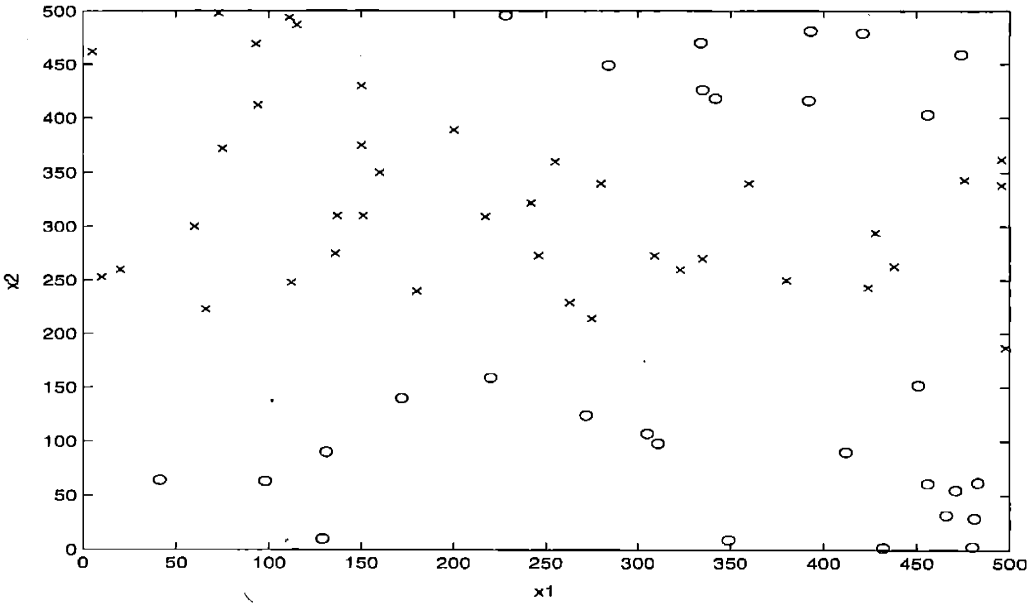

Figure 2-1: A given set of training data. Class 0 points are represented with an "o" and Class 1 points are represented by an "x".

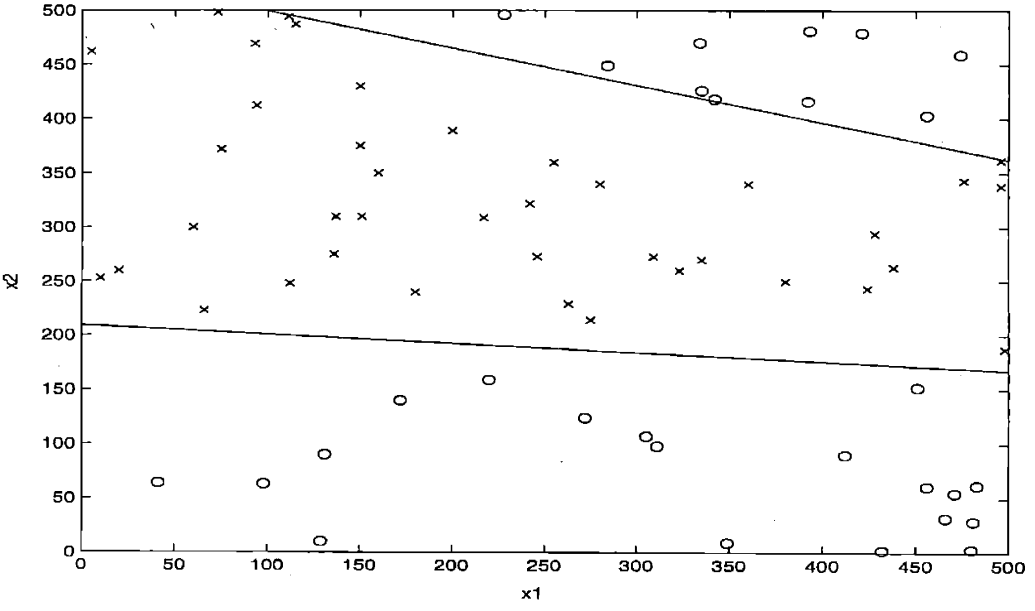

Figure 2-2: The output of CRIO.

and thus reduce computation time, we preprocess the data so that points form small clusters using a clustering algorithm (see Figure 2-4).

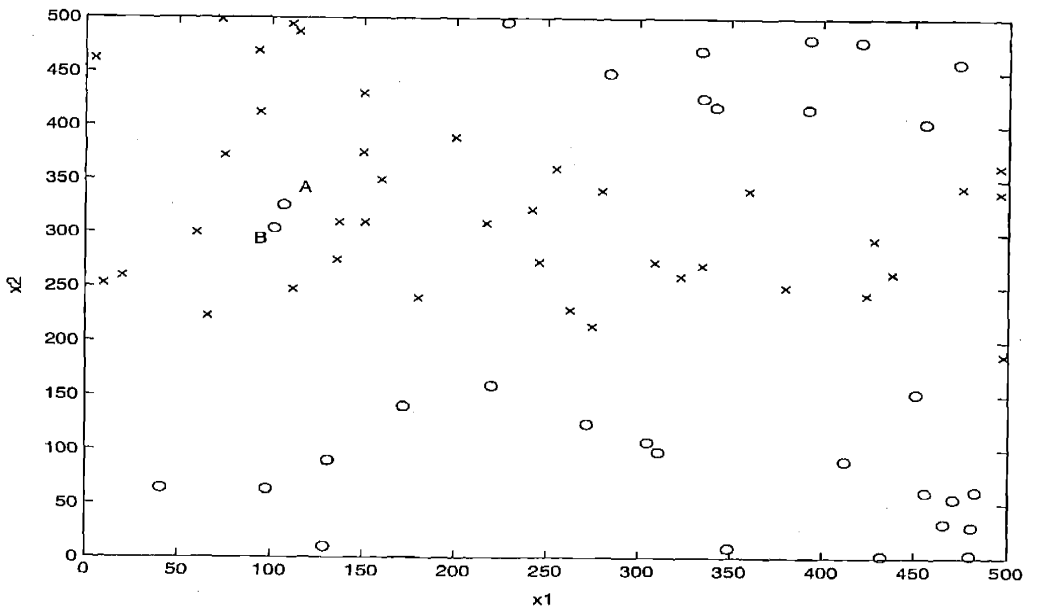

Figure 2-3: Illustration of outliers in the classification data. Points A and B are Class 0 outliers.

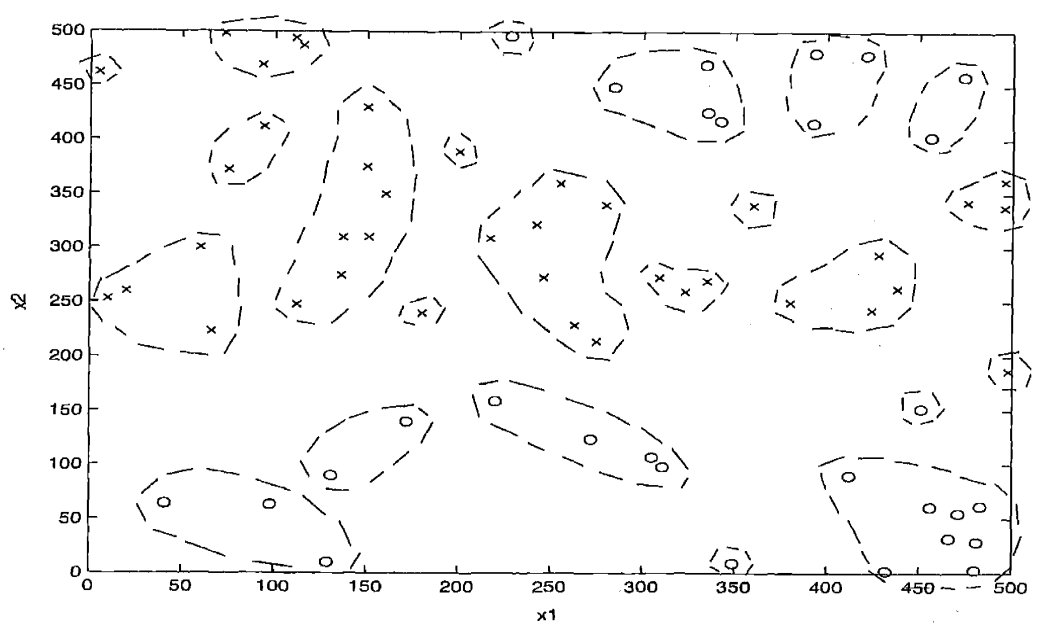

Figure 2-4: Data points are grouped into small clusters in x-space.

In the following sections, we present our methods first for binary and then multiple class classification problems. As outlined in Section 2.1, we first assign both Class 1

and Class 0 points into clusters (Section 2.3), then assign Class 1 clusters to groups via the use of a mixed-integer optimization model that also detects outliers (Section 2.4). Section 2.5 presents the methodology of assigning polyhedra to groups, and Section 2.6 summarizes the overall algorithm for binary classification. Section 2.7 extends these methods to multiple-class classification problems. Finally, Section 2.8 presents some theoretical stability results for our method. We start by presenting the basic mixed-integer optimization model in Section 2.2, which forms the basis of our final approach.

2.2 The Mixed-Integer Optimization Model

The training data consists of n observations $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, n$, with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$. Let m_0 and m_1 be the number of Class 0 and Class 1 points, respectively. We denote Class 0 and Class 1 points by $\boldsymbol{x}_i^0$, $i = 1, \ldots, m_0$, and $\boldsymbol{x}_i^1$, $i = 1, \ldots, m_1$, respectively. Let $M_0 = \{1, \ldots, m_0\}$, $M_1 = \{1, \ldots, m_1\}$ and $\overline{K} = \{1, \ldots, K\}$.

We want to partition Class 1 points into K disjoint groups, so that no Class 0 point can be expressed as a convex combination of these points. Let G_k be the set indices of Class 1 points that are in Group k, where $\bigcup_{k \in \overline{K}} G_k = M_1$ and $\bigcap_{k \in \overline{K}} G_k = \emptyset$. Thus, we require that the following system is infeasible, for all $i \in M_0$ and $k \in \overline{K}$:

$$\begin{aligned} \sum_{j \in G_k} \lambda_j \boldsymbol{x}_j^1 &= \boldsymbol{x}_i^0, \\ \sum_{j \in G_k} \lambda_j &= 1, \\ \lambda_j &\geq 0, \quad j \in G_k. \end{aligned} \tag{2.1}$$

From Farkas' lemma, System (2.1) is infeasible if and only if the following problem is feasible:

$$\begin{aligned} &\boldsymbol{p}'\boldsymbol{x}_i^0 + q < 0, \\ &\boldsymbol{p}'\boldsymbol{x}_j^1 + q \geq 0, \quad j \in G_k. \end{aligned} \tag{2.2}$$

We consider the optimization problem:

$$
\begin{aligned}
z_{k,i} = \text{maximize} \quad & \epsilon \\
\text{subject to} \quad & \boldsymbol{p}'\boldsymbol{x}_i^0 + q \leq -\epsilon, \\
& \boldsymbol{p}'\boldsymbol{x}_j^1 + q \geq 0, \quad j \in G_k, \\
& 0 \leq \epsilon \leq 1.
\end{aligned} \tag{2.3}
$$

If $z_{k,i} > 0$, System (2.2) is feasible and thus Problem (2.1) is infeasible. If $z_{k,i} = 0$, System (2.2) is infeasible and thus Problem (2.1) is feasible, i.e., $\boldsymbol{x}_i^0$ is in the convex hull of the points $\boldsymbol{x}_j^1$, $j \in G_k$. We add the constraint $\epsilon \leq 1$ to prevent unbounded solutions. Note that Problem (2.3) seeks to find a hyperplane $\boldsymbol{p}'\boldsymbol{x} + q = 0$ that separates point $\boldsymbol{x}_i^0$ from all the Class 1 points in Group k.

We want to expand Problem (2.3) for all $k \in \overline{K}$ and $i \in M_0$. If we knew which group each Class 1 point belonged to, then we would consider:

$$
\begin{aligned}
z = \text{maximize} \quad & \delta \\
\text{subject to} \quad & \boldsymbol{p}'_{k,i}\boldsymbol{x}_i^0 + q_{k,i} \leq -\delta, \quad i \in M_0; k \in \overline{K}, \\
& \boldsymbol{p}'_{k,i}\boldsymbol{x}_j^1 + q_{k,i} \geq 0, \quad i \in M_0; k \in \overline{K}; j \in G_k, \\
& \delta \leq 1.
\end{aligned} \tag{2.4}
$$

In order to determine if we can assign Class 1 points into K groups such that $z > 0$, we define decision variables for $k \in \overline{K}$ and $j \in M_1$:

$$
a_{k,j} = \begin{cases} 1, & \text{if } \boldsymbol{x}_j^1 \text{ is assigned to Group } k, \text{ (i.e., } j \in G_k), \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}
$$

We include the constraints $\boldsymbol{p}'_{k,i}\boldsymbol{x}_j^1 + q_{k,i} \geq 0$ in Problem (2.4) if and only if $a_{k,j} = 1$, i.e.,

$$
\boldsymbol{p}'_{k,i}\boldsymbol{x}_j^1 + q_{k,i} \geq M(a_{k,j} - 1),
$$

where M is a large positive constant. Note, however, that we can re-scale the variables

$\boldsymbol{p}_{k,i}$ and $q_{k,i}$ by a positive number, and thus we can take $M = 1$, i.e.,

$$\boldsymbol{p}'_{k,i}\boldsymbol{x}^1_j + q_{k,i} \geq a_{k,j} - 1.$$

Thus, we can check whether we can partition Class 1 points into K disjoint groups such that no Class 0 points are included in their convex hull, by solving the following mixed-integer optimization problem:

$$\begin{aligned} z^* = \text{maximize} \quad & \delta \\ \text{subject to} \quad & \boldsymbol{p}'_{k,i}\boldsymbol{x}^0_i + q_{k,i} \leq -\delta, && i \in M_0; k \in \overline{K}, \\ & \boldsymbol{p}'_{k,i}\boldsymbol{x}^1_j + q_{k,i} \geq a_{k,j} - 1, && i \in M_0; k \in \overline{K}; j \in M_1, \\ & \sum_{k=1}^{K} a_{k,j} = 1, && j \in M_1, \\ & \delta \leq 1, \\ & a_{k,j} \in \{0,1\}. \end{aligned} \tag{2.6}$$

If $z^* > 0$, the partition into K groups is feasible, while if $z^* = 0$, it is not, requiring us to increase the value of K.

2.3 The Clustering Algorithm

Problem (2.6) has $Km_0(d+1)+1$ continuous variables, Km_1 binary variables, and $Km_0 + Km_0m_1 + m_1$ rows. For large values of m_0 and m_1, Problem (2.6) becomes expensive to solve. Alternatively, we can drastically decrease the dimension of Problem (2.6) by solving a hyperplane for clusters of points at a time instead of point by point.

We develop a hierarchical clustering based algorithm that preprocesses the data to create clusters of Class 0 and Class 1 points. Collections of Class 0 (Class 1) points are considered a *cluster* if there are no Class 1 (Class 0) points in their convex hull. If we preprocess the data to find K_0 Class 0 clusters and K_1 Class 1 clusters, we can modify Problem (2.6) (see Formulation (2.9) below) to have $KK_0(d+1)+1$

continuous variables, KK_1 binary variables, and $Km_0 + KK_0m_1 + K_1$ rows.

The clustering algorithm applies the hierarchical clustering methodology (see [24]) where points or clusters with the shortest distances are merged into a cluster until the desired number of clusters is achieved. For our purposes, we need to check whether a merger of Class 0 (Class 1) clusters will not contain any Class 1 (Class 0) points in the resulting convex hull. We solve the following linear optimization problem to check whether Class 1 clusters r and s can be merged:

$$\begin{aligned} \delta^* = \text{maximize} \quad & \delta \\ \text{subject to} \quad & \boldsymbol{p}_i'\boldsymbol{x}_i^0 + q_i \leq -\delta, \quad i \in M_0, \\ & \boldsymbol{p}_i'\boldsymbol{x}_j^1 + q_i \geq \delta, \quad j \in C_r \cup C_s. \end{aligned} \tag{2.7}$$

where C_r and C_s are set of indices of Class 1 points in Clusters r and s, respectively.

If $\delta^* > 0$, then Clusters r and s can merge, while if $\delta^* = 0$, they can not since there is at least one Class 0 point in the convex hull of the combined cluster. The overall preprocessing algorithm that identifies clusters of Class 1 points is as follows:

1: **Initialize:** $K := m_1$, $k := 0$.
2: **while** $k < K$ **do**
3: Find the clusters with minimum pairwise distance — call these r and s.
4: Solve Problem (2.7) on Clusters r and s.
5: **if** $\delta^* = 0$ **then**
6: $k := k + 1$
7: **else**
8: Merge Clusters r and s.
9: $K := K - 1$, $k := 0$.
10: $k := k + 1$.

In the start of the algorithm, each point is considered a cluster, thus $K = m_1$. On Line 3, the minimum pairwise distances are calculated by comparing the statistical

distances[3] between the centers of all the clusters – known as the centroid method. We define the center of a cluster as the arithmetic mean of all the points that belong to that cluster. In the merging step on Line 4, these centers are updated. Finding clusters for Class 0 follows similarly.

After we have K_0 and K_1 clusters of Class 0 and Class 1 points, respectively, we run a modified version of Problem (2.6) to assign the K_1 Class 1 clusters to K groups, where $K < K_1 \ll m_1$. Let $\overline{K}_0 = \{1, \ldots, K_0\}$ and $\overline{K}_1 = \{1, \ldots, K_1\}$. Let C_t^0, $t \in \overline{K}_0$, be the set of indices of Class 0 points in Cluster t and C_r^1, $r \in \overline{K}_1$, be the set of indices of Class 1 points in Cluster r. We define the following binary variables for $r \in \overline{K}_1$ and $k \in \overline{K}$:

$$a_{k,r} = \begin{cases} 1, & \text{if Cluster } r \text{ is assigned to Group } k, \\ 0, & \text{otherwise.} \end{cases} \tag{2.8}$$

Analogously to Problem (2.6), we formulate the following mixed-integer optimization problem for clusters:

$$\begin{aligned} \text{maximize} \quad & \delta \\ \text{subject to} \quad & \boldsymbol{p}'_{k,t}\boldsymbol{x}_i^0 + q_{k,t} \leq -\delta, && i \in C_t^0; t \in \overline{K}_0; k \in \overline{K}, \\ & \boldsymbol{p}'_{k,t}\boldsymbol{x}_j^1 + q_{k,t} \geq a_{k,r} - 1, && t \in \overline{K}_0; r \in \overline{K}_1; k \in \overline{K}; j \in C_r^1, \\ & \sum_{k=1}^{K} a_{k,r} = 1, && r \in \overline{K}_1, \\ & a_{k,r} \in \{0, 1\}. \end{aligned} \tag{2.9}$$

If $a_{k,r} = 1$ in an optimal solution, then all Class 1 points in Cluster r are assigned to Group k, i.e., $G_k = \bigcup_{\{r | a_{k,r}=1\}} C_r^1$.

[3]Given a collection $\mathcal{F}$ of points, the *statistical* distance between points $\boldsymbol{x} \in \mathcal{F} \subseteq \mathbb{R}^d$ and $\boldsymbol{z} \in \mathcal{F} \subseteq \mathbb{R}^d$ is defined as

$$d_{\mathcal{F}}(\boldsymbol{x}, \boldsymbol{z}) = \sqrt{\sum_{j=1}^{d} \frac{(x_j - z_j)^2}{s_j^2}},$$

where s_j^2 is the sample variance of the jth coordinate of all points in $\mathcal{F}$. Statistical distance is a widely accepted metric in data mining to measure the proximity of points.

2.4 Elimination of Outliers

In the presence of outliers, it is possible that we may need a large number of groups – possibly leading to over-fitting. A point can be considered an outlier if it lies significantly far from any other point of its class (see Figure 2-3 for an illustration). In this section, we outline two methods that remove outliers: (a) based on the clustering algorithm of the previous section, and (b) via an extension of Problem (2.9).

Outlier Removal via Clustering

The clustering method of Section 2.3 applied on Class 0 points would keep outlier points in its own cluster without ever merging them with any other Class 0 cluster. Thus, after K_0 clusters are found, we can check the cardinality of each of the clusters and eliminate those with very small cardinality – perhaps less than 1% of m_0. Such a procedure can similarly be done on Class 1 points.

Outlier Removal via Optimization

Figure 2-3 illustrates how outlier points can prevent CRIO from grouping Class 1 points with small K, i.e., Problem (2.9) can only return a trivial solution where $\delta = 0$, $\boldsymbol{p}_{k,i} = \mathbf{0}$, $q_{k,i} = 0$ and $a_{k,j}$'s assigned arbitrarily. We want to modify Problem (2.9) to eliminate or ignore outlier points that prevent us from grouping Class 1 points into K groups.

One possible approach is to assign a binary decision variable to each point, so that it is removed if it is equal to 1, and not removed, otherwise. Such a modification can be theoretically incorporated into Formulation (2.9), but the large increase in binary variables can make the problem difficult to solve.

We propose a different approach that modifies Problem (2.9) to always return a partition of Class 1 points so that the total margin of the violation is minimized. Problem (2.10) is such a model, where ϵ_i^0 and ϵ_j^1 are violation margins corresponding

to Class 0 and Class 1 points, respectively. The model is as follows:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m_0} \epsilon_i^0 + \sum_{j=1}^{m_1} \epsilon_j^1 && (2.10)\\
\text{subject to} \quad & \boldsymbol{p}'_{k,t}\boldsymbol{x}_i^0 + q_{k,t} \leq -1 + \epsilon_i^0, && i \in C_t^0; t \in \overline{K}_0; k \in \overline{K},\\
& \boldsymbol{p}'_{k,t}\boldsymbol{x}_j^1 + q_{k,t} \geq -M + (M+1)a_{k,r} - \epsilon_j^1, && t \in \overline{K}_0; k \in \overline{K}; r \in \overline{K}_1; j \in C_r^1,\\
& \sum_{k=1}^{K} a_{k,r} = 1, && r \in \overline{K}_1,\\
& a_{k,r} \in \{0,1\}, \quad \epsilon_i^0 \geq 0, \quad \epsilon_j^1 \geq 0,
\end{aligned}
$$

where M is a large positive constant.

