Integer Optimization Methods for Machine Learning

by

Allison An Chang

Sc.B. Applied Mathematics, Brown University (2007)

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 2012

Author .....
Sloan School of Management
May 18, 2012

Certified by .....
Dimitris Bertsimas
Boeing Leaders for Global Operations Professor
Co-Director, Operations Research Center
Thesis Supervisor

Certified by .....
Cynthia Rudin
Assistant Professor of Statistics
Thesis Supervisor

Accepted by .....
Patrick Jaillet
Dugald C. Jackson Professor
Department of Electrical Engineering and Computer Science
Co-Director, Operations Research Center

Integer Optimization Methods for Machine Learning

by

Allison An Chang

Submitted to the Sloan School of Management
on May 18, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

In this thesis, we propose new mixed integer optimization (MIO) methods to address problems in machine learning. The first part develops methods for supervised bipartite ranking, which arises in prioritization tasks in diverse domains such as information retrieval, recommender systems, natural language processing, bioinformatics, and preventative maintenance. The primary advantage of using MIO for ranking is that it allows for direct optimization of ranking quality measures, as opposed to current state-of-the-art algorithms that use heuristic loss functions. We demonstrate using a number of datasets that our approach can outperform other ranking methods.

The second part of the thesis focuses on reverse-engineering ranking models. This is an application of a more general ranking problem than the bipartite case. Quality rankings affect business for many organizations, and knowing the ranking models would allow these organizations to better understand the standards by which their products are judged and help them to create higher quality products. We introduce an MIO method for reverse-engineering such models and demonstrate its performance in a case-study with real data from a major ratings company. We also devise an approach to find the most cost-effective way to increase the rank of a certain product.

In the final part of the thesis, we develop MIO methods to first generate association rules and then use the rules to build an interpretable classifier in the form of a decision list, which is an ordered list of rules. These are both combinatorially challenging problems because even a small dataset may yield a large number of rules and a small set of rules may correspond to many different orderings. We show how to use MIO to mine useful rules, as well as to construct a classifier from them. We present results in terms of both classification accuracy and interpretability for a variety of datasets.

Thesis Supervisor: Dimitris Bertsimas
Title: Boeing Leaders for Global Operations Professor
Co-Director, Operations Research Center

Thesis Supervisor: Cynthia Rudin
Title: Assistant Professor of Statistics

Acknowledgments

I gratefully acknowledge my advisors Dimitris Bertsimas and Cynthia Rudin for the integral roles they played in the production of this dissertation. From the time I met Dimitris even before joining the ORC, he has challenged me to reach my full potential as a student, and I especially appreciate his kind encouragement and invaluable assistance in helping me determine my post-graduation path. I thank Cynthia for the opportunities she gave me to teach in her classes, for the chance to work closely and interact directly with industry partners, for the sheer number of hours she devoted to reading and editing my papers, and for her caring mentorship.

In addition, I acknowledge the other MIT faculty and staff who have taught and supported me. In particular, I could not have asked for a more patient and generous advisor during my first few semesters of graduate school than Robert Freund, and I am pleased to have his continued guidance as a member of my thesis committee. I am also grateful to Cynthia Barnhart for helping me to start the research that would eventually become my thesis, and to Michael Cavaretta, Robert Thomas, and Gloria Chou from Ford Motor Company for their friendly and helpful collaboration.

My academic experience, though outstanding by itself, has been greatly enriched by all of my friends in the ORC, BDC, and TCC. It has been my privilege to be in the company of such talented and thoughtful individuals. In particular, I thank Andy Sun, Wei Sun, Claudio Telha, Shubham Gupta, Matthew Fontana, Michelle Lustrino, and Theresa Gipson for the time they shared with me. Our discussions have both sustained and entertained me, and through them, I have grown in many ways.

Finally, I am indebted to my family—my parents, grandmother, and sister—for their enduring support. For two decades, they have nurtured my interests, both academic and extracurricular, and provided the means for me to accomplish my goals. To Mom and Dad, who have always given me the freedom to choose my own direction while still offering sound counsel whenever I needed it; and to Emily, who has been a wonderful roommate, advisor, and friend for the past five years, and an inspiring sister for literally our entire lives; a most heartfelt thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we develop new algorithms based on mixed integer optimization (MIO) to address various problems in machine learning. The primary motivation for studying this topic is that the objective functions in many machine learning contexts are, in fact, discrete. Therefore, MIO is a natural framework for modeling and solving these problems. For example, consider the problem of binary classification, in which the goal is to be able to correctly determine in which of two categories an object belongs; the error to minimize is essentially the number of objects that are placed into the wrong category, and this is a discrete quantity—that is, the set of all possible values of the error is a subset of the integers. Even though MIO allows us to capture the exact objectives of interest, conventional machine learning algorithms typically do not use MIO, and instead use heuristics or convex proxies in place of the true objectives in order to solve extremely large problems with minimal computation time. However, not all problems are extremely large, and not all need to be solved quickly. For tasks of moderate size or for which a longer runtime is acceptable, we show that using MIO may have significant advantages.