As in Problem (2.9), the first constraint of Problem (2.10) requires $\boldsymbol{p}'_{k,t}\boldsymbol{x}_i^0 + q_{k,t}$ to be strictly negative for all Class 0 points. However, if a point $\boldsymbol{x}_i^0$ can not satisfy the constraint, Problem (2.10) allows the constraint to be violated, i.e., $\boldsymbol{p}'_{k,t}\boldsymbol{x}_i^0 + q_{k,t}$ can be positive if $\epsilon_i^0 > 1$. Similarly, Problems (2.9) and (2.10) require $\boldsymbol{p}'_{k,t}\boldsymbol{x}_j^1 + q_{k,t}$ to be nonnegative when $a_{k,r} = 1$ and arbitrary when $a_{k,r} = 0$. However, (2.10) allows $\boldsymbol{p}'_{k,t}\boldsymbol{x}_j^1 + q_{k,t}$ to be negative even when $a_{k,r} = 1$, since the second constraint becomes $\boldsymbol{p}'_{k,t}\boldsymbol{x}_j^1 + q_{k,t} \geq 1 - \epsilon_j^1$ when $a_{k,r} = 1$, and the left-hand-side can take on negative values if $\epsilon_j^1 > 1$. Thus, by allowing ϵ_i^0 and ϵ_j^1 to be greater than 1 when necessary, Problem (2.10) will always return K Class 1 groups by ignoring those points that initially prevented the groupings. These points with $\epsilon_i^0 > 1$ and $\epsilon_j^1 > 1$ can be considered outliers and be eliminated.

2.5 Assigning Groups to Polyhedral Regions

The solution of Problem (2.10) results in K disjoint groups of Class 1 points, such that no Class 0 point is in the convex hull of any of these groups. Our objective in this section is to represent each Group k geometrically with a polyhedron P_k. An initially apparent choice for P_k is to use the solution of Problem (2.10), i.e.,

$$P_k = \{\boldsymbol{x} \in \mathbb{R}^d |\ \boldsymbol{p}'_{k,t}\boldsymbol{x} \geq -q_{k,t}, k \in \overline{K}; t \in \overline{K}_0\}.$$

Motivated by the success of SVMs (see [43]), we present an approach of using hyperplanes that separate the points of each class such that the minimum Euclidean distance from any point to the hyperplane is maximized. This prevents over-fitting the model to the training data set, since it leaves as much distance between the boundary and points of each class as possible.

Our goal is to find a hyperplane

$$\boldsymbol{\pi}'_{k,t}\boldsymbol{x} = \alpha_{k,t}$$

for every Group k, $k \in \overline{K}$, of Class 1 points and for every Cluster t, $t \in \overline{K}_0$, of Class 0 points such that all points in Cluster t are separated from every point in Group k so that the minimum distance between every point and the hyperplane is maximized. The distance, $d(\boldsymbol{x}, \boldsymbol{\pi}_{k,t}, \alpha_{k,t})$, between a point $\boldsymbol{x}$ and the hyperplane $\boldsymbol{\pi}'_{k,t}\boldsymbol{x} = \alpha_{k,t}$ is

$$d(\boldsymbol{x}, \boldsymbol{\pi}_{k,t}, \alpha_{k,t}) = \frac{|\gamma|}{\|\boldsymbol{\pi}_{k,t}\|}, \quad \text{where} \quad \gamma = \boldsymbol{\pi}'_{k,t}\boldsymbol{x} - \alpha_{k,t}.$$

Thus, we can maximize $d(\boldsymbol{x}, \boldsymbol{\pi}_{k,t}, \alpha_{k,t})$ by fixing $|\gamma|$ and minimize $\|\boldsymbol{\pi}_{k,t}\|^2 = \boldsymbol{\pi}'_{k,t}\boldsymbol{\pi}_{k,t}$, thus solving the quadratic optimization problem:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\pi}'_{k,t}\boldsymbol{\pi}_{k,t} \\ \text{subject to} \quad & \boldsymbol{\pi}'_{k,t}\boldsymbol{x}_i^0 \leq \alpha_{k,t} - 1, \quad i \in C_t^0, \\ & \boldsymbol{\pi}'_{k,t}\boldsymbol{x}_j^1 \geq \alpha_{k,t} + 1, \quad j \in G_k. \end{aligned} \tag{2.11}$$

We solve Problem (2.11) for each $t \in \overline{K}_0$ and $k \in \overline{K}$, and find KK_0 hyperplanes. Thus, for each Group k the corresponding polyhedral region is

$$P_k = \{\boldsymbol{x} \in \mathbb{R}^d | \ \boldsymbol{\pi}'_{k,t}\boldsymbol{x} \geq \alpha_{k,t}, \ t \in \overline{K}_0\}. \tag{2.12}$$

The last step in our process is the elimination of redundant hyperplanes in the representation of polyhedron P_k given in Eq. (2.12). Figures 2-5 and 2-6 illustrate this

procedure. We can check whether the constraint

$$\boldsymbol{\pi}'_{k,t_0}\boldsymbol{x} \geq \alpha_{k,t_0} \tag{2.13}$$

is redundant for the representation of P_k by solving the following linear optimization problem (note that the decision variables are $\boldsymbol{x}$):

$$\begin{aligned} w_{k,t_0} = \text{minimize} \quad & \boldsymbol{\pi}'_{k,t_0}\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{\pi}'_{k,t}\boldsymbol{x} \geq \alpha_{k,t}, \qquad t \in \overline{K}_0 \setminus \{t_0\}, \\ & \boldsymbol{\pi}'_{k,t_0}\boldsymbol{x} \geq \alpha_{k,t_0} - 1. \end{aligned} \tag{2.14}$$

We have only included the last constraint to prevent Problem (2.14) from becoming unbounded. If $w_{k,t_0} \geq \alpha_{k,t_0}$, then the Constraint (2.13) is implied by the other constraints defining P_k, and thus it is redundant. However, if $w_{k,t_0} < \alpha_{k,t_0}$, then Constraint (2.13) is necessary for describing the polyhedron P_k. To summarize, the following algorithm eliminates all redundant constraints:

```
for k = 1 to K do
    for t_0 = 1 to K_0 do
        Solve Problem (2.14).
        if w_{k,t_0} ≥ α_{k,t_0} then
            Eliminate Constraint π'_{k,t_0} x ≥ α_{k,t_0}.
```

2.6 The Overall Algorithm for Classification

The overall algorithm for classification is as follows:

1. **Preprocessing**: Use the clustering algorithm outlined in Section 2.3 to find clusters. Eliminate clusters with cardinality less than 1% of m_0 (m_1) for Class 0 (Class 1) clusters.

2. **Assign Clusters to Groups**: Solve the mixed-integer optimization problem (2.10) to assign clusters of Class 1 points to groups, while eliminating potential

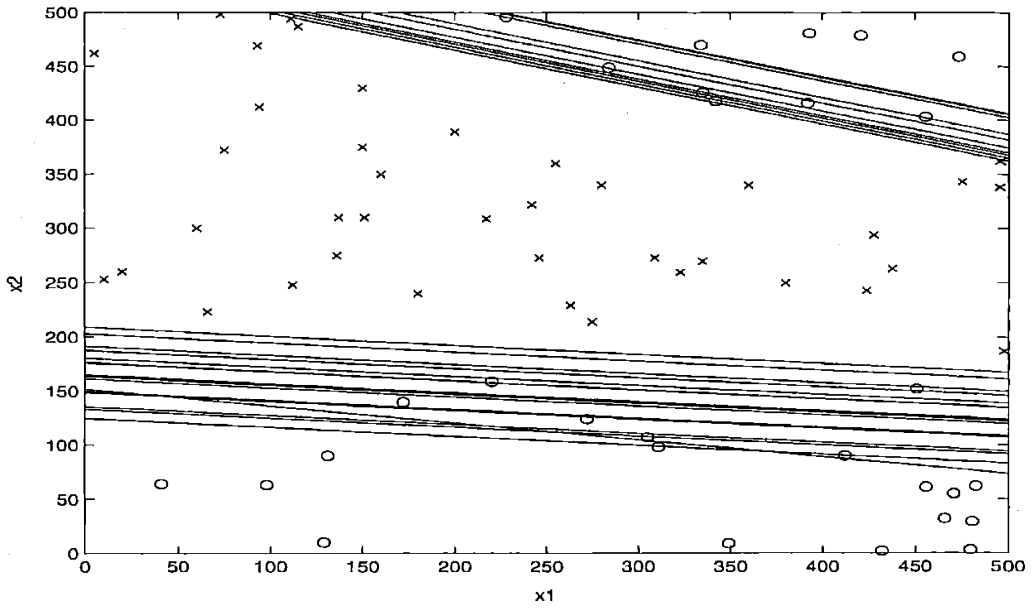

Figure 2-5: Before Eliminating Redundant Constraints.

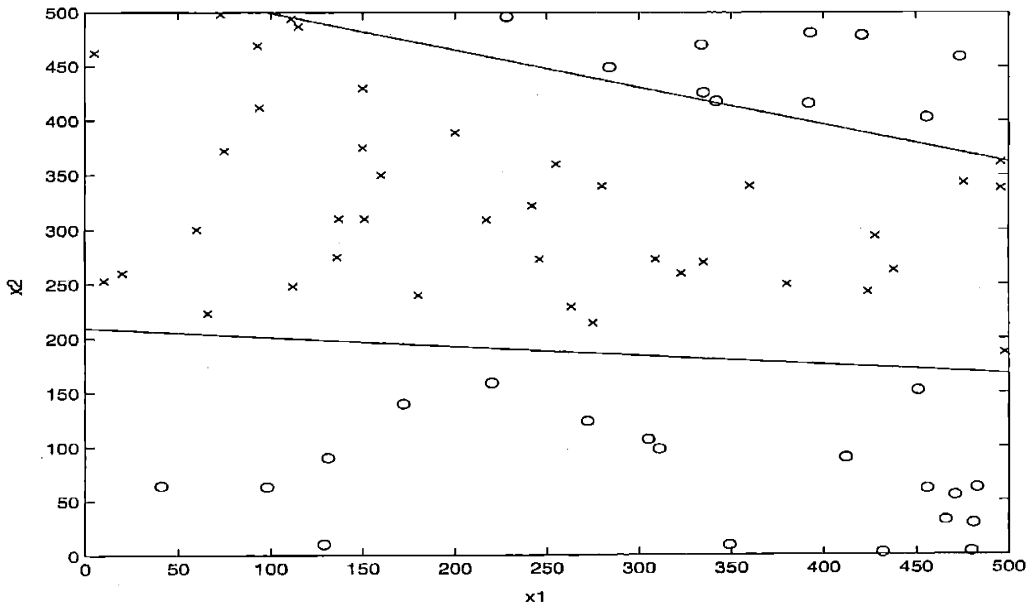

Figure 2-6: After Eliminating Redundant Constraints.

outliers.

3. **Assign Groups to Polyhedral Regions**: Solve the quadratic optimization problem (2.11) to find hyperplanes that define the polyhedron of each Class 1 group.

4. **Eliminate Redundant Constraints:** Remove redundant constraints from the polyhedra following the algorithm outlined in the end of Section 2.5.

After CRIO determines the non-redundant representations of the polyhedra, the model is used to predict the class of new data points. If the point lies in any of the K polyhedra, we label the point as a Class 1 point. If the point is not contained in any of the polyhedra, then we label the point as a Class 0 point.

2.7 Classification with Multiple Classes

We have only discussed binary classification problems so far, but the method can easily be extended to data sets with multiple classes by iteratively solving a binary classification problem. Suppose we have a data set with C classes, with N_i being the set of indices of class i points, $i = 1, 2, \ldots, C$. The following algorithm classifies such a data set:

1: **for** $c = 1$ to C **do**
2: Let $M_1 = N_c$, and $M_0 = \bigcup_{i=c+1,\ldots,C} N_i$.
3: Solve the binary classification problem using the method described in Section 2.6.

Thus, this algorithm finds polyhedra that separates Class 1 points from points of Class 2 through C, Class 2 points from points of Class 3 through C, etc. To classify a new point, we check whether it is contained in any Class 1 polyhedron, and if it does, we label the point Class 1; otherwise, we check whether it is contained in any Class 2 polyhedron. If it is not contained in any of the Class 1 through $C-1$ polyhedra, then we label it a Class C point. From computational experimentation, the method worked best when the classes are ordered in ascending cardinality, i.e., $|N_1| < |N_2| < \cdots < |N_C|$.

2.8 Stability Results for CRIO

As discussed in the previous sections, it is critical that a prediction method have strong prediction accuracy on the given data set, but it must also be robust to the randomness in the data. The prediction model would have no value even if it has good prediction accuracy on a particular sample of the data set, but fails on others. Thus, we want our algorithm to be stable – i.e., it's prediction error should not significantly vary with different random samples of the data.

The classification component of CRIO dealt with robustness by removing potential outliers via the clustering heuristic, allow small margins of error in its partition and maximize the minimum distance between the points and the the boundary of the points. Computational experimentations show that the variance on the prediction accuracy, when using different random samples of the training, validation and testing set, are relatively small (see Table 2.2). This seems to suggest that there exists bounds on the variability of its prediction error. In this section, we explore concepts of stability presented in the Machine Learning literature by Bousquet and Eliseeff [8]. We further extend these results to particular scenarios in CRIO.

Stability in Classification Algorithms

Randomness in classification can be attributed to two sources : one due to the random sampling of the training set, and the other due to measurement error. We focus on the former type, referred to as the *sampling randomness.* There has been significant work in the machine learning community in formalizing the measure of stability and generalization in terms of changes in the training set data. Specifically, [8] gives stability bounds for popular learning algorithms, such as regularized SVMs, when one data point is removed or replaced in a training set.

We first introduce some notation. Let $S = \{\boldsymbol{z}_1 = (\boldsymbol{x}_1, y_1), \ldots, \boldsymbol{z}_m = (\boldsymbol{x}_m, y_m)\}$ be a training set of size m. The data set, $Z = X \times Y$, is sampled i.i.d. from an unknown distribution D. We also look at a modified training set, $S^{\cap i}$, where $S^{\cap i} = S \setminus \{\boldsymbol{z}_i\}$. Let A be the algorithm that constructs a mapping $A_S : X \rightarrow Y$ using S as the training

set, i.e., A_S predicts the y-value for a given $\boldsymbol{x}$, where $A_S(\boldsymbol{x}) = \hat{y}$ is the predicted value using S as the training set, and $A_{S^{\cap i}}(\boldsymbol{x})$ is the predicted value using $S^{\cap i}$ as the training set.

Let the accuracy of the prediction be measured by the cost function $c : Y \times Y \rightarrow \mathbb{R}^+$. If $A_S(\boldsymbol{x})$ is the predicted value and y is the true value, then the cost of the prediction is $c(A_S(\boldsymbol{x}), y)$. In classification, this cost function is often an indicator function, i.e., $c(\bar{y}, y) = 1$ is $\bar{y} \neq y$ and $c(\bar{y}, y) = 0$ when $\bar{y} = y$. In the language of machine learning, one also defines a *loss* function, which measures the prediction loss of using the mapping A_S on data point $\boldsymbol{z} = (\boldsymbol{x}, y)$. Let ℓ be a loss function where $\ell(A_S, \boldsymbol{z}) = c(A_S(\boldsymbol{x}), y)$. This is defined mainly for notational purposes.

Ideally, a prediction algorithm wants to minimize the following criteria:

$$R(A, S) = E_z[\ell(A_S, \boldsymbol{z})]$$

which is a random variable with respect to the sampling S. This measure of error is known as the *generalization error* or the *true error*, which represents the expected error of the algorithm A on the training set S. However, we can only approximate this value since the distribution D is unknown. Assuming that the points in the set are selected i.i.d, the most common surrogate to the true error is the *empirical error*, R_{emp}, defined as

$$R_{emp}(A, S) = \frac{1}{m} \sum_{i=1}^{m} \ell(A_S, \boldsymbol{z}_i).$$

R_{emp} can be viewed as the sample average error of using algorithm A on training set S, and is often the criteria that is minimized in a prediction algorithm. Clearly, however, R_{emp} can only give an optimistic approximation of $R(A, S)$ since it is biased on the given sampled data set. Thus, an algorithm that simply minimizes the empirical error may have a true error that is significantly worse (which is the case of over-fitting). The goal of data mining models is prediction accuracy on future data sets, and not optimizing the fit on the given training set. Thus, we want to construct an algorithm whose empirical error does not depart significantly from the true error. Such an algorithm has to accept a greater number of training errors for it to be stable, but

these errors will be compensated by greater generalization.

The following notion of stability is used to construct bounds on $R - R_{emp}$:

Definition 1 (Uniform Stability) *An algorithm A has uniform stability δ with respect to the loss function ℓ if*

$$|\ell(A_S, .) - \ell(A_{S^{\cap i}}, .)| \leq \delta, \quad \forall S, \forall i, \forall \boldsymbol{z} \in Z. \tag{2.15}$$

An algorithm is considered *stable* if $\delta = O(\frac{1}{m})$.

Notice that for A to have uniform stability, this bound on the change in loss function needs to hold for every possible training set and future data points. There are weaker versions of stability that do not consider samples and data points that occur with probability 0. However, uniform stability allows us to get tighter bounds on the empirical error of an algorithm, as illustrated in the following theorem:

Theorem 1 (Bousquet & Elisseeff) *Let A be an algorithm with uniform stability β with respect to a loss function ℓ such that $0 \leq \ell(A_S, \boldsymbol{z}) \leq M$, for all $\boldsymbol{z} \in Z$ and all sets S. Then, for any $m \geq 1$, and any $\epsilon \in (0, 1)$, the following bounds hold with probability at least 1-ϵ over the random draw of the sample S:*

$$R \leq R_{emp} + 2\beta + (m\beta + M)\sqrt{\frac{\ln \frac{1}{\epsilon}}{2m}}, \tag{2.16}$$

which yields:

$$P_S[R - R_{emp} > \epsilon + 2\beta] \leq \exp\left(-\frac{2m\epsilon^2}{4m\beta + M}\right) \tag{2.17}$$

Let us measure the stability of the classification component of CRIO. We use the same notation as in previous sections, namely, L_0 is the number of Class 0 clusters, C_l^0 is the lth Class 0 cluster, K is the number of Class 1 groups and m_1 is the number of Class 1 points. To be consistent with traditional machine learning notation, let us use $y = -1$ as the class label for Class 0 points, i.e., the class labels are $Y = \{-1, 1\}$.

Let A_S be the real-valued classification function for CRIO that predicts the class of a point $\boldsymbol{x}$ by $sign(A_S(\boldsymbol{x}))$ (i.e., if $A_S(\boldsymbol{x}) \geq 0$, $\Rightarrow \hat{y} = sign(A_S(\boldsymbol{x})) = +1$. Otherwise,

$\hat{y} = sign(A_S(\boldsymbol{x})) = -1)$. Recall, that CRIO classifies a point as Class 1 if it is contained in one of the K polyhedra, P_k, defined by the set of linear SVMs that separate the L_0 Class 0 clusters from the kth Class 1 group. If $\boldsymbol{x}$ is not contained in any of the polyhedra, then it is classified as a Class 0 point. For notational simplicity, let us initially assume that $K = 1$.

Let $A_S^l(x) := \boldsymbol{\pi}_l'\boldsymbol{x} - \alpha_l$ be the resulting hyperplane that separates Class 0 Cluster l from the Class 1 group. From Section 2.5 and the definition of A_S and A_S^l, if $A_S^l(\boldsymbol{x}) \geq 0$ for all l, then $A_S(\boldsymbol{x}) \geq 0$. If $\exists l, l = 1, \ldots, L_0$, such that $A_S^l(\boldsymbol{x}) < 0$, then $A_S(\boldsymbol{x}) < 0$. We can clearly set $A_S(\boldsymbol{x}) := \min_{l=1\ldots,L_0} A_S^l(\boldsymbol{x})$, however, for purposes of theoretical analysis, we define A_S as follows:

$$A_S(\boldsymbol{x}) = \frac{1}{L_0} \sum_{l=1,\ldots,L_0} \rho(A_S^l(\boldsymbol{x}))$$

where

$$\rho(a) = \begin{cases} 0, & \text{if } a \geq 0, \\ a, & \text{if } a < 0. \end{cases}$$

Clearly, $sign\left(\frac{1}{L_0} \sum_{l=1,\ldots,L_0} \rho(A_S^l(\boldsymbol{x}))\right) = sign\left(\min_{l=1\ldots,L_0} A_S^l(\boldsymbol{x})\right)$ so both definitions are the same for our prediction purposes.

There are three possible scenarios when a point $\boldsymbol{x}_i$ is removed from the training set:

1. $\boldsymbol{x}_i$ is a Class 1 point, and its removal does not change the cluster membership of the Class 0 points,

2. $\boldsymbol{x}_i$ is a Class 0 point, and its removal does not change the cluster membership of the remaining Class 0 points, and

3. The removal $\boldsymbol{x}_i$ changes the cluster membership of the remaining points.

Lemma 2 and Corollary 1 give bounds to $|A_S(\boldsymbol{x}) - A_{S^{\cap i}}|$ for the first two cases.

Lemma 2 *If the point $\boldsymbol{x}_i$ removed from S is a Class 1 point, and A corresponds to the classification algorithm of CRIO. If its removal does not change the cluster*

membership of the Class 0 points, then:

$$|A_S(\boldsymbol{x}) - A_{S^{\cap i}}(\boldsymbol{x})| \leq \frac{1}{L_0} \sum_{l=1,\ldots,L_0} \frac{\kappa^2}{2(|C_l^0| + m_1)}, \quad \forall S, \forall \boldsymbol{x} \in X, \tag{2.18}$$

where κ is such that $\|\boldsymbol{x}\| \leq \kappa, \forall \boldsymbol{x} \in X$.

Proof: Removing $\boldsymbol{x}_i$ from the training set will not affect the allocation of the Class 1 points to their groups, which are constructed by the mixed-integer optimization model. It may, however, change the faces of the polyhedron P_k, which has up to L_0 faces (since there are L_0 Class 0 clusters), denoted as $A_S^l(\boldsymbol{x})$ above. From our definition, we have:

$$\begin{aligned}
|A_S(\boldsymbol{x}) - A_{S^{\cap i}}(\boldsymbol{x})| &= \left| \frac{1}{L_0} \sum_{l=1,\ldots,L_0} \rho(A_S^l(\boldsymbol{x})) - \frac{1}{L_0} \sum_{l=1,\ldots,L_0} \rho(A_{S^{\cap i}}^l(\boldsymbol{x})) \right|, \\
&\leq \frac{1}{L_0} \sum_{l=1,\ldots,L_0} |\rho(A_S^l(\boldsymbol{x})) - \rho(A_{S^{\cap i}}^l(\boldsymbol{x}))| \\
&\leq \frac{1}{L_0} \sum_{l=1,\ldots,L_0} |A_S^l(\boldsymbol{x}) - A_{S^{\cap i}}^l(\boldsymbol{x})| \quad \text{(Since } \rho(.) \text{ is clearly 1-Lipschitz)}
\end{aligned}$$

$A_S^l(x)$ are derived by linear SVMs and using the result from [8], we have

$$|A_S^l(\boldsymbol{x}) - A_{S^{\cap i}}^l(\boldsymbol{x})| \leq \frac{\kappa^2}{2(|C_l^0| + m_1)}.$$

□.

Corollary 1 *Suppose the point $\boldsymbol{x}_i$ removed from S is a Class 0 point and A corresponds to the classification algorithm of CRIO. If the removal of $\boldsymbol{x}_i$ does not change the cluster membership of the remaining Class 0 points, and if $\boldsymbol{x}_i \in C_{l(i)}^0$, then*

$$|A_S(\boldsymbol{x}) - A_{S^{\cap i}}(\boldsymbol{x})| \leq \frac{\kappa^2}{2L_0(|C_{l(i)}^0| + m_1)}, \quad \forall S, \forall \boldsymbol{x} \in X. \tag{2.19}$$

Proof: Assuming that the cluster assignments of the remaining points do not

change, then removing $\boldsymbol{x}_i$ will effect only $A_S^{l(i)}(x)$. Result (1) follows from the proof for Lemma 2 $\square$.

Finally, we extend Lemma 2 and Corollary 1 to when$K > 1$:

Corollary 2 *Under the conditions given in Lemma 2 or Corollary 1, we have*

$$|A_S(\boldsymbol{x}) - A_{S^{\cap i}}(\boldsymbol{x})| \leq \max_{k=1,\ldots,K} \sum_{l=1,\ldots,L_0} \frac{\kappa^2}{2L_0(|C_l^0| + |G_k|)} \tag{2.20}$$

where G_k are the set of Class 1 points in Group k.

Proof: Suppose $A_S^{l,k}$ is the linear SVM that separates Class 0 Cluster l from Class 1 Group k, and let $A_S^k = \frac{1}{L_0} \sum_{l=1,\ldots,L_0} \rho(A_S^l(\boldsymbol{x}))$ (i.e., A_S^k is the same as A_S when $k = 1$). If A_S is the classification function, then $A_S(\boldsymbol{x}) \geq 0$ if $\exists k$, such that $\boldsymbol{x} \in P_k$. Otherwise, $A_S(\boldsymbol{x}) < 0$. Thus, we set $A_S(\boldsymbol{x}) := \max_{k=1,\ldots,K} A_S^k$. From Lemma 2 and Corollary 1, we get the following:

$$\begin{aligned}
|A_S(\boldsymbol{x}) - A_{S^{\cap i}}(\boldsymbol{x})| &\leq |\max_{k=1,\ldots,K} A_S^k(\boldsymbol{x}) - \max_{k=1,\ldots,K} A_{S^{\cap i}}^k(\boldsymbol{x})| \\
&\leq \max_k |A_S^k(\boldsymbol{x}) - A_{S^{\cap i}}^k(\boldsymbol{x})| \\
&\leq \max_{k=1,\ldots,K} \sum_{l=1,\ldots,L_0} \frac{\kappa^2}{2L_0(|C_l^0| + |G_k|)}
\end{aligned}$$

$\square$.

When the cost function $c(.,.)$ is an indicator function, we can only show trivial cases of stability due to the discontinuity of the loss function. Thus, we modify the cost function to the following continuous function:

$$c_\gamma(\hat{y}, y) = \begin{cases} 1, & \text{if } \hat{y}y < 0, \\ 1 - \frac{\hat{y}y}{\gamma}, & \text{if } 0 \leq \hat{y}y < \gamma, \\ 0, & \text{otherwise,} \end{cases} \tag{2.21}$$

where γ is a predetermined constant. We similarly define $\ell_\gamma(A_S, \boldsymbol{z}) = c_\gamma(A_S(\boldsymbol{x}), y)$. $c_\gamma(\hat{y}, y)$ is $\frac{1}{\gamma}$-Lipschitz with respect to its first argument, which give us the following lemma:

Lemma 3 *Under the conditions of Corollary 2 and using the cost function (2.21), we have the following result for all training set S and all $\boldsymbol{z}$:*

$$|\ell(A_S, \boldsymbol{z}) - \ell(A_{S^{\cap i}}, \boldsymbol{z})| \leq \frac{1}{\gamma L_0} \max_{k=1,\ldots,K} \sum_{l=1,\ldots,L_0} \frac{\kappa^2}{2(|C_l^0| + |G_k|)} \tag{2.22}$$

Proof: From definition,

$$\begin{aligned} |\ell(A_S, \boldsymbol{z}) - \ell(A_{S^{\cap i}}, \boldsymbol{z})| &\leq |c_\gamma(A_S(\boldsymbol{x}), y) - c_\gamma(A_{S^{\cap i}}(\boldsymbol{x}), y)| \\ &\leq \frac{1}{\gamma}|A_S(\boldsymbol{x}) - A_{S^{\cap i}}(\boldsymbol{x})| \\ &\leq \frac{1}{\gamma L_0} \max_{k=1,\ldots,K} \sum_{l=1,\ldots,L_0} \frac{\kappa^2}{2(|C_l^0| + m_1)} \end{aligned}$$

□.

Lemma 3 comes close to stating that the CRIO has uniform convergence β, where $\beta = (1\backslash m)$. However, the lemma does not cover the scenario when the removal of a point changes the cluster memberships. In most cases, removing a point would not significantly change cluster membership when running hierarchical clustering of Section 2.3. For example, when the variance or spread of the points in a given cluster is relatively small and the cardinality large, removing a single point from this cluster will not affect the existing clusters. Also, removing a point from a cluster of large variance and small cardinality will also not change the clusters. However, in the less extreme cases, it is difficult to construct any stability bounds. This is characteristic of clustering algorithms in general. Although there has been significant work on stability bounds for supervised learning algorithms [8, 14, 13], we have not found any theoretical studies on cluster stability in terms of sampling randomness. However, empirical tests and intuition suggest that bounds on the classification function should not differ significantly from Equation 2. Once we can show that $|A_S(\boldsymbol{x}) - A_{S^{\cap i}}(\boldsymbol{x})| \leq \beta = O(1\backslash m)$ under scenario 3, then we can show an exponential bound for CRIO using Theorem 1 with $M = 1$ and $m = m_0 + m_1$.

2.9 Computational Results

We tested the classification component of CRIO on real data, and compared their prediction accuracy against the softwares CART[4] and SvmFu[5]. Each data set was split into three parts – the training set, the validation set and the testing set. The training set, comprised of 50% of the data, was used to develop the model. The validation set, composed of 30% of the data, was used to select the best values of K, K_0 and K_1. Finally, the remaining points are in the testing set, which ultimately decides the accuracy of the model. This set is put aside until the very end, after the parameters of the model are finalized. The assignment to each of the three sets was done randomly, and this process was repeated three times.

CART uses the validation set for pruning its initial decision tree built by the training set alone. The test set is classified using this final tree. For SvmFu, linear, polynomial (with degree 2 and 3) and gaussian kernels were all tested, as well as different cost per unit violation of the margin. The kernel and parameters resulting in the best validation set accuracy was chosen to classify the testing set.

We solved all optimization problems (mixed-integer, quadratic and linear) using CPLEX 7.1[6] [12] running in a Linux environment.

Classification Data

We tested our models on five real data sets found on the UCI data repository [7]. The "Cancer" data is from the Wisconsin Breast Cancer databases, with 682 data points and 9 explanatory variables. The "Liver" data is from BUPA Medical Research Ltd., with 345 data points and 6 explanatory variables. The "Diabetes" data is from the Pima Indian Diabetes Database, with 768 data points and 7 explanatory variables. The "Heart" data is from the SPECT heart database, where we combined the given

[4]CART is a product of Salford Systems, http://www.salford-systems.com.

[5]SvmFu is a freeware developed by Ryan Rifkin from MIT that implements SVMs with linear, polynomial and gaussian kernels. It can be downloaded from http://five-percent-nation.mit.edu/SvmFu/.

[6]CPLEX 7.1 is a product of ILOG http://www.ilog.com/products/cplex/

[7]http://www.ics.uci.edu/~mlearn/MLSummary.html

training and testing set to get 349 data points and 44 explanatory variables. The "Iris" data determines the type of iris, given the plant's physical measurements, and has 150 data points, 4 explanatory variables and 3 classes. The "Cancer", "Liver", "Diabetes" and "Heart" involve binary classification, while the "Iris" data set involves classification with three classes.

Results

Table 2.1 summarizes the classification accuracy of CART, SvmFu and CRIO for the five data sets. Each sub-panel in Table 2.1 corresponds to a data set. Each data set was partitioned using random sampling into training, validation and testing set, and we illustrate the average of these partitions. The columns labelled "Train", "Validation" and "Test" illustrate the percent of the training, validation and testing set, respectively, that the models correctly classified.

In all data sets, CRIO was significantly more successful than CART, with prediction accuracy up to 13% higher in the testing set. CRIO outperformed SvmFu in the "Liver" and "Diabetes" data sets, with prediction accuracy up to 10% higher. In the "Cancer" and "Iris" data set CRIO and SvmFu had the same performance, and in the "Heart" data set SvmFu outperformed CRIO. Most importantly, the impact of CRIO was greatest in the data sets that tend to be difficult to classify for the current methods ("Liver" and "Diabetes"). CRIO runs in practical running times of under half a minute for all data sets on a personal workstation.

The parameter M of Problem (2.10) was set to 1000 for all of our experiments. Both the values of K_0 and K_1 ranged from 1 to 3 in "Cancer" and "Heart", 4 to 8 in "Iris" and 8 to 12 in "Liver" and "Diabetes". "Cancer","Heart" and "Iris" used $K = 1$ group for all three cases, "Liver" used up to $K = 2$ groups and "Diabetes" up to $K = 3$ groups. We feel that CRIO's ability, via mixed-integer optimization, to categorize the points in more than one region may explain the prediction accuracy of CRIO in the "Liver" and "Diabetes" data sets.

Table 2.2 illustrates the standard deviation of the prediction accuracy across all the partitions. We use it to give us a rough empirical picture of the stability of

Cancer $n = 682, d = 9$	**Train** (%)	**Validation** (%)	**Test** (%)
CART	95.37	93.83	95.41
SvmFu	97.07	96.41	98.79
CRIO	97.26	96.90	98.79

Liver $n = 345, d = 6$	**Train** (%)	**Validation** (%)	**Test** (%)
CART	76.93	68.97	63.33
SvmFu	80.43	70.23	64.76
CRIO	73.06	74.43	71.43

Diabetes $n = 768, d = 8$	**Train** (%)	**Validation** (%)	**Test** (%)
CART	83.20	71.27	69.05
SvmFu	80.99	71.74	70.56
CRIO	80.12	79.42	77.06

Heart $n = 349, d = 44$	**Train** (%)	**Validation** (%)	**Test** (%)
CART	83.37	76.93	74.18
SvmFu	100.00	88.46	86.38
CRIO	99.81	83.65	81.69

Iris $n = 150, d = 4$	**Train** (%)	**Validation** (%)	**Test** (%)
CART	97.78	96.30	91.11
SvmFu	98.67	98.52	96.67
CRIO	99.56	97.04	96.67

Table 2.1: Prediction accuracy rates of CART, SvmFu and CRIO. n is the number of data points, and d is the number of explanatory variables.

the algorithms, as discussed in Section 2.8. Although, the size of the data set and the number of random partitions are too small to construct statistical measure of error and confidence intervals, it shows that there are no large deviations in the prediction error for all models. For the most part, it reflects the theoretical findings that SVMs are more stable to sampling randomness than CART [38, 20] (again, the major discrepancies are probably due mainly to the size of the data set). It also shows that the variability of CRIO's predictions are comparable to those of SVMs,

Cancer	Train	Validation	Test
CART	2.15	0.64	0.84
SvmFu	0.00	0.69	0.00
CRIO	1.19	1.58	0.42
Liver	**Train**	**Validation**	**Test**
CART	10.83	1.67	9.07
SvmFu	16.95	7.29	8.12
CRIO	3.31	5.35	3.78
Diabetes	**Train**	**Validation**	**Test**
CART	3.66	10.38	13.78
SvmFu	17.15	10.03	5.60
CRIO	1.10	4.30	2.30
Heart	**Train**	**Validation**	**Test**
CART	15.13	14.35	4.53
SvmFu	0.00	6.00	4.95
CRIO	0.33	5.09	7.04
Iris	**Train**	**Validation**	**Test**
CART	1.54	1.28	1.92
SvmFu	0.00	1.28	3.33
CRIO	0.77	1.28	3.33

Table 2.2: Standard deviation of prediction accuracy of CART, SvmFu and CRIO.

as the results in Section 2.8 suggested. Interestingly, the standard deviation of CRIO is significantly lower than those of CART and SVM in the "Liver" and "Diabetes" data sets. Those two data sets had large value for K than others, thus would be more prone to over-fitting. By the poor predictions of CART and SVM, perhaps the low variance of CRIO implies under-fitting of the first two methods, and that CRIO with $K = 2$ and $K = 3$ are good fits for "Liver" and "Diabetes", respectively.

2.10 Conclusion

The classification component of CRIO represents a new approach for solving classification problems. It combines the use of clustering techniques to alleviate the computational burden, pinpoints outliers before and during the model building step and uses linear and nonlinear optimization methods to construct polyhedral regions that define borders for each class. The key difference and advantage, however, is the

use of mixed-integer optimization models that are able to capture discrete behaviors in the data set.

The effective combination of these techniques allows CRIO to overcome some of the shortcomings of the current leading methods of CART and SVM. Unlike CART, CRIO is able to partition the explanatory variable space in a globally optimal manner, and not limited to hyper-rectangular regions. CRIO expands the concepts of linear SVMs in to general polyhedral regions, without utilizing nonlinear kernels. Although gaussian kernels have been shown to be very effective in separating the training set data, it is often seen to over-fit (even when cross validation is used) and difficult to intuitively visualize. Most importantly, the computational results of Section 2.9 illustrates the predictive powers of CRIO, especially on the data set "Liver" and "Diabetes" which the other methods had the most difficulty predicting.

Clearly, there are several follow-up works that can be done to improve CRIO's performance on classification problems. We would like to further test on larger data sets to see the scalability of CRIO in both the number of data points and dimension. It would be interesting to compare any differences in its predictive performance compared to SVMs which have been known to be successful in higher dimensions. Also, as we will touch upon in Chapter 4, variable selection is a key issue in both classification and regression. As the size of the data sets become larger, not all of the available explanatory variables are necessary in the model. To add robustness and simplicity to the model, we may want to find ways to effectively and efficiently implement this variable selection process.

Chapter 3

Regression via Integer Optimization

In a classical regression setting, we are given n data points $(\boldsymbol{x}_i, y_i)$, but y_i can take any real value, i.e. $y_i \in \mathbb{R}$, and is referred to as the *predicted* or *target variable.* We wish to find a simple mapping from the explanatory variables to the predicted variable, so that given new sets of explanatory variables, we can accurately predict their target values.

Two popular methods in regression are CART and Multivariate Adaptive Regression Splines (MARS). The regression component of CART similarly splits the space into hyper-rectangular regions, but assigns a constant predicted value to each of the regions. It is, thus, essentially fitting a step-wise function to the data points. MARS similarly partitions the explanatory variable space, but fits one dimensional linear splines to each region, while maintaining continuity of the predicted values among neighboring regions. Again, the main shortcomings of both CART and MARS is its greedy approach in constructing the different regions.

Similar to the classification component, the regression component of CRIO separates the explanatory variable space in a globally optimal manner via the use of mixed-integer optimization. For each region, it finds a linear mapping that minimizes the total absolute error. The following gives a geometric overview of our approach.

3.1 The Geometry of the Regression Approach

In a classical regression setting, we are given n data points $(\boldsymbol{x}_i, y_i)$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$ and $i = 1, \ldots, n$. We wish to find a linear relationship between $\boldsymbol{x}_i$ and y_i, i.e., $y_i \approx \boldsymbol{\beta}'\boldsymbol{x}_i$ for all i, where the coefficients $\boldsymbol{\beta} \in \mathbb{R}^d$ are found by minimizing $\sum_{i=1}^n (y_i - \boldsymbol{\beta}'\boldsymbol{x}_i)^2$ or $\sum_{i=1}^n |y_i - \boldsymbol{\beta}'\boldsymbol{x}_i|$ (for notational simplicity, we assume that the first element of every vector $\boldsymbol{x}_i$ is equal to 1, so β_1 corresponds to the intercept term). CRIO seeks to find K disjoint regions $P_k \subset \mathbb{R}^d$ and corresponding coefficients $\boldsymbol{\beta}_k \in \mathbb{R}^d$, $k = 1, \ldots, K$, such that if $\boldsymbol{x}_0 \in P_k$, our prediction for y_0 will be $\hat{y}_0 = \boldsymbol{\beta}_k'\boldsymbol{x}_0$. Figure 3-2 illustrates the output of CRIO, given the points in Figure 3-1 where $\boldsymbol{x}_i \in \mathbb{R}$.

CRIO first solves a mixed-integer optimization model to assign the n points into K groups (where K is a user-defined parameter). In addition, the mixed-integer optimization is further enhanced to detect and eliminate outlier points in the data set (see Figure 3-3). In contrast, traditional regression models deal with outliers *after* the slopes have been determined, by examining which points contribute the most to the total prediction error (see [39], [40]). This procedure can often be deceiving since the model is heavily influenced by the outlier points. After the points are assigned to the K groups, we determine the coefficients $\boldsymbol{\beta}_k$ that best fit the data for Group k, for $k = 1, \ldots, K$, and define polyhedra P_k to represent each group using linear optimization methods. After the coefficients and polyhedra are defined, we predict the y_0 value of a new point $\boldsymbol{x}_0$ as we outlined earlier. CRIO does not in fact create a partition of $\mathbb{R}^d$, so it is possible that a new point $\boldsymbol{x}_0$ may not belong to any of the P_k's [1]. In such a case, we assign it to the region P_r that contains the majority among its F (a user-defined number) nearest neighbors in the training set, and make the prediction $\hat{y}_0 = \boldsymbol{\beta}_r'\boldsymbol{x}_0$. Similarly to the classification model, we preprocess the data by clustering them into small clusters to reduce the dimension and thus the computation time of the mixed-integer optimization model (see Figure 3-4).

In summary, CRIO has a common approach to both problems of classification

[1] Note that in classification, the regions P_k also do not form a partition of $\mathbb{R}^d$, but if a point $\boldsymbol{x}_0$ belongs to any of the regions P_k, we classify it as Class 1, and if it does not belong to any of the regions P_k, we classify it as Class 0.

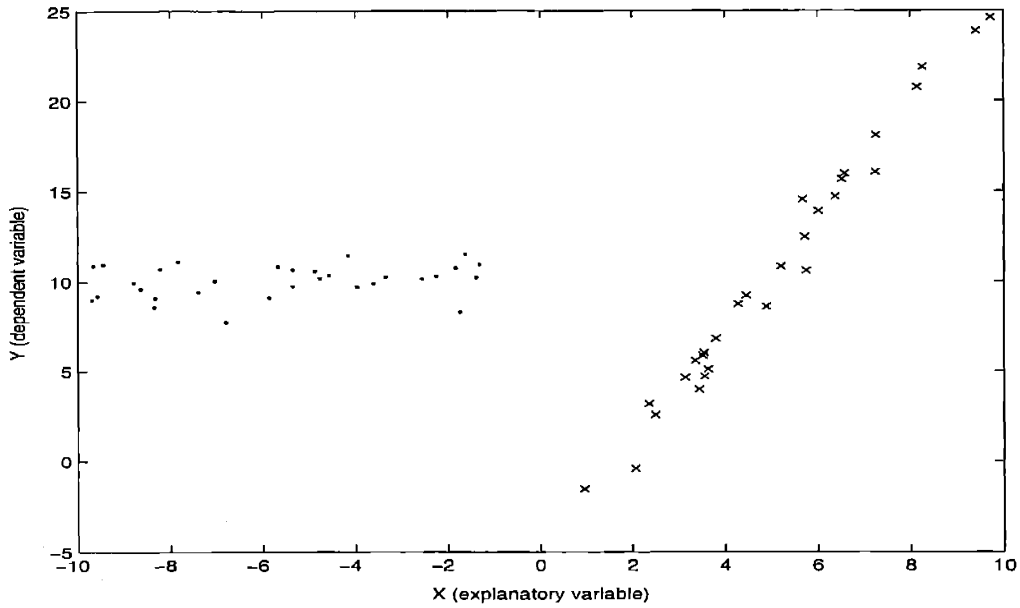

Figure 3-1: A given set of training data for regression with $d = 1$.

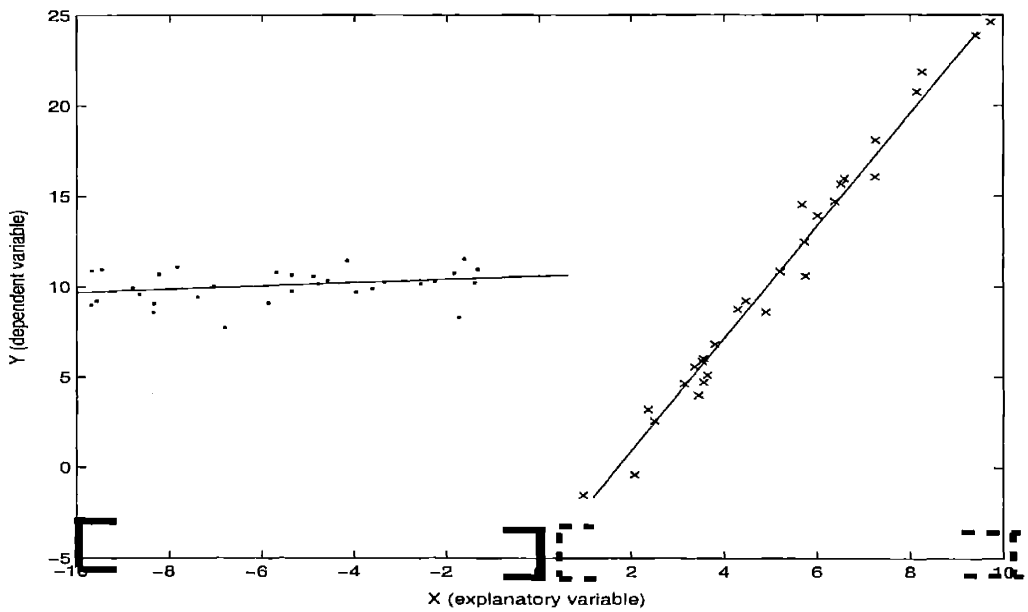

Figure 3-2: The output of CRIO, where the regression coefficient or slope is different for the two regions.

and regression: (a) Preprocess data by assigning points to clusters to reduce the dimensionality of the mixed-integer optimization problems solved next; (b) Solve

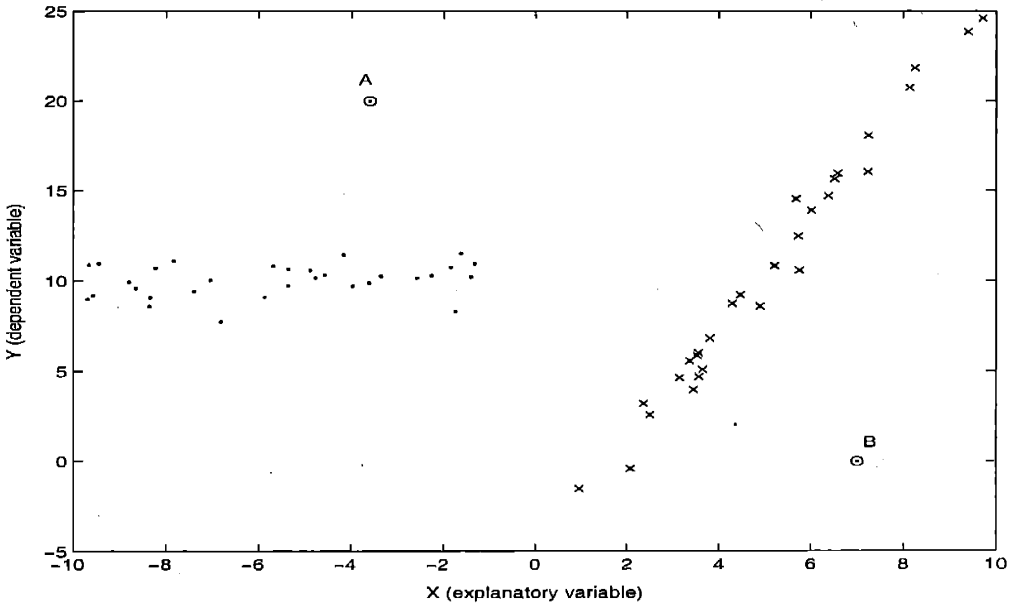

Figure 3-3: Illustration of outliers in regression data.

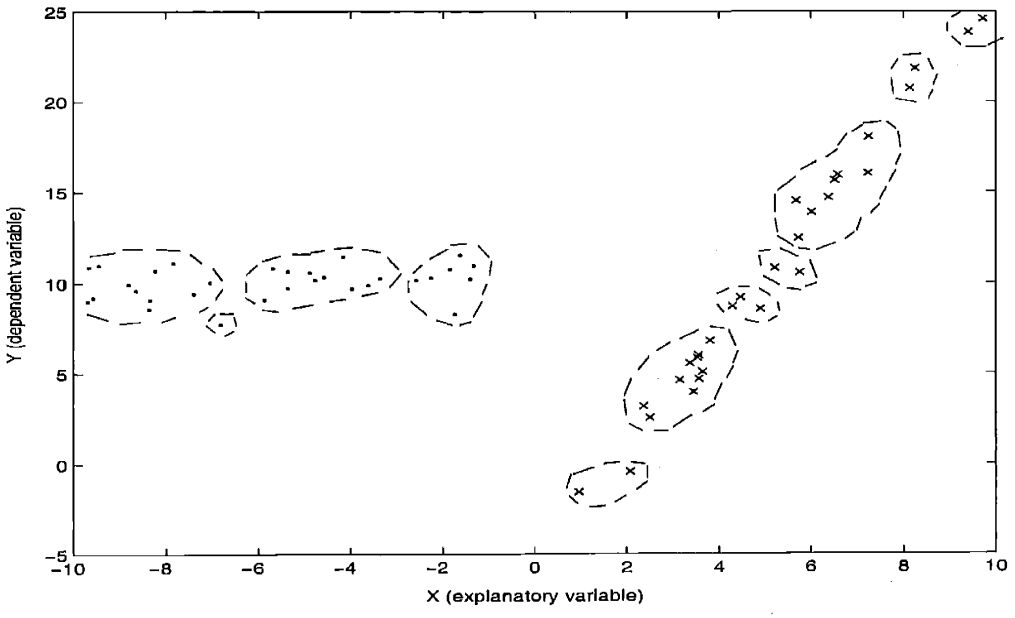

Figure 3-4: Data points clustered in (x, y)-space.

a mixed integer-optimization problem that assigns clusters to groups and removes outliers. In the case of regression the model also selects the regression coefficients

for each group; (c) Solve continuous optimization problems (quadratic optimization problems for classification and linear optimization problems for regression) that assign groups to polyhedral regions.

In the following sections, we present in detail our approach for regression. For presentation purposes, we start in Section 3.2 with an initial mixed-integer optimization model to assign points to groups, which, while not practical because of dimensionality problems, forms the basis of our approach. As outlined in Section 3.1, we first assign points to clusters (Section 3.3), then assign clusters to groups of points (Section 3.4), which we then represent by polyhedral regions P_k (Section 3.5). In Section 3.6, we propose a method of automatically finding nonlinear transformations of the explanatory variables to improve the predictive power of the method. Finally, we present the overall algorithm in Section 3.7.

3.2 Assigning Points to Groups: An Initial Model

The training data consists of n observations $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, n$, with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. We let $N = \{1, \ldots, n\}$, $\overline{K} = \{1, \ldots, K\}$ and M be a large positive constant. We define binary variables for $k \in \overline{K}$ and $i \in N$:

$$a_{k,i} = \begin{cases} 1, & \text{if } \boldsymbol{x}_i \text{ assigned to Group } k, \\ 0, & \text{otherwise.} \end{cases}$$

The mixed-integer optimization model is as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} \delta_i \\ \text{subject to} \quad & \delta_i \geq (y_i - \boldsymbol{\beta}_k' \boldsymbol{x}_i) - M(1 - a_{k,i}), \quad k \in \overline{K}; i \in N, \\ & \delta_i \geq -(y_i - \boldsymbol{\beta}_k' \boldsymbol{x}_i) - M(1 - a_{k,i}), \quad k \in \overline{K}; i \in N, \\ & \sum_{k=1}^{K} a_{k,i} = 1, \qquad i \in N, \\ & a_{k,i} \in \{0, 1\}, \quad \delta_i \geq 0. \end{aligned} \tag{3.1}$$

From the first and second constraints, δ_i is the absolute error associated with point $\boldsymbol{x}_i$. If $a_{k,i} = 1$, then $\delta_i \geq (y_i - \boldsymbol{\beta}_k' \boldsymbol{x}_i)$, $\delta_i \geq -(y_i - \boldsymbol{\beta}_k' \boldsymbol{x}_i)$, and the minimization of δ_i sets δ_i equal to $|y_i - \boldsymbol{\beta}_k' \boldsymbol{x}_i|$. If $a_{k,i} = 0$, the right-hand-side of the first two constraints becomes negative, making them irrelevant since δ_i is nonnegative. Finally, the third constraint limits the assignment of each point to just one group.

We have found that even for relatively small n ($n \approx 200$), Problem (3.1) is difficult to solve in reasonable time. For this reason, we initially run a clustering algorithm, similar to that of Section 2.3, to cluster nearby $\boldsymbol{x}_i$ points together. After L such clusters are found, for $L \ll n$, we can solve a mixed-integer optimization model, similar to Problem (3.1), but with significantly fewer binary decision variables.

3.3 The Clustering Algorithm

We apply a nearest neighbor clustering algorithm in the combined $(\boldsymbol{x}, y)$ space, as opposed to just the $\boldsymbol{x}$ space, in order to find L clusters. Specifically, the clustering algorithm initially starts with n clusters, then continues to merge the clusters with points close to each other until we are left with L clusters. More formally, the clustering algorithm is as follows:

1: **Initialize:** $k = n$. $C_i = \{i\}$, $i = 1, \ldots, n$.
2: **while** $l < L$ **do**
3: Find the points $(\boldsymbol{x}_i, y_i)$ and $(\boldsymbol{x}_j, y_j)$, $i < j$, with minimum pairwise statistical distance. Let $l(i)$ and $l(j)$ be the indices of the clusters that $(\boldsymbol{x}_i, y_i)$ and $(\boldsymbol{x}_j, y_j)$ currently belong to, respectively.
4: Add Cluster $l(j)$'s points to Cluster $l(i)$, i.e., $C_{l(i)} := C_{l(i)} \cup C_{l(j)}$.
5: Let the pairwise statistical distance between all the points in $C_{l(i)}$ be ∞.
6: $l = l - 1$.

In the clustering algorithm for classification problems (Section 2.3), we merged clusters who had centers of close proximity. However, in the present clustering algorithm, we merge clusters which contains points of close proximity – known as the single-linkage methods. Computational experimentations showed that this latter

method of clustering suited the regression problem better.

3.4 Assigning Points to Groups: A Practical Approach

Although we can continue the clustering algorithm of the previous section until we find K clusters, define them as our final groups, and find the best $\boldsymbol{\beta}_k$ coefficient for each group by solving separate linear regression problems, such an approach does not combine points in order to minimize the total absolute error. For this reason, we use the clustering algorithm until we have L, $L > K$, clusters and then solve a mixed-integer optimization model that assigns the L clusters into K groups in order to minimize the total absolute error. Another key concern in regression models is the presence of outliers. The mixed-integer optimization model we present next is able to remove potential outliers by eliminating points in clusters that tend to weaken the fit of the predictor coefficients.

Let C_l, $l \in \overline{L} = \{1, \ldots, L\}$, be Cluster l, and denote $l(i)$ as $\boldsymbol{x}_i$'s cluster. Similarly to Problem (3.1), we define the following binary variables for $k \in \overline{K} \cup \{0\}$ and $l \in \overline{L}$:

$$a_{k,l} = \begin{cases} 1, & \text{if Cluster } l \text{ is assigned to Group } k, \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

We define $k = 0$ as the outlier group, in the sense that points in Cluster l with $a_{0,l} = 1$ will be eliminated. The following model assigns clusters to groups and allows

the possibility of eliminating clusters of points as outliers:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \delta_i \\
\text{subject to} \quad & \delta_i \geq (y_i - \boldsymbol{\beta}_k' \boldsymbol{x}_i) - M(1 - a_{k,l(i)}), \quad k \in \overline{K}; i \in N, \\
& \delta_i \geq -(y_i - \boldsymbol{\beta}_k' \boldsymbol{x}_i) - M(1 - a_{k,l(i)}), \quad k \in \overline{K}; i \in N, \\
& \sum_{k=0}^{K} a_{k,l} = 1, \qquad l \in \overline{L}, \\
& \sum_{l=1}^{L} |C_l| a_{0,l} \leq \rho |N|, \\
& a_{k,l} \in \{0,1\}, \quad \delta_i \geq 0,
\end{aligned}
\tag{3.3}
$$

where M is a large positive constant, and ρ is the maximum fraction of points that can be eliminated as outliers.

From the first and second set of constraints, δ_i is the absolute error associated to point $\boldsymbol{x}_i$. If $a_{k,l(i)} = 1$, then $\delta_i \geq (y_i - \boldsymbol{\beta}_k' \boldsymbol{x}_i)$, $\delta_i \geq -(y_i - \boldsymbol{\beta}_k' \boldsymbol{x}_i)$, and the minimization of δ_i sets it equal to $|y_i - \boldsymbol{\beta}_k' \boldsymbol{x}_i|$. If $a_{k,l(i)} = 0$, the first two constraints become irrelevant, since δ_i is nonnegative. The third set of constraints limits the assignment of each cluster to just one group (including the outlier group). The last constraint limits the percentage of points eliminated to be less than or equal to a pre-specified number ρ. If $a_{k,l} = 1$, then all the points in Cluster l are assigned to Group k, i.e., $G_k = \bigcup_{\{l | a_{k,l}=1\}} C_l$.

Problem (3.3) has KL binary variables as opposed to Kn binary variables in Problem (3.1). The number of clusters L controls the tradeoff between quality of the solution and efficiency of the computation. As L increases, the quality of the solution increases, but the efficiency of the computation decreases. In Section 3.8 we discuss appropriate values for K, L and ρ from our computational experimentation.

3.5 Assigning Groups to Polyhedral Regions

We identify K groups of points solving Problem (3.3). In this section, we establish a geometric representation of Group k by a polyhedron P_k.

It is possible for the convex hulls of the K groups to overlap, and thus, we may not be able to define disjoint regions of P_k that contains all the points of Group k. For this reason, our approach is based on separating pairs of groups with the objective of minimizing the sum of violations. We first outline how to separate Group k and Groups r, where $k < r$. We consider the following two linear optimization problems:

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in G_k} \epsilon_i + \sum_{l \in G_r} \epsilon_l \\ \text{subject to} \quad & \boldsymbol{p}'_{k,r}\boldsymbol{x}'_i - q_{k,r} \leq -1 + \epsilon_i, \quad i \in G_k, \\ & \boldsymbol{p}'_{k,r}\boldsymbol{x}_l - q_{k,r} \geq 1 - \epsilon_l, \quad l \in G_r, \\ & \boldsymbol{p}'_{k,r}\boldsymbol{e} \geq 1, \\ & \epsilon_i \geq 0, \quad \epsilon_l \geq 0, \end{aligned} \tag{3.4}$$

and

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in G_k} \epsilon_i + \sum_{l \in G_r} \epsilon_l \\ \text{subject to} \quad & \boldsymbol{p}'_{k,r}\boldsymbol{x}_i - q_{k,r} \leq -1 + \epsilon_i, \quad i \in G_k, \\ & \boldsymbol{p}'_{k,r}\boldsymbol{x}_l - q_{k,r} \geq 1 - \epsilon_l, \quad l \in G_r, \\ & \boldsymbol{p}'_{k,r}\boldsymbol{e} \leq -1, \\ & \epsilon_i \geq 0, \quad \epsilon_l \geq 0, \end{aligned} \tag{3.5}$$

where $\boldsymbol{e}$ is a vector of ones.

Both Problems (3.4) and (3.5) find a hyperplane $\boldsymbol{p}'_{k,r}\boldsymbol{x} = q_{k,r}$ that softly separates

points in Group k from points in Group r, i.e., points in either group can be on the wrong side of this hyperplane if necessary. The purpose of the third constraint is to prevent getting the trivial hyperplane $\boldsymbol{p}_{k,r} = \mathbf{0}$ and $q_{k,r} = 0$ for the optimal solution. Problem (3.4) sets the sum of the elements of $\boldsymbol{p}_{k,r}$ to be strictly positive and Problem (3.5) sets the sum of the elements of $\boldsymbol{p}_{k,r}$ to be strictly negative. Both of these problems need to be solved since we do not know *a priori* whether the sum of the elements of the optimal non-trivial $\boldsymbol{p}_{k,r}$ is positive or negative [2]. The solution of the problems that results in the least number of violated points is chosen as our hyperplane.

After we solve Problems (3.4) and (3.5) for every pair of groups, we let

$$P_k = \{\boldsymbol{x} |\ \boldsymbol{p}'_{k,i}\boldsymbol{x} \leq q_{k,i}, i = 1, \ldots, k-1, \quad \boldsymbol{p}'_{k,i}\boldsymbol{x} \geq q_{k,i}, i = k+1, \ldots, K\}. \tag{3.6}$$

After P_k is defined, we recompute $\boldsymbol{\beta}_k$ using all the points contained in P_k since it is possible that they are different from the original G_k that Problem (3.3) found. We solve a linear optimization problem that minimizes the absolute deviation of all points in P_k to find the new $\boldsymbol{\beta}_k$.

3.6 Nonlinear Data Transformations

In order to improve the predictive power of CRIO, we augment the explanatory variables with nonlinear transformations. In particular, we consider the transformations x^2, $\log x$ and $1/x$ applied to the coordinates of the given points. We can augment each d-dimensional vector $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,d})'$ with $x_{i,j}^2$, $\log x_{i,j}$, $1/x_{i,j}$, $j = 1, \ldots, d$, and apply CRIO to the resulting $4d$-dimensional vectors, but the increased dimension slows down the computation time. For this reason, we use a simple heuristic method to choose which transformation of which variable to include in the data set.

For $j = 1, \ldots, d$, we run the one dimensional regressions: (a) $(x_{i,j}, y_i)$, $i \in N$, with

[2] If the optimal non-trivial $\boldsymbol{p}_{k,r}$ is orthogonal to $\boldsymbol{e}$, then we would not be able to find that vector with either formulation. However, we will obtain a hyperplane that is slightly perturbed from the optimal, which suffice for our purposes

sum of squared errors equal to $f_{j,1}$, (b) $(x_{i,j}^2, y_i)$, $i \in N$, with sum of squared errors equal to $f_{j,2}$, (c) $(\log x_{i,j}, y_i)$, $i \in N$, with sum of squared errors equal to $f_{j,3}$, (d) $(1/x_{i,j}, y_i)$, $i \in N$, with sum of squared errors equal to $f_{j,4}$. If $f_{j,2} < f_{j,1}$, we add $x_{i,j}^2$ and eliminate $x_{i,j}$. If $f_{j,3} < f_{j,1}$, we add $\log x_{i,j}$ and eliminate $x_{i,j}$. If $f_{j,4} < f_{j,1}$, we add $1/x_{i,j}$ and eliminate $x_{i,j}$. Otherwise, we do not add any nonlinear transformation to the data set.

3.7 The Overall Algorithm for Regression

The overall algorithm for regression is as follows:

1. **Nonlinear transformation**: Augment the original data set with nonlinear transformations using the method discussed in Section 3.6.

2. **Preprocessing**: Use the clustering algorithm to find $L \ll n$ clusters of the data points.

3. **Assign Clusters to Groups**: Solve Problem (3.3) to determine which points belong to which group, while eliminating potential outliers.

4. **Assign Groups to Polyhedral Regions**: Solve the linear optimization problems (3.4) and (3.5) for all pairs of groups, and define polyhedra as in Eq. (3.6).

5. **Re-computation of $\boldsymbol{\beta}$'s:** Once the polyhedra P_k are indentified, recompute $\boldsymbol{\beta}_k$ using only the points that belong in P_k.

Given a new point $\boldsymbol{x}_0$ (augmented by the same transformations as applied in the training set data), if $\boldsymbol{x}_0 \in P_k$, then we predict $\hat{y}_0 = \boldsymbol{\beta}_k' \boldsymbol{x}_0$. Otherwise, we assign $\boldsymbol{x}_0$ to the region P_r that contains the majority among its F nearest neighbors in the training set, and make the prediction $\hat{y}_0 = \boldsymbol{\beta}_r' \boldsymbol{x}_0$.

3.8 Computational Results

We tested the regression component of CRIO on three real data sets found on the UCI data repository and compared the results to commercial softwares of CART [9] and MARS [17][3]. Similar to the experiments on the classification data sets in Section 2.9, each of the regression data was split into three parts, with 50%, 30% and 20% of the data used for training, validating and testing, respectively.

We used the validation set for CRIO to fine tune the values of parameters K, L and ρ of Problem (3.3). In CART, we use the validation set in the pruning step to prevent over-fitting of the initial tree. In MARS, we use the validation set to choose the appropriate maximum number of basis functions. Also, we adjusted the running time versus accuracy parameter in MARS to maximize accuracy over running time.

Regression Data

The "Boston" data, with 13 explanatory variables and 506 observations, is the Boston housing data set with dependent variable being the median value of houses in the suburbs of Boston. The "Abalone" data, with 8 explanatory variables and 4177 observations, attempts to predict the age of an abalone given its physiological measurements. Finally, the "Auto" data, with 5 explanatory variables and 392 observations, is the auto-mpg data set which determines the miles-per-gallon fuel consumption of an automobile, given the mechanical attributes of the car.

Results

Table 3.1 illustrate the results of CART, MARS and CRIO for each data set. In addition, we ran MARS on the transformed data used in CRIO, creating MARS-transf. MAE is the mean absolute error and MSE is the mean squared error.

CRIO chose $K = 2$ for all data sets, $L = 3$ and 4 for "Boston" and "Auto" and $L = 10 - 14$ for "Abalone", and $\alpha = 1$ and 2% for all data sets. We commented earlier that it is possible for a new point not to be contained in any of the polyhedra P_k. We

[3]MARS is a product of Salford Systems, http://www.salford-systems.com

Boston	**Train**		**Validation**		**Test**	
$n = 506, d = 13$	MAE	MSE	MAE	MSE	MAE	MSE
CART	1.883	8.121	2.840	17.746	2.984	21.057
MARS	2.346	10.363	2.536	11.750	2.600	15.810
MARS-transf	2.283	9.907	2.463	11.391	2.543	14.543
CRIO	2.171	11.508	2.554	13.882	2.337	14.197

Abalone	**Train**		**Validation**		**Test**	
$n = 4177, d = 8$	MAE	MSE	MAE	MSE	MAE	MSE
CART	1.387	4.676	1.577	5.655	1.628	5.978
MARS	1.492	4.263	1.563	6.518	1.552	6.127
MARS-transf	1.507	4.373	1.552	4.690	1.540	4.525
CRIO	1.506	4.594	1.515	4.880	1.469	4.376

Auto-mpg	**Train**		**Validation**		**Test**	
$n = 392, d = 5$	MAE	MSE	MAE	MSE	MAE	MSE
CART	1.397	4.898	2.444	11.350	2.435	12.784
MARS	1.898	6.282	2.176	9.098	2.127	8.433
MARS-transf	2.084	7.386	1.884	7.029	1.922	8.101
CRIO	1.765	6.936	2.133	9.200	2.051	7.851

Table 3.1: Results of models CART, MARS, MARS-transf and CRIO on "Boston", "Abalone" and "Auto" data sets. n is the number of data points, and d is the number of explanatory variables.

found this to be an extremely rare case for all the data sets, but when it did occur, we assigned the point to the region that contains the majority of $F = 11$ nearest neighbors. The nonlinear transformation step did improve the model, sometimes significantly, and log was the most commonly used transformation for each data set. The run times of CRIO for "Boston" and "Auto" is comparable to CART and MARS, taking couple CPU seconds on average. CRIO takes up to 13 minutes to solve for "Abalone", which is a significantly larger data set.

CRIO clearly outperformed both CART and MARS in the testing set accuracy. To our surprise, it had better MSE values although our model minimized the total absolute error, whereas CART's and MARS' goodness-of-fit criterion is proportional to the squared error. Compared to CART, CRIO had up to 21.7% lower mean absolute error and 38.6% lower mean squared error. Although MARS was more accurate than CART on average, CRIO still had up to 10.1% lower mean absolute error and 28.6% lower mean squared error.

We used MARS-transf to assess whether the reason of CRIO's superior performance was the nonlinear transformation of the variables or the mixed-integer optimization methodology. It is clear that MARS-transf improves upon MARS, but it is also clear that CRIO's performance is not only due to the nonlinear transformation. CRIO still has up to 8.1% lower mean absolute error and 3.6% lower mean squared error compared to MARS-transf.

Table 3.2 illustrates the standard deviations of the prediction for CART, MARS and CRIO on all three data sets. The number of data points and partitions are too small to make accurate measures of error and confidence intervals, but it helps to give a rough idea on the stability of each model across different training set partitions. It does not seem as though one method dominates over others, except in the "Boston" data set. However, this is mainly due to the measurement error in the data set. The response variable for "Boston" is cut off at 50 – i.e., even if the actual value is greater than 50, it was recorded as 50. This contributed to large errors for both CART and CRIO, but MARS successfully handles these clipped data points. Thus, the existence of these clipped data points in the training set greatly affects the prediction results

Boston	**Train**		**Validation**		**Test**	
	MAE	MSE	MAE	MSE	MAE	MSE
CART	0.147	0.598	0.392	7.374	0.244	6.993
MARS	0.159	1.793	0.072	0.832	0.280	2.749
MARS-transf	0.144	1.324	0.122	1.110	0.250	2.400
CRIO	0.066	1.295	0.088	2.958	0.321	6.798
Abalone	**Train**		**Validation**		**Test**	
	MAE	MSE	MAE	MSE	MAE	MSE
CART	0.042	0.317	0.030	0.358	0.060	0.392
MARS	0.069	0.355	0.045	2.987	0.125	3.358
MARS-transf	0.059	0.273	0.040	0.219	0.110	0.654
CRIO	0.054	0.295	0.035	0.130	0.085	0.454
Auto-mpg	**Train**		**Validation**		**Test**	
	MAE	MSE	MAE	MSE	MAE	MSE
CART	0.397	2.132	0.175	1.959	0.110	1.362
MARS	0.092	0.328	0.099	0.793	0.242	1.959
MARS-transf	0.151	0.835	0.255	2.091	0.157	0.966
CRIO	0.057	0.649	0.191	1.226	0.112	0.847

Table 3.2: Standard deviation of the prediction of CART, MARS, MARS-transf and CRIO on "Boston", "Abalone" and "Auto" data sets.

for CART and CRIO, but less so for MARS.

3.9 Conclusion

As in classification, the regression component of CRIO is a new approach to solve regression problems. It combines the use of nonlinear transformation of the explanatory variables, single-linkage clustering in the combined x and y space to speed computation as well as detect outliers, and mixed-integer optimization to partition the explanatory variable space. Unlike, popular regression packages of CART and MARS, CRIO partitions the explanatory variable space in a globally optimal manner so that the total absolute error is minimized. We believe that this globally optimal partition provided by the mixed-integer optimization model, combined with transformations and clusterings , is the key reason for CRIO's success. Most notably, it had up to 38% and 28% lower mean squared error in the the testing set compared to CART and MARS, respectively.

Another significant advantage of using integer optimization in regression is its ability to remove outliers a priori and during the model creation step. In traditional regression, data points are often labelled as outliers by observing high residuals after the regression model is fit. However, the outlier points have already heavily influenced or leveraged the regression line. Also, unlike popular outlier elimination methods that can only detect one outlier point at a time, the clustering heuristic combined with the mixed integer optimization model is able to remove multiple outlier points at once. The traditional regression methods, that observes the diagonal element of the "hat" matrix, cannot capture outliers that come in clusters [39].

These preliminary results encourage us to explore further extensions to our regression model. As mentioned in the previous chapter and further discussed in Chapter 5, subset selection is a key problem in regression. We would like to incorporate the variable selection techniques of Chapter 5 to CRIO, which currently uses all of the explanatory variables. Also, our current model does not guarantee continuity of our prediction function at the borders of the K regions. The continuity condition can be incorporated in the optimization model, but will make the solution time for the integer optimization problem significantly longer. Since continuity may improve the prediction stability of the algorithm, it is an extension that warrants further investigation.

Chapter 4

Quadratic Variable Selection via Integer Optimization

In this chapter, we discuss a new method for solving quadratic variable selection problems, which we define as quadratic optimization problems that limit the number of variables one can use in the solution, and its direct application to subset selection in regression and portfolio selection in asset allocation. We take advantage of the special structures of these problems by implementing a combination of implicit branch-and-bound, Lemke's pivoting method, variable deletion and problem reformulation. Unlike the previous two chapters where we introduced new models for solving data mining problems, this chapter presents are new solution methodology for solving data mining problems with this particular structure.

A quadratic variable selection problem has the following form:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\boldsymbol{x}'\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}'\boldsymbol{x}, \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ & |supp(\boldsymbol{x})| \leq K, \\ & x_i \geq \alpha_i, \qquad i \in supp(\boldsymbol{x}), \\ & 0 \leq x_i \leq u_i, \qquad \forall i. \end{aligned} \tag{4.1}$$

where $\boldsymbol{Q} \in \mathbb{R}^{d\times d}$ is symmetric positive semi-definite, $\boldsymbol{c} \in \mathbb{R}^d$, $\boldsymbol{A} \in \mathbb{R}^{m\times d}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\alpha_i \in (0,1]$, u_i is the nonnegative upper bound of x_i, K is some positive integer and $supp(\boldsymbol{x}) = \{i|x_i \neq 0\}$. The second set of constraints, referred to as the cardinality constraint, and the third set of constraints, referred to as the lower bound constraints, introduce discreteness to the problem, making this a quadratic mixed-integer optimization problem. Unlike linear integer optimization, quadratic mixed-integer optimization problems have received little attention due to the difficulty of re-optimizing the relaxed quadratic subproblems in a branch-and-bound setting. Our approach attempts to overcome this computational problem.

It is possible to introduce binary inclusion variables, z_i (where $z_i = 1$ if and only if $x_i > 0$, and $z_i = 0$ if and only if $x_i = 0$), and express the cardinality constraint as $\sum_i z_i \leq K$ and the lower bound constraint as $x_i \geq \alpha_i z_i$. However, this doubles the number of variables and makes the continuous relaxation significantly harder to solve. In Bienstock's implicit branch-and-bound algorithm algorithm [6], the cardinality constraint is replaced by a surrogate constraint $\sum_i (x_i/u_i) \leq K$, and the $\boldsymbol{x}$ variables are branched on directly – i.e., when branching down on x_j, x_j is set to 0, and when branching up on x_j, we ensure $x_j \geq \alpha_i$. However, [6] still solves the quadratic relaxation using gradient methods that have difficulty starting from an infeasible point. Also, it only considers the case where $\alpha_i = 0$ for all i. McBride and Yormark [30] extends the implicit enumeration concept for linear pure integer optimization [19] to solves a pure 0-1 quadratic optimization problem. [30] uses Lemke's pivoting algorithm that enables them to restrict variables to 0 or 1 without adding constraints.

We present a general method to solve Problem (4.1) that combines the use of implicit branch-and-bound, Lemke's method [27, 11], variable deletion and simple reformulation techniques to overcome the computational obstacles. Section 4.1 elaborates on this general methodology for solving cardinality constrained quadratic mixed-integer optimization problems. We further tailor our method to solve two important problems in Statistics and Finance: subset selection in regression and portfolio selection in finance.

Subset Selection in Regression

In traditional multivariate regression, we are given n data points $(\boldsymbol{x}_i, y_i)$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, and we want to find $\boldsymbol{\beta} \in \mathbb{R}^d$ such that $\sum_i (y_i - \boldsymbol{x}_i'\boldsymbol{\beta})^2$ is minimized. This has a closed-form solution, $\boldsymbol{\beta} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Y}$, where $\boldsymbol{X}$ is a n by d matrix with $\boldsymbol{x}_i$ as its ith row, and $\boldsymbol{Y}$ is a n-dimensional vector with y_i as its ith element.

Often, however, statisticians only want to use a small subset of the variables, mainly for robustness purposes (i.e., to limit the variace of the predicted Y) and perhaps due to the high cost of obtaining data for large numbers of variables [31, 40, 1]. Beale et al [2], Hockings and Leslie [21] and Furnival and Wilson [18] are works that use pivoting and branch-and-bound techniques to search for subsets with the best regression "fit" (e.g., minimum total sum of squared errors) for subsets of all sizes. Narula and Wellington [32] solves linear mixed-integer optimization problems that finds a subset of K variables ($K < n$) that has the minimum total absolute error.

The above exact enumeration methods, however, have been deemed impractical for large problems (e.g., $n > 30$) to be solved in practical time [40]. Currently, simple greedy heuristics, such as forward, backwards and stepwise regression, are the most commonly used methods to choose which variables to include and exclude [31, 40].

We solve the problem of choosing a subset of K variables that minimizes the total sum of squared errors to optimality, by modelling it as a unconstrained, free-variable version of Problem (4.1). We will show that our implicit branch-and-bound procedures, together with efficient computation of the optimal objective value at each node, finds solutions that significantly out-performs forward regression, in terms of total sum of squares, within few seconds. Section 4.2.1 elaborates on our method, and Section 4.3.1 illustrates the performance of our approach compared to forward regression heuristic and an explicit branch-and-bound procedure.

Optimal Portfolio Selection Problem

The problem of constructing a portfolio of n stocks that minimizes the variance, while ensuring that the expected rate of returns is greater than a given threshold, can be formulated as a convex quadratic optimization problem. Often, however, transaction

costs and other overhead expenses lead small investors to invest in only a small number of the total possible stocks. Such limited diversification constraints, along with fixed transaction costs and minimum transaction levels, present discrete constraints and variables to the quadratic problem (see for example Bertsimas et al. [4]).

Given the difficulty of solving quadratic integer optimization problems, such a portfolio problem have commonly been approached in one of two ways. The first approach is approximating the problem to a simpler form. For example, Sharpe [41, 42] approximates the quadratic expected variance function by a linear and piecewise linear function. Jacob [22] similarly linearizes the quadratic objective function, and further assumes equal weights across assets to formulate the problem as a pure 0-1 problem. Patel and Subrahmanyam [35] simplify the covariance matrix of the rates of return and shows how the resulting portfolio problem with fixed-transaction costs can be solved efficiently. The second approach uses heuristics to find strong feasible solutions. Mansini and Speranza [29] solves a linear mixed-integer optimization based heuristic, and Blog et al [7] introduces a dynamic programming based heuristic that often finds the optimal solution. Chang et al [10] illustrates genetic algorithm, tabu search and simulated annealing approaches for the problem.

There has been, however, significant efforts in finding exact algorithms to efficiently solve the discrete portfolio problem. Owens-Butera [34] extends the implicit branch-and-bound techniques of Bienstock to limited diversification portfolios. Jobst et al [23] implements FortQP as the quadratic solver, and solves portfolio problems with minimum transaction levels, limited diversification and roundlot constraints (which requires investing in discrete units) in a branch-and-bound context. FortQP uses an approach similar to the simplex method to solve the quadratic problem.

We propose to solve portfolio problems with limited diversification, fixed-transaction costs and minimum transaction levels via implicit branch-and-bound, Lemke's pivoting method, variable deletion and reformulation. We elaborate on our methodology in Section 4.2.2 and illustrate computational results in Section 4.3.2.

4.1 General Methodology

In a branch-and-bound setting, we solve the continuous relaxation of Problem (4.1) via Lemke's method, then choose a branching variable x_s. When branching down, we update the subsequent subproblem by deleting the data associated to x_s, and when branching up, we reformulate the problem or modify Lemke's method so that $x_s \geq \alpha_s$.

The continuous relaxation we solve at each node has the following form:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\boldsymbol{x}'\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}'\boldsymbol{x}, \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ & \boldsymbol{x} \geq 0, \end{aligned} \tag{4.2}$$

where the cardinality constraint and lower bound constraints are removed. We do not replace the cardinality constraint with Bienstock's surrogate constraint [6], $\sum_i (x_i/u_i) \leq K$, because u_i is unknown in the subset selection problem and the constraint is dominated by a constraint in $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ in the portfolio selection problem. The lower bound constraint, $x_i \geq \alpha_i$ for x_i positive, are enforced either by the reformulation procedure in the branching up step or implicitly enforced by Lemke's method. Section 4.1.1 describes the use of Lemke's method to solve Problem (4.2) in the context of branch-and-bound. Section 4.1.2 illustrates the procedure for updating the subproblem after the branching variable is deleted, and Section 4.1.3 describes our reformulation procedure and Lemke's method with lower bounds to enforce the lower bound constraint without adding new constraints or changing the structure of the problem.

4.1.1 Lemke's Method as Underlying Quadratic Optimizer

We use Lemke's pivoting method to optimize the continuous relaxation of the subproblem at each node. This method was originally developed to solve linear complementarity problems (of which quadratic programs are a special case) via pivoting akin to the simplex method. As with the dual simplex method in linear optimization,

the key advantage of Lemke's method is its ease and efficiency of starting from an infeasible point. This is critical in the branch-and-bound setting, since the optimal solution of the parent node can be used as the initial point to solve the problem of the current node. Thus, this solver has a significant advantage over interior point methods which cannot start from an infeasible point and thus must solve from scratch at each node.

First, we will introduce linear complementarity problems and their extensions to quadratic optimization. Then we present an overview of the Lemke's method to solve LCPs. Finally, we describe the implementation of Lemke's method in the context of branch-and-bound.

The Linear Complementarity Problem

A linear complementarity problem (LCP) is the following: Given $\boldsymbol{q} \in \mathbb{R}^n$ and $\boldsymbol{M} \in \mathbb{R}^{n \times n}$, find $\boldsymbol{z} \in \mathbb{R}^n$ and $\boldsymbol{w} \in \mathbb{R}^n$ such that,

$$\boldsymbol{w} = \boldsymbol{M}\boldsymbol{z} + \boldsymbol{q}, \tag{4.3}$$

$$\boldsymbol{z} \geq \boldsymbol{0}, \boldsymbol{w} \geq \boldsymbol{0}, \tag{4.4}$$

$$\boldsymbol{z}'\boldsymbol{w} = 0. \tag{4.5}$$

The above problem is referred to as LCP$(\boldsymbol{q}, \boldsymbol{M})$. A vector $\boldsymbol{z}$ is called *feasible* if it satisfies Equations (4.3) and (4.4), and *complementary* if it satisfies (4.5). A vector $\boldsymbol{z}$ that is both feasible and complementary is called an *equilibrium point* or a *solution* of LCP$(\boldsymbol{q}, \boldsymbol{M})$. Note that due to the nonnegativity of the variables, $\boldsymbol{z}$ is complementary if and only if

$$z_i w_i = 0 \qquad \forall i = 1, \ldots, n. \tag{4.6}$$

The pair z_i and w_i are called *complementary pairs.*

Now, suppose we have the following quadratic optimization problem:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\boldsymbol{x}'\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}'\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned} \tag{4.7}$$

where $\boldsymbol{Q} \in \mathbb{R}^{d\times d}$, $\boldsymbol{A} \in \mathbb{R}^{m\times d}$ and $\boldsymbol{b} \in \mathbb{R}^m$ are given. For positive semi-definite matrix $\boldsymbol{Q}$, the KKT conditions for (4.7), which are necessary and sufficient for finding global minima, are as follows:

$$\begin{aligned} \boldsymbol{c} + \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{A}'\boldsymbol{y} - \boldsymbol{u} &= 0, \\ \boldsymbol{y}'(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) &= 0, \\ \boldsymbol{u}'\boldsymbol{x} &= 0, \\ \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u} &\geq \boldsymbol{0}. \end{aligned} \tag{4.8}$$

where $\boldsymbol{y} \in \mathbb{R}^m$ and $\boldsymbol{u} \in \mathbb{R}^d$ are the dual variables. By introducing another variable $\boldsymbol{v} \in \mathbb{R}^m$, (4.8) can be written as:

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{c} + \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{A}'\boldsymbol{y}, \\ \boldsymbol{v} &= \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}, \\ \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v} &\geq \boldsymbol{0}, \\ \boldsymbol{x}'\boldsymbol{u} = 0, \boldsymbol{y}'\boldsymbol{v} &= 0. \end{aligned} \tag{4.9}$$

The system in (4.9) can be represented as an LCP$(\boldsymbol{q}, \boldsymbol{M})$ with

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{b} \end{bmatrix}, \boldsymbol{M} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}' \\ -\boldsymbol{A} & 0 \end{bmatrix}, \tag{4.10}$$

and $\boldsymbol{z}$ and $\boldsymbol{w}$ of (4.3)-(4.5) as

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix}.$$

In order to speak of bases and basic variables, let us first re-write (4.3) as

$$\boldsymbol{I_n w} - \boldsymbol{M z} = \boldsymbol{q}, \qquad \text{or} \qquad \begin{bmatrix} \boldsymbol{I_n}, -\boldsymbol{M} \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{z} \end{bmatrix} = \boldsymbol{q}.$$

where $\boldsymbol{I_n}$ is an n-dimensional identity matrix. We can find a feasible solution to Equations (4.3)-(4.4), by selecting n linearly independent columns of $\begin{bmatrix} \boldsymbol{I_n}, -\boldsymbol{M} \end{bmatrix}$, forming the basis $\boldsymbol{B}$, such that $\boldsymbol{B}^{-1}\boldsymbol{q} \geq \boldsymbol{0}$. To further get a complementary solution to Equations (4.3)-(4.5), the basis contains z_i if and only if w_i is not in the basis, and vice versa. Let β be the set of indices of $\boldsymbol{z}$ that are in the basis. Then the associated *complementary basis*, $\boldsymbol{B}$, is such that:

$$\boldsymbol{B}_{.i} = \begin{cases} -\boldsymbol{M}_{.i} & \text{if} \quad i \in \beta \\ \boldsymbol{e}_i & \text{if} \quad i \notin \beta \end{cases} \tag{4.11}$$

where $\boldsymbol{B}_{.i}$ is the ith column of $\boldsymbol{B}$ and $\boldsymbol{e}_i$ is an unit vector with the ith element equal to 1. The goal of Lemke's method is to find such a complementary basis, $\boldsymbol{B}$, such that $\boldsymbol{B}^{-1}\boldsymbol{q} \geq 0$. The following section elaborates on this methodology.

Lemke's Method

Given $\boldsymbol{q}$ and $\boldsymbol{M}$ and some user-defined *covering vector* $\boldsymbol{h} \in \mathbb{R}^n$, where $\boldsymbol{h} > \boldsymbol{0}$, Lemke's method solves LCP$(\boldsymbol{q}, \boldsymbol{M})$ via series of pivot operations. First, it checks whether $\boldsymbol{q} \geq 0$, in which case $\boldsymbol{z} = 0$ and $\boldsymbol{w} = \boldsymbol{q}$ is a solution. Otherwise, it augments

LCP$(\boldsymbol{q}, \boldsymbol{M})$ to

$$\boldsymbol{w} = \boldsymbol{q} + \boldsymbol{h}z_0 + \boldsymbol{M}\boldsymbol{z} \geq \boldsymbol{0}, \quad z_0 \geq 0, \quad \boldsymbol{z} \geq \boldsymbol{0}, \quad \boldsymbol{z}'\boldsymbol{w} = 0. \tag{4.12}$$

Akin to the dual simplex method, we need to start the algorithm with a complementary basis that does not necessarily satisfy the nonnegativity constraint (4.4). A simple default basis is to have all the $\boldsymbol{z}$ variables be nonbasic and $\boldsymbol{w}$ be basic. We then set the auxiliary variable z_0 to the smallest positive value such that $\boldsymbol{w} \geq \boldsymbol{0}$ when $\boldsymbol{z} = \boldsymbol{0}$, i.e., $z_0 = \max_i(-q_i/h_i), i = 1, \ldots, d$. Thus, $z_i w_i = 0, i = 1, \ldots, n$ and $z_0 > 0$, and z_0 is pivoted into the basis in place of w_r, where $r = \text{argmax}(-q_i/h_i)$. Such a point is called an *almost complementary* point for the augmented problem (4.12). The algorithm follows a path from one almost complementary basic solution to the next [26], until z_0 is pivoted out to be a nonbasic variable or LCP$(\boldsymbol{q}, \boldsymbol{M})$ is shown to be infeasible.

After the initial pivot, there are exactly one complementary pair that are both nonbasic variables (for all other pairs, one of the complement is basic and the other is nonbasic). Such a pair is called the *nonbasic pair*. In each iteration, one of the nonbasic pair just became nonbasic via the previous pivot. Its complement can thus be increased to a positive value and maintain the almost complementary condition. This complement, called the *driving variable*, is increased until one of the basic variables becomes zero. This basic variable that blocks further increase of the driving variable is called the *blocking variable*. The driving variable enters into the basis in place of the blocking variable which becomes nonbasic. This process is repeated until either

1. z_0 becomes the blocking variable, or

2. We fail to find a blocking variable.

In the first case, z_0 is pivoted out of the basis and set to 0. Thus, we have a basis that is both feasible and complementary for the original problem. If we encounter the second case, we can show that the original linear complementarity problem and thus the quadratic optimization problem (4.7) is infeasible.

The user-defined covering vector, $\boldsymbol{h} \in \mathbb{R}^n$, is set to the default value of $h_i = 1, \quad i = 1, \ldots, n$. Although, $\boldsymbol{h}_i$ need only to be all strictly positive, it has been shown that different choices of $\boldsymbol{h}$ can greatly effect the number of pivots for some instances of the problem. Smart selection of the covering vector is a problem on its own and needs further investigation. Assuming that the Problem (4.12) is nondegenerate (an assumption that can be relaxed with certain lexicographic pivoting rules discussed in section 4.9 of [11]), Lemke's method is a finite algorithm, since there are only finite number of complementary bases. Also, given that the matrix $\boldsymbol{Q}$ is positive semi-definite, the value of z_0 monotonically decreases at each pivot. Appendix A elaborates on these and other properties of the Lemke's method.

Lemke's Method in the Context of Branch-and-Bound

As in linear mixed-integer optimization, we want to be able to resolve a new subproblem starting with the basic solution resulting from the parent subproblem. In the linear case, the dual simplex method can easily start from an infeasible basis, and thus is key to solving each subproblem efficiently. The same is true for Lemke's method, which can start from an infeasible complementary basis. The following describes this resolve procedure.

Suppose $\hat{\boldsymbol{Q}}$, $\hat{\boldsymbol{A}}$, $\hat{\boldsymbol{c}}$ and $\hat{\boldsymbol{b}}$ are the input data for the current subproblem (4.2), and $\hat{\boldsymbol{M}}$ and $\hat{\boldsymbol{q}}$ are constructed according to (4.10). At an intermediary pivot step, with $\boldsymbol{B}$ as the complementary basis and $\boldsymbol{N}$ as the nonbasic columns of $\hat{\boldsymbol{M}}$, the full tableau would look as follows:

$$\begin{bmatrix} \boldsymbol{q}_B & | & \boldsymbol{I} & -\boldsymbol{M}_B \end{bmatrix},$$

where $\boldsymbol{q}_B = \boldsymbol{B}^{-1}\hat{\boldsymbol{q}}$ and $\boldsymbol{M}_B = -\boldsymbol{B}^{-1}\boldsymbol{N}$. This tableau is pivoted until the auxiliary variable z_0 is pivoted out of the basis. As in the revised simplex method, we maintain $\boldsymbol{B}^{-1}$, $\boldsymbol{M}_B$ and $\boldsymbol{q}_B$ at each pivot iteration. Suppose Lemke's method terminates with $\boldsymbol{B}^*$, $\boldsymbol{N}^*$ and thus $\boldsymbol{M}_B^* = -\boldsymbol{B}^{*-1}\boldsymbol{N}^*$ and $\boldsymbol{q}_B^* = \boldsymbol{B}^{*-1}\boldsymbol{q}$. The solution to this subproblem is thus $x_i = q^*_{B(i)}$ if x_i is the $B(i)^{th}$ basic variable, and $x_i = 0$ if x_i is nonbasic.

When constructing the subsequent subproblem after branching, all or some of the data (namely, $\hat{\boldsymbol{Q}}$, $\hat{\boldsymbol{A}}$, $\hat{\boldsymbol{c}}$ and or $\hat{\boldsymbol{b}}$) may be modified. Let $\overline{\boldsymbol{M}}$ and $\overline{\boldsymbol{q}}$ correspond to $\hat{\boldsymbol{M}}$ and $\hat{\boldsymbol{q}}$, respectively, using the modified data. Also, let $\overline{\boldsymbol{B}}$ and $\overline{\boldsymbol{N}}$ be the basis and the nonbasic columns, respectively, after the modification. To solve the next subproblem, we want Lemke's method to start with $\overline{\boldsymbol{B}}$ as its initial complementary basis, instead of its default basis. Empirical testing shows that starting from the previous basis dramatically reduces the total number of pivots required to pivot out the auxiliary variable z_0. In this case, Lemke's initializes with $\boldsymbol{M}_{\bar{\boldsymbol{B}}} = -\overline{\boldsymbol{B}}^{-1}\overline{\boldsymbol{N}}$ and $\boldsymbol{q}_{\bar{\boldsymbol{B}}} = \overline{\boldsymbol{B}}^{-1}\overline{\boldsymbol{q}}$. Since recomputing the inverse basis and the matrix multiplication to compute $\boldsymbol{M}_{\bar{\boldsymbol{B}}}$ is expensive, Section 4.1.2 and 4.1.3 describes methods to update the new inputs efficiently after branching down and up, respectively.

4.1.2 Branching Down

When branching down on x_s, we delete all the data associated to x_s for the subsequent subproblems and update the basis accordingly. We chose to delete the variable instead of explicitly adding the constraint $x_s = 0$ to prevent increasing the size of the subproblem as well as for numerical stability purposes. We will show that in most cases, the inverse of the new basis can be efficiently derived from the inverse of the old basis by using elementary row operations.

Let us assume that x_s is a basic variable in the previous solution (note that if x_s is a nonbasic variable, we can apply the following methodology for its complementary variable, w_s, which must be a basic variable). We delete the column and row of $\boldsymbol{B}$ corresponding to $\boldsymbol{x}_s$ and the column and row of $\boldsymbol{N}$ corresponding to its dual variable $\boldsymbol{w}_s$, where $\boldsymbol{B}$ is the basis and $\boldsymbol{N}$ are nonbasic columns of the solution to the subproblem of the parent node (i.e., if x_s is the ith basic variables, then we delete the ith column and s^{th} row of $\boldsymbol{B}$. Similarly for $\boldsymbol{w}_s$ and $\boldsymbol{N}$). Although we can get the new inverse of the basis simply by inverting the modified basis, calculating the inverse can be a significant bottleneck. Instead, we calculate the new inverse from the previous inverse using elementary row operations.

Suppose, for notational purposes, that the column and row needed to be deleted

in $\boldsymbol{B}$ are the last column and first row, and $\boldsymbol{B} \in \mathbb{R}^{n \times n}$, so that:

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B_{row}}' & v \\ \overline{\boldsymbol{B}} & \boldsymbol{B_{col}} \end{bmatrix}, \qquad \boldsymbol{B^{-1}} = \begin{bmatrix} \boldsymbol{U_{row}}' & u \\ \overline{\boldsymbol{U}} & \boldsymbol{U_{col}} \end{bmatrix},$$

where $\overline{\boldsymbol{B}}$ and $\overline{\boldsymbol{U}}$ are $n-1$ by $n-1$ upper-left submatrices of $\boldsymbol{B}$ and $\boldsymbol{B^{-1}}$, respectively, $\boldsymbol{B_{col}}$, $\boldsymbol{B_{row}}$, $\boldsymbol{U_{col}}$ and $\boldsymbol{U_{row}}$ are $(n-1)$-dimensional column vectors, and v and u are scalars. We know that:

$$\boldsymbol{B^{-1}B} = \begin{bmatrix} \boldsymbol{U_{row}}'\overline{\boldsymbol{B}} + u\boldsymbol{B_{row}}' & \boldsymbol{U_{row}}'\boldsymbol{B_{col}} + uv \\ \overline{\boldsymbol{UB}} + \boldsymbol{U_{col}}\boldsymbol{B_{row}}' & \overline{\boldsymbol{U}}\boldsymbol{B_{col}} + v\boldsymbol{U_{col}} \end{bmatrix} = \boldsymbol{I}_n.$$

Thus we get

$$\overline{\boldsymbol{UB}} = \boldsymbol{I_{n-1}} - \boldsymbol{U_{col}}\boldsymbol{B_{row}}'. \tag{4.13}$$

Since $\boldsymbol{U_{col}}\boldsymbol{B_{row}}'$ is a rank one matrix, we can execute linear number of elementary row operations to the matrix $\boldsymbol{I}_{n-1} - \boldsymbol{U_{col}}\boldsymbol{B_{row}}'$ to get $\boldsymbol{I}_{n-1}$. Let $\boldsymbol{E}$ be the matrix representing those operations. Then if $\overline{\boldsymbol{B}}$ is invertible, then $\boldsymbol{E}\overline{\boldsymbol{U}}$ is the inverse matrix of $\overline{\boldsymbol{B}}$.

In the previous section, we stated that we use $\boldsymbol{M}_{\bar{B}} = -\overline{\boldsymbol{B}}^{-1}\overline{\boldsymbol{N}}$ as input to Lemke's. We avoid this matrix multiplication via similar elementary row operations. Suppose $\boldsymbol{N}$ are the nonbasic columns and $\boldsymbol{M}_B = -\boldsymbol{B}^{-1}\boldsymbol{N}$ at termination of Lemke's at the parent node. Again, let us assume that the column corresponding to w_s in $\boldsymbol{N}$ is the last one. Then,

$$\boldsymbol{M}_B = -\boldsymbol{B}^{-1}\boldsymbol{N} = -\begin{bmatrix} \boldsymbol{U_{row}}' & u \\ \overline{\boldsymbol{U}} & \boldsymbol{U_{col}} \end{bmatrix} \begin{bmatrix} \boldsymbol{N_{row}}' & p \\ \overline{\boldsymbol{N}} & \boldsymbol{N_{col}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M_{row}}' & w \\ \overline{\boldsymbol{M}} & \boldsymbol{M_{col}} \end{bmatrix},$$

where $\overline{\boldsymbol{N}}$ and $\overline{\boldsymbol{M}}$ are $n-1$ by $n-1$ upper-left submatrices of $\boldsymbol{N}$ and $\boldsymbol{M}_B$, respectively, and $\boldsymbol{N_{col}}$, $\boldsymbol{N_{row}}$, $\boldsymbol{M_{col}}$ and $\boldsymbol{M_{row}}$ are $(n-1)$-dimensional column vectors, and p and

w are scalars. Again, we know that:

$$-\overline{U}\overline{N} = \overline{M} + U_{col}N_{row}'.$$

Since $E\overline{U} = \overline{B}^{-1}$, the new $M_{\bar{B}}$ matrix will be

$$M_{\bar{B}} = E(\overline{M} + U_{col}N_{row}'). \tag{4.14}$$

There are several cases that need to be checked before executing the above procedures. Most critically, if $\overline{B}$ is singular, then E may be undefined. In such a case, we start Lemke's method from scratch with the initial basis, $\overline{B}$, equal to I_{n-1}. Clearly, this is not the only solution to this problem, but the scenario occurred rarely enough in practice so that this method was adequate for our purposes. Also, we assumed that we deleted the first row and n^{th} column from B, B^{-1}, N and M_B. The general case can be easily modified to this special case.

To update q_B, we delete the sth element of c, giving us $\overline{c}$. Suppose $q_B = B^{-1}q$ at the termination of Lemke's method, where $q = \begin{bmatrix} c \\ b \end{bmatrix}$. Again, assuming that $s = 1$, we have:

$$q_B = \begin{bmatrix} \tilde{q}_s \\ \tilde{q}_B \end{bmatrix} = B^{-1}q = \begin{bmatrix} U_{row}' & u \\ \overline{U} & U_{col} \end{bmatrix} \begin{bmatrix} q_s \\ \overline{q} \end{bmatrix},$$

where $\tilde{q}_B$ and $\overline{q}$ is the $(n-1)$ lower subvector of q_B and q, respectively and $q_s = c_s$. Similarly to $M_{\bar{B}}$, we get:

$$q_{barB} = E(\tilde{q}_B - q_s U_{col}). \tag{4.15}$$

LU Decomposition of the Basis

In computational experimentations, we saw that maintaining B^{-1} during Lemke's method was costly in terms of runtime. Also, from Equations (4.14) and (4.15), we only require to know one column of B^{-1} to update M and q. Thus, instead of explicitly maintaining B^{-1}, we calculate the LU decomposition of the basis B at the termination of Lemke's method, and use it only to derive the required column of B^{-1}.

We use Crout's algorithm to construct the LU decomposition of $\boldsymbol{B}$, and derive the sth column of $\boldsymbol{B}^{-1}$ using backsubstitution [33]. If $\boldsymbol{x}_s$ is the ith basic variable, then we get $\boldsymbol{U}_{col}$ by deleting the ith element of the the column. Given $\boldsymbol{B}$, $\boldsymbol{N}$ and $\boldsymbol{U}_{col}$, we can update $\boldsymbol{M}$ and $\boldsymbol{q}$ according to Equations (4.14) and (4.15), respectively.

4.1.3 Branching Up

When $\alpha_i > 0$ in Problem (4.1), we need to ensure that $x_s \geq \alpha_s$ in the subsequent subproblems when branching up on x_s. We present two potential methods to maintain this condition: one using reformulation and the other by modifying Lemke's method to deal with nonzero lower-bounds.

Branch Up by Reformulation

We avoid explicitly adding the constraint $x_s \geq \alpha_s$ by replacing $\boldsymbol{x}$ by $\boldsymbol{y}$, where $y_i = x_i$, for $i \neq s$ and $y_s = x_s - \alpha_s$. The $\boldsymbol{A}$ matrix stays the same, but the right-hand-side vector $\boldsymbol{b}$ and the cost vector $\boldsymbol{c}$ need to be updated accordingly.

The original constraints were:

$$\sum_{j \neq s} \boldsymbol{A}_{.j} x_j + \boldsymbol{A}_{.s} x_s \leq \boldsymbol{b}$$

where $\boldsymbol{A}_{.j}$ is the jth column of matrix $\boldsymbol{A}$. Replacing x_s with $y_s + \alpha_s$ gives us

$$\sum_{j \neq s} \boldsymbol{A}_{.j} y_j + \boldsymbol{A}_{.s} y_s \leq \boldsymbol{b} - \alpha_s \boldsymbol{A}_{.s}.$$

Thus, the new right hand side would be $\tilde{\boldsymbol{b}} = \boldsymbol{b} - \alpha_s \boldsymbol{A}_{.s}$.

The objective function is also modified to

$$\frac{1}{2}(\boldsymbol{y} + \alpha_s \boldsymbol{e}_s)' \boldsymbol{Q} (\boldsymbol{y} + \alpha_s \boldsymbol{e}_s) + \boldsymbol{c}'(\boldsymbol{y} + \alpha_s \boldsymbol{e}_s)$$

which simplifies to

$$\frac{1}{2} \boldsymbol{y}' \boldsymbol{Q} \boldsymbol{y} + \tilde{\boldsymbol{c}}' \boldsymbol{y} + C_0$$

where $\tilde{\boldsymbol{c}} = \boldsymbol{c} + \alpha_s \boldsymbol{Q}_{.s}$ and $C_0 = \frac{1}{2}\alpha_s r_s + \alpha_s^2 Q_{s,s}$ is a constant. The reformulated problem is thus:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\boldsymbol{y}'\boldsymbol{Q}\boldsymbol{y} + \tilde{\boldsymbol{c}}'\boldsymbol{y} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{y} \leq \tilde{\boldsymbol{b}}, \\ & \boldsymbol{y} \geq 0, \end{aligned} \tag{4.16}$$

which is in the same format as Problem (4.2). If $\boldsymbol{B}$ is the basis corresponding to the solution of the parent node, then $\boldsymbol{q}_B$ is modified to

$$\tilde{\boldsymbol{q}}_B = \boldsymbol{B}^{-1} \begin{bmatrix} \tilde{\boldsymbol{c}} \\ \tilde{\boldsymbol{b}} \end{bmatrix}$$

If we have the LU decomposition of the basis instead of the entire inverse matrix, then $\tilde{\boldsymbol{q}}_B$ can similarly be calculated by backsubstitution of the vector $\begin{bmatrix} \tilde{\boldsymbol{c}} \\ \tilde{\boldsymbol{b}} \end{bmatrix}$. Note that $\boldsymbol{M}$ need not be updated.

Lemke's Method with Lower Bounds

Another approach to enforce the lower bound, $x_s \geq \alpha_s$, without adding the constraint is to modify the Lemke's method to deal with nonzero lower bounds. Just like in simplex method with lower and upper bounds, the pivoting step can easily be altered if some basic variables have nonzero lower bounds. In a pivot step, we want to increase the value of the driving variable from zero to some positive number, while ensuring that all of the variables remain larger than its lower bound. The basic variable that hits its lower bound first is the blocking variable and would be pivoted out of the basis. Thus, the only modification required is in determining the blocking variable. When branching up on x_s, we need to update the lower bound of that variable for all subsequent subproblems,so that the Lemke's method would prevent x_s from being less than α_s when pivoting in a variable. At the termination of Lemke's method, if a branched-up variable is nonbasic, it's value would equal its lower bound.

Using this modified Lemke's method, we do not need to update the vector $\boldsymbol{q}$ as in the previous section. However, when branching down on subsequent nodes, we need to check whether currently nonbasic variable has been branched up. If U is the set of variables that are branched up and NBV are the set of variables that are currently nonbasic, then when updating the $\boldsymbol{q}$ vector as in Equation (4.15), q_s will equal $c_s + \sum_{i \in U \cap NBV} \alpha_i \Sigma_{s,i}$, since $x_i = \alpha_i$, for $i \in U \cap NBV$ and thus the data need to be updated accordingly.

4.1.4 Additional Algorithmic Ideas within Branch and Bound

To reduce the total number of nodes in the branch-and-bound tree, we introduce heuristics to find strong upper bounds and stop branching after branching up K variables.

Heuristic for Upper Bounds

We use the heuristic similar to those in [6] and [23] at the root. Let $\boldsymbol{x}^*$ be the solution of the continuous relaxation at the root, and $G = \{i | x_i$ has one of $K + W$ largest absolute value out of all n variables $\}$, where W is a user-defined small positive integer such that $|G| = K + W \ll d$. We then set all $x_i = 0$ where $i \in \{1, \ldots, n\} \setminus G$ and branch exclusively on variables x_j where $j \in G$, and also limit the number of nodes to examine.

Solving Subproblem after Branching up K Variables

After branching up on K variables, the remaining variables must be zero in order to be feasible for Problem (4.1). However, conventional branch-and-bound procedures would continue branching down the tree, until all of the variables are branched. We can halt all branching once we branch up K variables, then solve Problem (4.1) only on the K branched up variables. This saves considerable computation time, especially if there are many variables yet to be branched.

4.2 Applications of Quadratic Variable Selection Problems

We focus on applying the methodology described in Section 4.1 to the K-subset selection problem in regression and optimal portfolio selection in Section 4.2.1 and 4.2.2, respectively.

4.2.1 Subset Selection in Regression

Suppose we are given n data points $(\boldsymbol{x}_i, y_i)$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, and we want to choose K variables ($K < d$) that minimizes the total sum of squared errors. We formulate this problem as:

$$\begin{aligned} \text{minimize} \quad & (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta})'(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}) \\ \text{subject to} \quad & |supp(\boldsymbol{\beta})| \leq K. \end{aligned} \tag{4.17}$$

When the cardinality constraint is relaxed, the optimal objective value is $\boldsymbol{Y}'\boldsymbol{Y} - \boldsymbol{Y}'\boldsymbol{X}'(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Y}$, thus we clearly do not need to run the Lemke's method. The main computational work is in the branch down procedure. There are, however, regression problems that have linear constraints with respect to the coefficients. We would use the general methodology in such cases, but let us focus on the unconstrained regression for now.

When branching down on x_s, we delete the sth row and column of $\boldsymbol{X}'\boldsymbol{X}$, and the inverse $(\boldsymbol{X}'\boldsymbol{X})^{-1}$ is updated as illustrated in Section 4.1.2. To further alleviate computation, we set $\boldsymbol{v} = \boldsymbol{X}'\boldsymbol{Y} \in \mathbb{R}^d$ at the root node. Thus, deleting x_s corresponds to deleting the sth element of $\boldsymbol{v}$. Thus, there is no need to multiply $\boldsymbol{X}'$ and $\boldsymbol{Y}$ in subsequent nodes – we simply need to delete corresponding elements from $\boldsymbol{v}$. The optimal objective value of a given node is then:

$$\boldsymbol{Y}'\boldsymbol{Y} - \overline{\boldsymbol{v}}'(\overline{\boldsymbol{X}'\boldsymbol{X}})^{-1}\overline{\boldsymbol{v}}.$$

where $\overline{\boldsymbol{X}'\boldsymbol{X}}$ and $\overline{\boldsymbol{v}}$ are the updated $\boldsymbol{X}'\boldsymbol{X}$ and $\boldsymbol{v}$, respectively. Thus, to calculating the objective requires only matrix-vector multiplications. There is no need to update the subproblem when branching up, since the optimal solution of the parent node is optimal for the next node. Section 4.3.1 illustrates computational results of our approach.

4.2.2 Portfolio Selection

Investors often aim to construct a portfolio of at most K stocks that matches a benchmark portfolio comprised of n stocks ($K < n$) as closely as possible, while minimizing transaction costs and limiting the total investment change within each industry sector. This problem can be represented by the following formulation:

$$\begin{aligned}
\text{minimize} \quad & -\boldsymbol{r}'\boldsymbol{x} + \tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{x}^{\boldsymbol{B}})'\boldsymbol{\Sigma}(\boldsymbol{x}-\boldsymbol{x}^{\boldsymbol{B}}) + \sum_{i \in supp(\boldsymbol{x})} t_i, & & (4.18)\\
\text{subject to} \quad & |\sum_{i \in S_l}(x_i - x_i^B)| \leq \epsilon_l, & \forall l,\\
& \sum_{i=1}^{n} x_i = 1, & \\
& |supp(\boldsymbol{x})| \leq K, & \\
& x_i \geq \alpha_i, & i \in supp(\boldsymbol{x}),\\
& x_i \geq 0, & \forall i,
\end{aligned}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of the rates of return, x_i^B is the benchmark weight for stock i, $\boldsymbol{r}$ is a $n-$dimensional vector of the expected rates of return, α_i is the minimum transaction level of stock i, t_i is the fixed transaction cost of trading stock i, and S_l is the set of indices of stocks in sector l. The first constraint limits the total change in the portfolio weights in sector l to be less than some ϵ_l. The second set of constraints ensures that the weights sum up to 1, and the third constraint limits investing up to K stocks. The fourth set of constraints implies that if we invest in stock i, then x_i must be at least α_i. Clearly, Problem (4.18) is in the form of Problem (4.1) and can be solved using our methodology.

We rewrite Problem (4.18) as

$$
\begin{aligned}
\text{minimize} \quad & -\tilde{\boldsymbol{r}}'\boldsymbol{x} + \tfrac{1}{2}\boldsymbol{x}'\boldsymbol{\Sigma}\boldsymbol{x} + \sum_{i \in supp(\boldsymbol{x})} t_i + C_1, \\
\text{subject to} \quad & \sum_{i \in S_l} x_i \leq \epsilon_l + \sum_{i \in S_l} x_i^B, \\
& -\sum_{i \in S_l} x_i \leq \epsilon_l - \sum_{i \in S_l} x_i^B, \\
& \sum_{i=1}^{n} x_i = 1, \\
& |supp(\boldsymbol{x})| \leq K, \\
& x_i \geq \alpha_i, \qquad i \in supp(\boldsymbol{x}), \\
& x_i \geq 0, \qquad \forall i,
\end{aligned}
\tag{4.19}
$$

where $\tilde{\boldsymbol{r}} = \boldsymbol{r} + \boldsymbol{\Sigma}\boldsymbol{x}^{\boldsymbol{B}}$ and $C_1 = \frac{1}{2}\boldsymbol{x}^{\boldsymbol{B}\prime}\boldsymbol{\Sigma}\boldsymbol{x}^{\boldsymbol{B}}$ is a constant. The key difference between Problem (4.19) and Problem (4.1) is the term for fixed transaction cost, $\sum_{i \in supp(\boldsymbol{x})} t_i$, in the objective value, but our general approach in Section 4.1 can be easily extended to deal with this term. Suppose we introduce binary variables $z_i \in \{0, 1\}$, such that $z_i = 1$ if and only if $x_i > 0$ and $z_i = 0$ if and only if $x_i = 0$. Then we can formulate Problem (4.19) as

$$
\begin{aligned}
\text{minimize} \quad & -\tilde{\boldsymbol{r}}'\boldsymbol{x} + \tfrac{1}{2}\boldsymbol{x}'\boldsymbol{\Sigma}\boldsymbol{x} + \boldsymbol{c}'\boldsymbol{z}, \\
\text{subject to} \quad & \sum_{i \in S_l} x_i \leq \epsilon_l + \sum_{i \in S_l} x_i^B, \\
& \sum_{i \in S_l} x_i \leq \epsilon_l - \sum_{i \in S_l} x_i^B, \\
& \sum_{i=1}^{n} x_i = 1, \\
& \sum_{i=1}^{n} z_i \leq K, \\
& x_i \geq \alpha_i z_i, \qquad \forall i, \\
& x_i \leq z_i, \qquad \forall i.
\end{aligned}
\tag{4.20}
$$

The solution to this relaxation will result in $x_i = z_i$ in the optimal solution, thus we can simply substitute x_i for z_i, and eliminate the need to introduce the variables $\boldsymbol{z}$. Also, we do not include the surrogate constraint since it is always dominated by the constraint $\sum_{i=1}^{n} x_i = 1$.

At a given node, let F be the set of indices of variables that have not yet been branched, U be the set of indices of variables that are branched up, and D be the set of indices of variables that are branched down (i.e., $F \cup U \cup D = \{1, \ldots, n\}$). Also, let $\delta(i)$ be a mapping from set of current variables to the set of original variables, where x_i in the current subproblem corresponds to $x_{\delta(i)}$ in the original problem (this data tracking is necessary due to variable deletion). The subproblem to solve at the current node is:

$$
\begin{aligned}
f(\boldsymbol{x}) = \text{minimize} \quad & \sum_{\delta(i)\in F} (c_i - \tilde{r}_i)x_i + \sum_{\delta(i)\in U} (-\tilde{r}_i + \alpha_i \Sigma_{.i})x_i + \frac{1}{2}\boldsymbol{x}'\boldsymbol{\Sigma}\boldsymbol{x}, \qquad (4.21)\\
\text{subject to} \quad & \sum_{\delta(i)\in S_l} x_i \le \epsilon_l + \sum_{i\in S_l} x_i^B - \sum_{i\in S_l\cap U} \alpha_i,\\
& -\sum_{\delta(i)\in S_l} x_i \le \epsilon_l - \sum_{i\in S_l} x_i^B + \sum_{i\in S_l\cap U} \alpha_i,\\
& \sum_{i=1}^{n} x_i = 1 - \sum_{i\in U} \alpha_i,\\
& x_i \ge 0.
\end{aligned}
$$

If $\boldsymbol{x}^*$ is the optimal solution of the above problem, then the true objective value is $f(\boldsymbol{x}^*) + \sum_{i\in U} c_i + \sum_{i\in U}(\frac{1}{2}\alpha_i r_i + \alpha_i^2 Q_{i,i}) + C_1$, and the solution is $\boldsymbol{x}$ where $x_{\delta(i)} = x_i^* + \alpha_{\delta(i)}$, if $\delta(i) \in U$, $x_{\delta(i)} = 0$, if $\delta(i) \in D$, and $x_{\delta(i)} = x_i^*$ otherwise.

We solve Problem (4.21) using Lemke's method described in Section 4.1.1, and branch down and up on a variable as in Section 4.1.2 and 4.1.3, respectively. Section 4.3.2 illustrates the computational results of this method.

Heuristic Tailored to Portfolio Selection

When K is relatively large (i.e., $K > 100$), solving even for the subset of variables, $G \subset \{x_1, ..., x_d\}$, can become computationally difficult. Since the goal of this root

heuristic is to find a good feasible solution quickly, we may limit $|G|$ to be smaller than K – perhaps 10% of the total number of variables. Unlike in the subset selection problem in regression, where solutions to most subproblems have no nonzero values, the nonnegativity constraints and the fixed costs make the solution to the portfolio problems have substantial number of variables equal to zero. Thus, the cardinality constraint in the optimal solution may not necessary be tight. Also, although the set of variables in the optimal solution may not coincide with our choice of G, we see that diminishing the size of our root problem gave us good feasible solutions, as we illustrate in Section 4.3.2.

4.3 Computational Experimentation

We describe our computational experimentation on subset selection and portfolio selection problems in Section 4.3.1 and 4.3.2, respectively. Each section gives the performance results of our methods compared to alternative approaches, and describe the advantages and weaknesses of our method.

4.3.1 Results for Subset Selection

We compared our implicit branch-and-bound method of solving subset selection with forward regression and an explicit branch-and-bound model solved by Cplex 8.0 [1] [12]. Forward regression is a greedy heuristic that adds each variable that reduces the residual error the most, given the variables already chosen, i.e., the first variable chosen, $x_{(1)}$, minimizes $\sum_i (y_i - x_{i,j}\beta_j)^2$. The next variable minimizes $\sum_i (\bar{y}_i - x_{i,j}\beta_j)^2$, where $\bar{y}_i = y_i - x_{i,(1)}\beta_{(1)}$ and $\beta_{(1)}$ is the regression coefficient for $x_{(1)}$ found in the previous step. This step is repeated until K variables are chosen [31]. We used CPLEX 8.0's quadratic mixed-integer optimizer to solve Problem (4.17) by introducing binary inclusion variables z_i, replacing the cardinality constraint by $\sum_i z_i \leq K$ and adding constraints $\beta_i \geq -Mz_i$ and $\beta_i \leq Mz_i$, where M is some large positive number. We found that setting $M = 100$ was sufficiently large to solve our generated

[1] CPLEX 8.0 is a product of ILOG http://www.ilog.com/products/cplex/

			Forward	ImplicitBnB			CplexMIQP		
d	K	n	RSS	CPU sec	# nodes	RSS	CPU sec	# nodes	RSS
20	10	100	4,464	0.016	263	2,176	0.096	197	2,176
20	5	100	9,819	0.010	224	6,765	0.118	227	6,765
50	40	500	47,755	0.914	1,023	2,156	19.402	15,679	2,156
50	20	500	125,520	34.226	98,365	55,753	54.204	50,037	60,597
100	80	1000	383,666	60.000	18,623	16,896	60.000	16,723	692,896
100	50	1000	386,922	60.000	23,291	115,388	60.000	19,222	528,967
100	20	1000	696,772	60.000	68,000	541,950	60.000	18,825	570,801

Table 4.1: Results for Subset Selection. d is the number of variables, K is the size of the selected subset, n is the number of data points and RSS is the residual sum of squares.

problems effectively. CPLEX 8.0 uses a pivoting method similar to Lemke's method to solve the relaxation problem at each node. It does not, however, delete variables nor reformulate the problem during branching.

For each n (the number of observations), d (the number of total variables), and K (the size of the desired subset), we randomly generated instances of $\boldsymbol{X}$ and $\boldsymbol{\beta}$, and set $\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{\epsilon}$, where $\epsilon_i \sim N(0,1)$ for each i. For each problem size, we generate five such instances and averaged the performances of the methods over all of them. We also set the time limit to 60 CPU seconds. Our implicit branch-and-bound and CPLEX's branch-and-bound search procedure were set depth first search, and branches on the variable with maximum absolute value first. Table 4.1 illustrates the performance results of our model, forward regression and CPLEX.

The columns "Forward", "ImplicitBnB", and "CplexMIQP" corresponds to the results of the forward regression, our method, and CPLEX, respectively. We did not record the running time for forward regression, since the simple heuristic was able to solve all the instances in a fraction of a second. The column labelled "CPU sec" is the total CPU time required to solve the problem up to 60 CPU seconds. "ImplicitBnB" solved all the instances to optimality except for $d = 100$. "CplexMIQP" hit the 60 CPU limit before getting a provable optimal solution for all instances with $d = 100$, and four instances with $d = 50, K = 20$. The column labelled "nodes" is the total number of nodes in the branch-and-bound tree at termination, and "RSS" is the best total sum of squared errors for subsets of size K found. Both the number of nodes

			Forward	ImplicitBnB			CplexMIQP		
d	K	n	RSS	# nodes	best node	RSS	#nodes	best node	RSS
50	40	500	24,116	183	0	1,075	55,399	55,358	1,075
50	20	500	111,735	710,243	17,657	56,708	915,159	892,052	56,708
100	80	1000	326,914	257,003	74,186	14,234	992,506	987,366	365,582
100	50	1000	555,370	779,800	2,840	128,936	1,086,256	0	737,592
100	20	1000	862,900	1,959,170	8,292	624,438	1,006,151	0	657,418

Table 4.2: Results for Subset Selection with 3600 CPU seconds. d is the number of variables, K is the size of the selected subset, n is the number of data points and RSS is the residual sum of squares.

and RSS were rounded to the nearest integer.

It is apparent from Table 4.1 that the optimization approaches significantly improves upon the forward regression in terms of residual sum of squares, even when they do not solve to provable optimality. "ImplicitBnB" is also finds strong feasible solutions significantly faster than the explicit branch-and-bound formulation solved by CPLEX, mainly because the former takes great advantage of the special structure of Problem (4.17), which does not require introducing binary variables.

Even for several hundred variables, the overall node-to-node computation time in our model takes a fraction of a CPU second. Also, the heuristic in Section 4.1.4 often finds the optimal objective value at the root. The story does not seem to change when using longer running times. Table 4.2 illustrates results when our method and CPLEX are run for 3600 CPU seconds (note: these were run on similar but different data sets that those of Table 4.1). The column labelled "best node" refers to the node that found the best feasible solution. Although CPLEX was now able to solve problems with $n = 50$ to optimality (though not always provable optimality) within 3600 CPU seconds, the solutions are found significantly later than by our method.

The main bottleneck, however, is the number of nodes needed to prove optimality. Even when the heuristic finds the optimal solution at the root, the branch-and-bound tree can grow to several million nodes to prove optimality, even for moderate sized problems (e.g. $d = 50$). The main factors preventing significant pruning of the tree are the free variables and the lack of constraints. A subproblem solution almost always has all non-zero variables.

4.3.2 Results for Portfolio Selection

We tested our approaches described in Section 4.2.2 against two alternative methods. One method uses CPLEX's quadratic barrier method to solve the relaxation problem (4.21), and the second uses CPLEX's quadratic mixed-integer solver to solve the explicit integer formulation, as in Problem (4.20). We also tested reformulation versus using Lemke's method with lowerbounds when branching up. Again, for each n (number of total assets), S (number of sectors), and K (the upper bound on diversification), we generated five random instance of Problem (4.18) and averaged the results. All branch-and-bound procedures were set to depth first search, branch up first and branch on variable with maximum absolute value. Tables 4.3 and 4.4 illustrates the results.

"LemkeRef" is the combination of Lemke's method and implicit branch-and-bound using reformulation when branching up. "LemkeLB" is the same, but uses Lemke's method with lowerbound instead of reformulation. "Barrier" is also implemented with implicit branch-and-bound, but uses CPLEX's barrier method to solve the continuous quadratic optimization problem. Finally, "CplexMIQP" is the result of using using CPLEX quadratic mixed-integer solver. All four methods were run for a total of 120 CPU seconds. The column labelled "nodes" is the average of the total number of nodes each method explored, "best node" is the average of the node where the best feasible solutions were found, and "UB" is the best feasible objective value found within the time limit.

Table 4.3 runs all four methods without running any heuristic methods to find a good feasible solution, whereas Table 4.4 runs such a heuristic once at the root. For "LemkeRef", "LemkeLB" and "Barrier", we ran the heuristic described in Section 4.1.4 for at most 30 CPU seconds after the root is solved. For every instance, we ran the heuristic using $K = 0.1n$ and $|G| = K + 10$. We were not able to put a time limit on the heuristic for CplexMIQP, which ran until Cplex deemed it had an acceptable upperbound. When the size of K was relatively large ($K > 0.1n$), this heuristic ran for at most 10 CPU seconds, whereas it can last over 100 CPU seconds when n is

			LemkeRef			LemkeLB		
n	K	S	nodes	best node	UB	nodes	best node	UB
100	50	10	17486.20	16227.80	28.26	16992.20	16039.60	28.46
100	10	4	26937.20	8345.40	18.83	27231.40	8329.40	18.83
200	100	10	2936.60	2924.80	142.80	2952.80	2933.80	142.73
200	20	4	6924.80	1281.00	38.73	6913.20	1281.00	38.73
500	200	10	142.20	-	-	142.40	-	-
500	100	10	161.60	126.20	159.71	164.40	137.80	159.68
500	50	4	63.80	44.00	91.06	64.20	46.40	90.89

			Barrier			CplexMIQP		
n	K	S	nodes	best node	UB	nodes	best node	UB
100	50	10	4526.80	4472.00	44.51	119697.00	76987.00	19.02
100	10	4	5686.00	1678.20	19.57	58530.20	12736.60	18.49
200	100	10	751.20	729.40	150.57	30188.60	30062.80	81.86
200	20	4	1284.00	656.20	39.77	7236.80	1245.60	38.73
500	200	10	30.80	-	-	4864.00	4812.60	345.46
500	100	10	33.20	-	-	782.60	398.00	158.68
500	50	4	35.40	-	-	226.40	140.40	90.89

Table 4.3: Results for portfolio selection, without Heuristic, solved until 120 CPU seconds. n is the number of variables, K is the size of the selected subset, S is the number of sectors (Problem 4.18), "UB" is the best feasible solution found, "best node" is the node where "UB" was found and "nodes" is the total number of nodes explored.
9in

large ($n > 200$) and K is small compared to n (e.g., $K = 0.1n$).

The node to node runtime of "LemkeRef" and "LemkeLB" were statistically the same, thus avoiding reformulation did not seem to gain significant running time. Since the computational characteristics of both methods were very similar, we will refer to both methods as "Lemke" from here on end. Both pivoting methods, "Lemke" and "CplexMIQP", are significantly faster than "Barrier" for every instance. Although the relative difference in the total number of nodes explored did decrease as the problem size increased, the advantage of interior point methods in large dimensions over pivoting methods did not compensate the latter's advantage in solving from the previous solution. For example, for problems that would take an average of 400 pivots

			LemkeRef			LemkeLB		
n	K	S	nodes	best node	UB	nodes	best node	UB
100	50	10	14235.80	0.00	12.93	13998.40	0.00	12.93
100	10	4	23926.40	0.00	14.87	23858.00	0.00	14.87
200	100	10	2501.00	0.00	32.90	2490.60	0.00	32.90
200	20	4	6248.80	58.60	34.87	6232.80	58.60	34.87
500	200	10	117.80	0.00	83.93	119.40	0.00	83.93
500	100	10	132.80	0.00	80.40	134.00	0.00	80.40
500	50	4	47.20	0.00	81.53	47.00	0.00	81.53

			Barrier			CplexMIQP		
n	K	S	nodes	best node	UB	nodes	best node	UB
100	50	10	4029.40	2678.20	36.36	116468.20	70551.00	18.68
100	10	4	4166.40	0.00	15.66	58410.40	12736.60	18.49
200	100	10	485.40	0.00	34.51	18533.20	7800.80	52.84
200	20	4	916.40	58.60	35.58	6937.00	1245.60	38.73
500	200	10	24.60	-	-	1098.00	0.00	138.83
500	100	10	27.00	-	-	1097.40	0.00	140.29
500	50	4	27.40	-	-	146.40	103.20	91.14

Table 4.4: Results for portfolio selection, with running Heuristic, solved until 120 CPU seconds.n is the number of variables, K is the size of the selected subset, S is the number of sectors (Problem 4.18), "UB" is the best feasible solution found, "best node" is the node where "UB" was found and "nodes" is the total number of nodes explored.
9in

to solve from scratch, any intermediary node would require only about 5 pivots to resolve the subproblem of that node. Also, the pivoting methods always gives a basic feasible solution to the quadratic problem. Thus, it is guaranteed to give the solution to the relaxation with the minimum support, unlike the interior point method. Since many of the instances generated and most real world problems do not guarantee positive definiteness (only positive semi-definiteness) in the quadratic matrix, this difference can be another significant advantage.

It is clear that the node to node computation time of "CplexMIQP" is faster than "Lemke" for most instances. However, the relative difference significantly decreases as K becomes small relative to n. For example, when $K \approx 0.5n$, the difference in number of nodes explored is about a factor of 10, whereas when $K \approx 0.1n$, it reduces to a factor of 2 to 3. For the case when $n = 200$ and $K = 20$, the total computation time of "Lemke" and "CplexMIQP" are about the same. We believe that the gap gets closer when K is small, since "Lemke" takes advantage of the special structure of Problem (4.18) more than the explicit formulation can, which becomes increasingly important and thus advantageous as the K becomes smaller. The story does not change when we significantly increase the running time. Table 4.5 are the results for running our method with Lemke's with lower bounds and CPLEX for 3600 CPU seconds.

Running the heuristic, even for just 30 CPU seconds, brought significant improvement to our models in terms of finding good feasible solutions. Again, we tailored our heuristic to the special structure of the problem. Using $|G|$ small enough so that Lemke's method can run sufficiently fast allowed us to find a good feasible solution quickly.

4.4 Conclusion

Our tailored approaches for solving quadratic variable selection problems in statistics and finance show computational advantages over general mixed-integer optimization formulation and significantly better solution quality than simple heuristic methods.

			Lemke			CplexMIQP		
n	K	S	nodes	best node	UB	nodes	best node	UB
500	200	10	6,100	0	83.93	49,720	0	138.83
500	100	10	5,960	0	80.40	55,183	0	140.29
500	50	4	13,828	0	80.01	15,078	7,995	89.34

Table 4.5: Results for portfolio selection, with root Heuristic, solved for 3600 CPU seconds.

For subset selection in regression, the optimization based approaches were able to find the subset of variables with significantly better fit than the forward regression heuristic. Also, since we took advantage of the special structure of the problem, our implicit branch-and-bound formulation had faster node-to-node running times than CPLEX's quadratic mixed-integer solver.

For the portfolio selection problem, the combination of implicit branch-and-bound and Lemke's method has significantly faster running times compared to using the barrier method to solve the continuous quadratic optimization problem. The key bottleneck for efficient quadratic mixed-integer optimization has been the inability of interior point methods to start at infeasible points. Although they are undoubtedly more effective in solving high dimensional quadratic optimization problems than pivoting methods started from scratch, the pivoting methods can re-solves each subproblem more efficiently at each node of the branch-and-bound tree.

CPLEX's quadratic mixed-integer solver has a more sophisticated pivoting and branch-and-bound implementation, yet our tailored approach compensates for our lack of software engineering prowess. Our root heuristic finds good upper bounds quickly, and our variable deletion, reformulation and modified Lemke's method updates each subproblem without increasing the size of the problem. With further improvements in implementation (e.g., regarding data structures, decompositions, and memory handling), we believe our methodology will have comparable node-to-node running times.

There are several potential followup work to our model. We implemented all of our branch-and-bound procedures using depth-first-search, due to the ease of pro-

gramming. Although we can find good upper bounds faster with this approach, we often get stuck in a local subtree. Also, with large number of nodes, we cannot utilize best lower bounds effectively, since the root relaxation would be the lower bound for a large majority of the nodes we explore. There is also merit in investigating other principal pivoting techniques other than Lemke's. The main drawback of Lemke's method is the lack of choice in the driving variable. Alternative pivoting methods, though more difficult to start, have the flexibility of choosing amongst several driving variables, akin to the simplex method. It also does not augment the LCP, thus not introducing an auxiliary variable and column. These properties may allow us to converge faster to the solution.

Finally, we would like to expand the applications of our methodologies. For example, we can extend the regression problem to those with linear constraints. There may be bounds on the value of the regression coefficients, and limitations on the changes in the regression coefficients in time-series regression [5]. We would be able to use our general methodology to solve this combined subset selection and constrained regression problem. Also, we can clearly extend our methodologies to general quadratic mixed-integer optimization as well.

Chapter 5

Conclusions

Classification, regression and variable selection are key problems in data mining. Classification and regression are particular types of prediction problems, arising in numerous fields, including marketing, finance, healthcare and artificial intelligence. Variable selection is a more general tool, where the essential variables are selected in prediction or optimization problems.

Due to the importance of these problems and their inherent difficulty, a large range of work has been done in developing algorithms from both the statistics and computer science community. Many of these have been successful and popular, including those discussed in this thesis, however, there seems to be a tradeoff between prediction accuracy, robustness, complexity and computation time. Since classification, regression and variable selection all exhibit complex combinatorial properties, a large portion of the existing methodologies solve them using simple heuristics. For the most part, developers in this community believe that integer optimization models are intractable, and are also unfamiliar with modelling problems as integer programs.

Our main goal in this thesis is to introduce integer optimization in the context of data mining, and illustrate its effectiveness and practical efficiency. CRIO incorporated existing methods with integer optimization models to solve classification and regression problems. Our quadratic variable selection method uses a combination of implicit branch-and-bound and Lemke's method to solve variable selection problems in regression and portfolio selection. Both CRIO and our quadratic variable selection

solver are able to solve these problems comparably and often significantly better than the current state-of-the-art tools, under practical running times.

The key components of CRIO include: (1) Clustering methods to reduce dimensionality; (2) Nonlinear transformations of the variables to improve predictive power; (3) Mixed-integer optimization methods to simultaneously group points together and eliminate outlier data; and (4) Continuous optimization methods (linear and quadratic optimization) to represent groups by polyhedral regions. We feel that the key advantage of CRIO is its use of mixed-integer optimization that allows us to partition the data in a more globally optimally manner compared to the current leading methods. We may further experiment on the various clustering approaches, and formalize parameter selection.

Our quadratic variable selection solver efficiently solves variable selection in regression and portfolio selection problems. The combination of implicit branch-and-bound, Lemke's method, variable deletion and reformulation all contribute to keep the problem size small and minimize the computation time. On the subset selection problem, our approach found good feasible solutions faster than running CPLEX on the explicit formulation, and its average error was significantly better than the forward regression heuristic. On portfolio selection problems, our approach was consistently faster than using barrier method to solve the continuous quadratic programming problem, and comparable to CPLEX's quadratic mixed-integer solver. Future work will entail experimentations with different pivoting methods, matrix decomposition and variations in node selection strategies. Also, this method can clearly be extended to solve general quadratic mixed-integer optimization problems.

More generally, our preliminary work is strong evidence of integer optimization as a powerful and practical tool for statistical computing. It is able to capture characteristics in data that continuous optimization and heuristic methods cannot. Also, with appropriate tailored formulations, we are able to solve these integer optimization problems in practical time. We hope these encouraging results will motivate the statistics and data mining community to re-examine the potential of integer optimization methods in their fields.

Appendix A

Lemke's Method

Chapter 4 gave a general outline of Lemke's method for solving linear complementarity problems. This appendix provides deeper theoretical properties of the algorithm, including its guarantee to solve convex quadratic optimization problems in finite time. Lemke's method and special properties of LCPs have been studied at great length [11, 15, 26], thus the results we illustrate can be attributed to several independent sources. However, since many of them deal with general LCPs, there is no single source that gives a comprehensive and cohesive picture of Lemke's method on quadratic optimization problems. For that reason, we present a summary of the key properties of Lemke's method specifically for solving convex quadratic optimization problems. We assume throughout the entire section that our problem is nondegenerate. [11] and [15] show these results hold for the degenerate case when using lexicographic formulation.

The following is the key result:

Theorem 4 *Let $\boldsymbol{q}$ and $\boldsymbol{M}$ be constructed from the convex quadratic optimization problem (4.7) using Equation (4.10). Lemke's method solves LCP($\boldsymbol{q}, \boldsymbol{M}$) in finite time, i.e., if the QP is feasible Lemke's terminates with the optimal solution. Otherwise, it indicates that the problem is infeasible.*

We construct the proof for Theorem 4 by series of lemmas. The following links the solutions of convex quadratic optimization problems to that of LCPs using the

KKT condition.

Lemma 5 *A solution LCP($\boldsymbol{q}, \boldsymbol{M}$) with $\boldsymbol{q}$ and $\boldsymbol{M}$ as in Equation (4.10) has one-to-one correspondence with the optimal solution to the QP (4.7). Also, LCP($\boldsymbol{q}, \boldsymbol{M}$) is infeasible if and only if the QP is infeasible.*

Proof: The KKT conditions are necessary and sufficient for finding optimal solutions to convex quadratic optimization problems with linear constraints (notice, this is the global optimum due to convexity). Again, Equation 4.9 are the KKT equations for the QP (4.7). Thus, if $\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$ is a solution for Equation (4.9), then $\boldsymbol{x}$ is an optimal solution for the QP (4.7). Conversely, if $\boldsymbol{x}$ is an optimal primal solution and $\boldsymbol{y}$ the optimal dual solution for QP (4.7), then $\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$ is a solution to Equation (4.9). Thus, Equation (4.9) is infeasible if and only if the QP (4.7) is infeasible. □

From Lemma 5, if we can show that Lemke's method solves LCP($\boldsymbol{q}, \boldsymbol{M}$), then it solves the convex quadratic optimization problem. We now focus on how Lemke's method can be guaranteed to solve LCP($\boldsymbol{q}, \boldsymbol{M}$).

There are two possible scenarios when Lemke's method terminates:

1. z_0 is pivoted out of the basis, or

2. no blocking variable is found in the last pivot.

If Lemke's terminates with the first scenario, it is clear that it has found a solution to the LCP. At each pivot (subsequent to the initial one when z_0 was pivoted into the basis), the complementarity condition (4.5) and the nonnegativity constraint (4.4) are satisfied. However, the basic solutions at each pivot satisfies the linear equalities of the augmented problem (4.12) but not Equation (4.3) since $z_0 > 0$. Thus, when z_0 is pivoted out of the basis, then $z_0 = 0$ and now the basic solution satisfies the condition for the original problem represented by Equations (4.3)-(4.5).

Before discussing the implications of the second scenario, let us define *secondary rays* in the context of the augmented problem:

Definition 2 (Secondary Rays) *Suppose $(\boldsymbol{w}^*, \boldsymbol{z}^*, z_0^*)$ is a basic complementary solution to the augmented LCP$(\boldsymbol{q} + \boldsymbol{d}z_0, \boldsymbol{M})$. The vectors$(\tilde{\boldsymbol{w}}, \tilde{\boldsymbol{z}}, \tilde{z}_0) \neq \boldsymbol{0}$ is a secondary ray associated with this basis, if for every $\lambda > 0$:*

$$
\begin{aligned}
(\boldsymbol{w}^* + \lambda\tilde{\boldsymbol{w}}) &= \boldsymbol{q} + \boldsymbol{d}(z_0^* + \lambda\tilde{z}_0) + \boldsymbol{M}(\boldsymbol{z}^* + \lambda\tilde{\boldsymbol{z}}), \\
\boldsymbol{w}^* + \lambda\tilde{\boldsymbol{w}} &\geq \boldsymbol{0}, \\
\boldsymbol{z}^* + \lambda\tilde{\boldsymbol{z}} &\geq \boldsymbol{0}, \\
z_0^* + \lambda\tilde{z}_0 &\geq 0, \\
(\boldsymbol{w}^* + \lambda\tilde{\boldsymbol{w}})'(\boldsymbol{z}^* + \lambda\tilde{\boldsymbol{z}}) &= 0.
\end{aligned}
$$

In other words, we can move in the direction $(\tilde{\boldsymbol{w}}, \tilde{\boldsymbol{z}}, \tilde{z}_0)$ from the basic complementary point $(\boldsymbol{w}^*, \boldsymbol{z}^*, z_0^*)$ arbitrarily far and still satisfy conditions for LCP$(\boldsymbol{q} + \boldsymbol{d}z_0, \boldsymbol{M})$.

Lemma 6 describes a certificate for infeasibility for general LCPs.

Lemma 6 *For any square matrix $\boldsymbol{M}$ and vector $\boldsymbol{q}$, if there exists $\boldsymbol{r} \geq \boldsymbol{0}$ such that $\boldsymbol{M}'\boldsymbol{r} \leq \boldsymbol{0}$ and $\boldsymbol{q}'\boldsymbol{r} < 0$, then LCP$(\boldsymbol{q}, \boldsymbol{M})$ is infeasible.*

Proof: We show this by contradiction. Suppose there exist a solution $\boldsymbol{w}$ and $\boldsymbol{z}$ to LCP$(\boldsymbol{q}, \boldsymbol{M})$. Thus,

$$\boldsymbol{q} = \boldsymbol{w} - \boldsymbol{M}\boldsymbol{z}, \quad \boldsymbol{z}'\boldsymbol{w} = 0, \quad \boldsymbol{w} \geq 0, \boldsymbol{z} \geq 0.$$

Multiplying the linear equality by $\boldsymbol{r}$, we get

$$\boldsymbol{q}'\boldsymbol{r} = \boldsymbol{w}'\boldsymbol{r} - \boldsymbol{z}'\boldsymbol{M}'\boldsymbol{r} < 0.$$

However, $\boldsymbol{M}'\boldsymbol{r} \leq \boldsymbol{0}$ and $\boldsymbol{z} \geq \boldsymbol{0}$, thus $-\boldsymbol{z}'\boldsymbol{M}'\boldsymbol{r} \geq 0$, which implies $\boldsymbol{w}'\boldsymbol{r} - \boldsymbol{z}'\boldsymbol{M}'\boldsymbol{r} \geq 0$, a contradiction. □

We use the properties of secondary rays and Lemma 6 to show infeasibility of LCP$(\boldsymbol{q}, \boldsymbol{M})$ when Lemke's method terminates with the second scenario. Note, that $\boldsymbol{M}succeq0$ since we are concerned only convex quadratic optimization problems, i.e.,

$$Q \succeq 0, \Rightarrow M \succeq 0 \text{ since } z'Mz = \begin{bmatrix} x' & y' \end{bmatrix} \begin{bmatrix} Q & A' \\ -A & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = x'Qx \geq 0, \quad \forall z.$$

Lemma 7 *If Msucceq0 and Lemke's method cannot find a blocking variable, then the original LCP(q, M) is infeasible.*

Proof: In a pivot step, we increase the value of a nonbasic variable (the driving variable) from zero to some positive number until a basic variable decreases to zero. If there is no blocking variable, we can increase the driving variable arbitrarily large while maintaining feasibility and complementarity. Thus, the basic ray associated to this pivot must be a secondary ray by definition.

Suppose $(\tilde{w}, \tilde{z}, \tilde{z}_0)$ is this secondary ray, and the current basic complementary solution was (w, z, z_0). Clearly, $z_0 > 0$, else we would have a solution to LCP(q, M). We show that $\tilde{z}$ is such that $\tilde{z} \geq 0$, $M'\tilde{z} \leq 0$ and $q'\tilde{z} < 0$ and use Lemma 6 to prove infeasibility of LCP(q, M).

From the definition of secondary rays and some algebraic manipulation, we have:

$$\tilde{w} = d\tilde{z}_0 + M\tilde{z}, \tag{A.1}$$
$$w'\tilde{z} = 0, \quad \tilde{q}'z = 0, \quad \tilde{w}'\tilde{z} = 0.$$

Further, we show that $\tilde{z} \neq \mathbf{0}$, by contradiction. Suppose $\tilde{z} = \mathbf{0}$. Since $(\tilde{w}, \tilde{z}, \tilde{z}_0) \neq 0$, this implies $\tilde{z}_0 > 0$ (otherwise $\tilde{w} = d\tilde{z}_0 = \mathbf{0}$). Thus, $\tilde{w} = d\tilde{z}_0 > \mathbf{0}$ (since $d > \mathbf{0}$), and from the complementarity condition above, $\tilde{w}'z = 0$, $\Rightarrow z = \mathbf{0}$. From the nondegeneracy assumption, this implies that we terminated with the initial basis since a basis cannot repeat (i.e., the path of almost complementary points cannot have a cycle). However, in the initial basis, there was exactly one choice of driving variable to increase. Thus, $\tilde{z}$ corresponding to this basic ray must have a positive element. Thus, $\tilde{z} \neq \mathbf{0}$.

With $\tilde{z} \neq \mathbf{0}$, we have the following:

$$-\tilde{w}'\tilde{z} = \tilde{z}_0 d'\tilde{z} + \tilde{z}'M\tilde{z} = 0, \quad \text{(from (A.1))}$$
$$\Rightarrow -\tilde{z}_0 d'\tilde{z} = \tilde{z}'M\tilde{z}.$$

Since we are given that $M \succeq 0$, we get:

$$
\begin{aligned}
&-\tilde{z}_0 \boldsymbol{d}'\tilde{\boldsymbol{z}} = \tilde{\boldsymbol{z}}'M\tilde{\boldsymbol{z}} \geq 0, \\
&\Rightarrow \tilde{z}_0 \boldsymbol{d}'\tilde{\boldsymbol{z}} \leq 0, \\
&\Rightarrow \tilde{z}_0 = 0, \quad (\text{since } \boldsymbol{d}'\tilde{\boldsymbol{z}} > 0 \text{ due to } \tilde{\boldsymbol{z}}), \\
&\Rightarrow \tilde{\boldsymbol{z}}'M\tilde{\boldsymbol{z}} = 0, \\
&\Rightarrow M\tilde{\boldsymbol{z}} = -M'\tilde{\boldsymbol{z}} \quad (\text{from positive semi-definiteness of } M).
\end{aligned}
$$

Since $\tilde{z}_0 = 0 \Rightarrow \tilde{\boldsymbol{w}} = M\tilde{\boldsymbol{z}} = -M'\tilde{\boldsymbol{z}}$. Thus, we have $M'\tilde{\boldsymbol{z}} \leq \mathbf{0}$ from the nonnegativity of $\tilde{\boldsymbol{w}}$.

All that is left to show is $\boldsymbol{q}'\tilde{\boldsymbol{z}} < 0$, as illustrated below:

$$
\begin{aligned}
& \boldsymbol{q} = \boldsymbol{w} - M\boldsymbol{z} - \boldsymbol{d}z_0, \\
\Rightarrow \quad & \boldsymbol{q}'\tilde{\boldsymbol{z}} = \boldsymbol{w}'\tilde{\boldsymbol{z}} - \boldsymbol{z}'M'\tilde{\boldsymbol{z}} - z_0\boldsymbol{d}'\tilde{\boldsymbol{z}}, \\
& = -\boldsymbol{z}'M'\tilde{\boldsymbol{z}} - z_0\boldsymbol{d}'\tilde{\boldsymbol{z}} \quad (\text{form complementarity}), \\
& = \boldsymbol{z}'M\tilde{\boldsymbol{z}} - z_0\boldsymbol{d}'\tilde{\boldsymbol{z}} \quad (\text{from } M \succeq 0), \\
& = \boldsymbol{z}'\tilde{\boldsymbol{w}} - z_0\boldsymbol{d}'\tilde{\boldsymbol{z}} \quad (\text{from } \tilde{z}_0 = 0), \\
& = -z_0\boldsymbol{d}'\tilde{\boldsymbol{z}} < 0.
\end{aligned}
$$

Thus, $\tilde{\boldsymbol{z}}$ corresponding to the secondary ray gives a certificate of infeasibility for LCP($\boldsymbol{q}$, M). □

From Lemma 6 and 7, we see that Lemke's method terminates with either a solution to the LCP (and thus the QP) or indicates that the LCP (and thus the QP) is infeasible. Under the nondegeneracy assumption, the algorithm is finite since there are only finite numbers of complementary bases and bases cannot repeat.

Finally, we conclude with the following result:

Lemma 8 *Given $M \succeq 0$, z_0 decreases monotonically with each pivot step of the Lemke's method on LCP($\boldsymbol{q}, M$).*

Proof: [15] shows this result for all $M \in P_0$ (matrices with nonnegative principal minors), but we summarize the result just for the positive semi-definite case.

We denote the set of basic variables at the lth pivot as β^l, and $\beta^1, \beta^2, \ldots, \beta^k$ is the pivot sequence that Lemke's method followed. Also, let $z_0(\beta^l)$ be the z_0 value corresponding to the basis β^l.

First, we show that $z_0(\beta^1) \geq z_0(\beta^l)$, for all $k = 1, \ldots, k$. Suppose, β^l is the first basis where $z_0(\beta^l) > z_0(\beta^1)$, $1 < l \leq k$. Recall that $z_0(\beta^1)$ was selected so that $\boldsymbol{q} + z_0(\beta^1)\boldsymbol{d} \geq \boldsymbol{0}$, and the corresponding basis was such that $\beta^1 \cap \{z_1, \ldots, z_n\} = \emptyset$. However, β^1 is a solution to the augmented LCP where $z_0 = z_0(\beta^k) > z_0(\beta^1)$. Thus, $\beta^k = \beta^1$, which is a contradiction due to nondegeneracy.

Second, we show that $z_0(\beta^1) \geq z_0(\beta^2) \geq, \ldots, \geq z_0(\beta^k)$. Suppose, β^l is such that $z_0(\beta^l) > z_0(\beta^{l-1})$. From the mean value theorem, there exist j, $1 \leq j < l$, such that:

$$Z^* = \lambda z_0(\beta^{j-1}) + (1-\lambda) z_0(\beta^j) = \mu z_0(\beta^{l-1}) + (1-\mu) z_0(\beta^l),$$

where $\lambda \in [0,1]$ and $\mu \in [0,1]$. Let $(\boldsymbol{w}^j, \boldsymbol{z}^j)$ be the solution corresponding to β^j, and define $(\bar{\boldsymbol{w}}, \bar{\boldsymbol{z}})$ and $(\tilde{\boldsymbol{w}}, \tilde{\boldsymbol{z}})$ as

$$\begin{aligned}
\bar{\boldsymbol{w}} &= \lambda \boldsymbol{w}^{j-1} + (1-\lambda)\boldsymbol{w}^j, \\
\bar{\boldsymbol{z}} &= \lambda \boldsymbol{z}^{j-1} + (1-\lambda)\boldsymbol{z}^j, \\
\tilde{\boldsymbol{w}} &= \mu \boldsymbol{w}^{l-1} + (1-\mu)\boldsymbol{w}^l, \\
\tilde{\boldsymbol{z}} &= \mu \boldsymbol{z}^{l-1} + (1-\mu)\boldsymbol{z}^l.
\end{aligned}$$

Thus, $(\bar{\boldsymbol{w}}, \bar{\boldsymbol{z}})$ and $(\tilde{\boldsymbol{w}}, \tilde{\boldsymbol{z}})$ are two distinct nondegenerate complementary solutions for LCP$(\boldsymbol{q} + Z^*\boldsymbol{d}, \boldsymbol{M})$. This implies

$$\begin{aligned}
\bar{\boldsymbol{w}} - \boldsymbol{M}\bar{\boldsymbol{z}} &= \tilde{\boldsymbol{w}} - \boldsymbol{M}\tilde{\boldsymbol{z}}, \\
\Rightarrow \bar{\boldsymbol{w}} - \tilde{\boldsymbol{w}} &= \boldsymbol{M}(\bar{\boldsymbol{z}} - \tilde{\boldsymbol{z}}).
\end{aligned}$$

Multiplying both sides by $(\bar{z} - \tilde{z})'$, we get

$$(\bar{w} - \tilde{w})'(\bar{z} - \tilde{z}) = (\bar{z} - \tilde{z})'M(\bar{z} - \tilde{z}).$$

We see that $(\bar{w} - \tilde{w})'(\bar{z} - \tilde{z}) = -\bar{w}'\tilde{z} - \tilde{w}'\bar{z} \leq 0$. Also, since $M \succeq 0$, $\Rightarrow (\bar{z} - \tilde{z})'M(\bar{z} - \tilde{z}) \geq 0$. Thus, we get

$$\bar{w}'\tilde{z} = \tilde{w}'\bar{z} \tag{A.2}$$

However, note that $supp(\bar{w}) \cap supp(\tilde{z}) \neq \emptyset$ and $supp(\bar{z}) \cap supp(\bar{w}) \neq \emptyset$ since $j \neq l$ and bases do not repeat. Thus, from nonnegativity and (A.2), we get a contradiction. $\square$

Bibliography

[1] Arthanari, T.S., Dodge, Y. (1993). *Mathematical Programming in Statistics*, Wiley, Wiley Classics Library Edition, New York.

[2] Beale, E.M.L.; Kendall, M.G.; Mann, D.W. (1967) "The Discarding of Variables in Multivariate Analysis", *Biometrika*, Vol. 54, Issue 3/4, 357 - 366.

[3] Bennet, K. P., Blue, J. A. (1984). Optimal Decision Trees, *R.P.I. Math Report No.214*.

[4] Bertsimas, D.; Darnell, C.; and Soucy, R. (1999). "Portfolio construction through mixed-integer programming at Grantham, Mayo, Van Otterloo and Company," *Interfaces*, Vol. 29, No. 1 (January-February), pp. 49-66.

[5] Bertsimas, D.;Gamarnik, D.;Tsitsiklis, J. (1999), "Estimation of Time-Varying Parameters in Statistical Models: An Optimization Approach", *Machine Learning*, vol. 35, p 225-245.

[6] Bienstock, D. (1996) "Computational Study on Families of Mixed-Integer Quadratic Programming Problems", *Mathematical Programming*, Vol. 74, 121-140.

[7] Blog, B.; van der Hoek, G.. Rinnooy Kan, A.H.G.;Timmer, G.T. (1983) "Optimal Selection of Small Portfolio", *Management Science*, Vol. 29, No. 7, 792-798.

[8] Bousquet, Olivier; Elisseeff, Andre, (2002) "Stability and Generalization", *Journal of Machine Learning Research*, vol. 2, p. 499-526.

[9] Breiman, L., Friedman, J., Olshen, R., Stone, C. (1984). *Classification and Regression Trees*, Wadsworth International,CA.

[10] Chang, T.J.; Meade, N.; Beasley, J.E.; Sharaiha, Y.; (2000) "Heuristics for Cardinality Constrained Portfolio Optimisation", *Computers and Operations Research*, Vol. 27, 1271-1302.

[11] Cottle, R. W.; Pang, J.; Stone, R. E., (1992) *The Linear Complementarity Problem,* Academic Press, Inc.

[12] ILOG CPLEX 8.1 User Manual, (2002). ILOG CPLEX Division, Incline Village, NV.

[13] Devroye, L.; Gyorki, L.; Lugosi, G. (1991). *A Probabilistic Theory of Pattern Recognition*, Springer, New York.

[14] Devroye, L.; Wagner, T. (1979). "Distribution-free Performance Bounds for the Deleted and Holdout Error Estimates." *IEEE Transactions on Information Theory*, vol. 25, no. 2, p.202-207.

[15] Eaves, B.C. (1971) "The Linear Complementarity Problem", *Management Science*, Vol. 17., No. 9, 612-634.

[16] Faaland, B. (1974) "An Integer Programming Algorithm for Portfolio Selection", *Management Science*, Vol. 20, No. 10, 1376-1384.

[17] Friedman, J. (1991). "Multivariate Adaptive Regression Splines", *Annals of Statistics*, Vol. 19:1, 1-67.

[18] Furnival, G.M.; Wilson Jr., R.W. (1974) "Regression by Leaps and Bounds", *Technometrics*, Vol. 16, No. 4, 499-511.

[19] Geoffrion, A.M. (1967) "Integer Programming by Implicit Enumeration and Balas' Method," *SIAM Review*, Vol. 9., 178-190.

[20] Hastie, T., Tibshirani, R., Friedman, J. (2001). *The Elements of Statistical Learning*, Springer-Verlag, New York.

[21] Hockings, R.R.; Leslie, R.N. (1967) "Selection of the Best Subset in Regression Analysis", *Technometrics*, Vol. 9, No. 4, 531 - 540.

[22] Jacob, N. (1974) "A Limited Diversification Portfolio Selection Model for the Small Investor", *The Journal of Finance*, Vol. 29, Issue 3, 847-856.

[23] Jobst, N.; Horniman, M.; Lucas, C.; Mitra, G. (2001) "Computational Aspects of Alternative Portfolio Selection Models in the Presense of Discrete Asset Choice Constraints", *Quantitative Finance*, Vol. 1 , No. 5, 489-501.

[24] Johnson, R.A., Wichern, D.W. (1998) *Applied Multivariate Statistical Analysis*, 4th edt., Prentice Hall, NJ.

[25] LaMotte, L.R.; Hockings, R.R. (1970) "Computational Efficiency in the Selection of Regression Variables", *Technometrics*,Vol. 12, No. 1 , 83 - 93.

[26] Lemke, C.E. (1965) "Bimatrix Equilibrium Points and Mathematical Programming", *Management Science*, Vol.11, No. 7, 681-689.

[27] Lemke, C.E.; Howson Jr., J.T. (1964) "Equilibrium Points of Bimatrix Games", *Journal of the Society for Industrial and Applied Mathematics*, Vol. 12, No. 2, 413-423.

[28] Mangasarian, O.L. (1965). "Linear and Nonlinear Separation of Patterns by Linear Programming", *Operations Research*, Vol. 13:3, 444-452.

[29] Mansini, R.; Speranza, M.G. (1999) "Heuristic Algorithms for Portfolio Selection Problem with Minimum transaction Lots", *European Journal of Operations Research*, Vol. 114, Issue 1, 219 - 233.

[30] McBride, R. D.; Yormark, J.S. (1980) "An Implicit Enumeration Algorithm for Quadratic Integer Programming", *Management Science*, Vol. 26, No. 3, 282-296.

[31] Miller, A., (1990) *Subset Selection in Regression*, Monographs on Statistics and Applied Probability 40, Chapman and Hall.

[32] Narula, S.; Wellington, J. (1979) "Selection of Variables in Linear Regression Using the Minimum Sum of Weighted Absolute Errors Criterion", *Technometrics*, Vol. 21, No. 3, 299-311.

[33] Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, http://www.nr.com.

[34] Owens Butera, G. (1997), " The Solution of a Class of Limited Diversification Portfolio Selection Problems", Rice University Ph.D. Thesis, CRPC-TR97724-S.

[35] Patel, N.; Subrahmanyam, M. (1982) "A Simple Algorithm for Optimal Portfolio Selection with Fixed Transaction Costs", *Management Science*, Vol. 28, No. 3, 303-314.

[36] Quinlan, R. (1993). *C4.5: Programns for Machine Learning*, Morgan Kaufman, San Mateo.

[37] Rice, J.A. (1995) *Mathematical Statistics and Data Analysis*, 2nd edt., Duxbury Press, CA.

[38] Rifkin, Ryan. (2002) "Everything Old is New Again: A Fresh Look at Historical Approaches in Machine Learning". MIT Doctoral Thesis.

[39] Rousseeuw, P.j., Leroy, A.M. (1987). *Robust regression and outlier detection*, Wiley, New York.

[40] Ryan, T.P. (1997) *Modern Regression Methods*, Wiley Series in Probabilty and Statistics, New York.

[41] Sharpe, W. (1967) "A Linear Programming Algorithm for Mutual Fund Portfolio Selection", *Management Science*, Vol. 13, No. 7, 499-510.

[42] Sharpe, W. (1971) "A Linear Programming Approximation for the General Portfolio Analysis Problem", *Journal of Financial and Quantitative Analysis*, Vol. 6, Issue 5, 1263-1275.

[43] Vapnik, V. (1999) *The Nature of Statistical Learning Theory*, Springer-Verlag, New York.