The machine learning problems we study are *supervised ranking*, *association rule mining*, and *associative classification*. The supervised ranking problem is to find a scoring function such that when we rank a list of objects by their scores according to the scoring function, the ranked list is optimal with respect to some ranking quality measure. In the bipartite case, there are two classes of objects, positive and negative,

and one simple ranking quality measure is the number of positive-negative pairs for which the positive object is ranked higher than the negative object by the scoring function. In a more general setting than the bipartite case, in which perhaps we are given a "true" ranking of objects, we might aim to construct a scoring function that maximizes the number of all pairs for which the higher-ranked object according to our scores is also higher according to the true ranks. In association rule mining, we find correlations between features of objects, which indicate that the presence of a certain set of features implies with high probability the presence of another set of features. Associative classification is the problem of combining these patterns to form models for classification. We describe in detail all of these machine learning problems later in the thesis and show how to solve them using MIO. Below, we first give background on MIO and the progress over the past few decades in our ability to solve MIO problems.

We use MIO to refer specifically to mixed integer linear optimization. The general form of an MIO problem is

$$\begin{aligned} \max \quad & \sum_{j \in I} c_j x_j + \sum_{j \in C} c_j x_j \\ \text{s.t.} \quad & \sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \begin{cases} \geq \\ = \\ \leq \end{cases} b_i, \quad \forall i, \\ & x_j \in \mathbb{Z}_+, \quad \forall j \in I, \\ & x_j \in \mathbb{R}_+, \quad \forall j \in C. \end{aligned} \tag{1.1}$$

In words, the problem is to maximize an objective function subject to a set of equality and inequality constraints, where the variables in I are restricted to be integral and the variables in C can take continuous values. If $I = \emptyset$, then (1.1) is called a linear optimization problem; if $C = \emptyset$, then (1.1) is an integer optimization problem; and if all variables are restricted to be either 0 or 1, then (1.1) is a binary integer optimization problem. If we relax the constraint $x_j \in \mathbb{Z}_+$ to $x_j \in \mathbb{R}_+$ for all $j \in I$, then the resulting problem is called the linear relaxation of (1.1).

MIO is a powerful modeling methodology primarily due to its ability to capture logical relations among various decisions. To illustrate this point, suppose we would like to select 10 players for a sports team out of a pool of 25 people, where we need to obey the following restrictions:

- If player 3 is selected, then player 5 must also be selected,
- If player 6 is not selected, then players 13 and 20 cannot be selected,
- At least one of players 7, 8, and 9 must be selected,
- No more than two of players 10, 11, 12, 15, and 18 can be selected.

We use binary variables x_i that take value 1 if player i is selected and 0 otherwise. The first statement above says that $x_3 = 1$ implies $x_5 = 1$; the second says that $x_6 = 0$ implies both $x_{13} = 0$ and $x_{20} = 0$; and so on. These statements can be captured respectively with:

$$x_5 \geq x_3, \quad x_{13} + x_{20} \leq 2x_6, \quad x_7 + x_8 + x_9 \geq 1, \quad \text{and} \quad x_{10} + x_{11} + x_{12} + x_{15} + x_{18} \leq 2.$$

There may be multiple correct formulations to solve the same problem. For instance, the second statement above is also correctly captured by the pair of constraints

$$x_{13} \leq x_6, \quad \text{and} \quad x_{20} \leq x_6.$$

However, it is important to note that not all valid formulations are equally strong. In fact, the choice of MIO formulation critically influences our ability to solve a problem. Briefly, this is because even though two formulations may correspond to the same discrete set $\mathcal{F}$ of feasible points, the polyhedra formed by the constraints of their linear relaxations are not the same, as shown in Figure 1-1. An MIO formulation is stronger if its linear relaxation corresponds to a smaller feasible set; in particular, it is stronger if it is closer to the convex hull of $\mathcal{F}$ [see Bertsimas and Weismantel, 2005, for details]. This is drastically different from the case of linear optimization, where a good formulation is simply one that has a small number of variables and constraints,

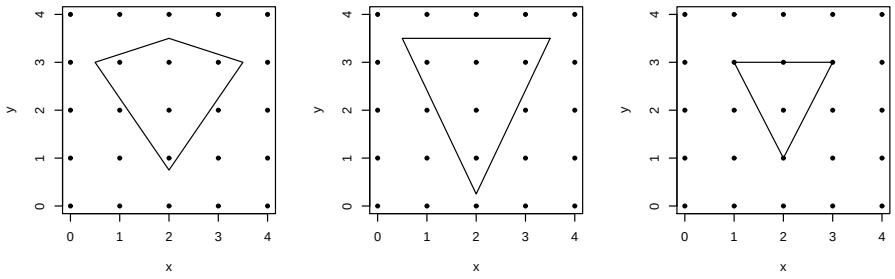


Figure 1-1: Two polyhedra that both contain $\mathcal{F} = \{(2,1),(2,2),(1,3),(2,3),(3,3)\}$ (left and center), and the convex hull of $\mathcal{F}$ (right).

and the choice of formulation is not critical for solving a problem. In contrast, when there are integer variables, it is often an improvement to the formulation to add valid constraints that "cut" the feasible region so that it is closer to the convex hull of $\mathcal{F}$.

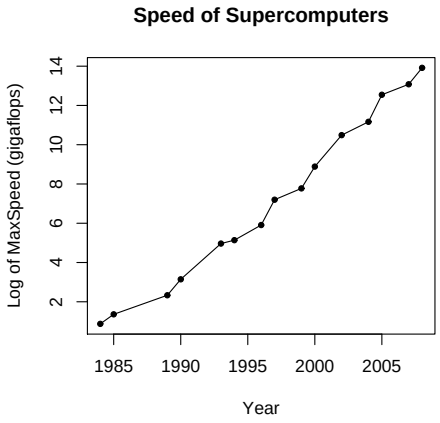
MIO is known to be NP-hard. Nevertheless, our ability to solve MIO problems is constantly improving. From the Gurobi Optimization website:

> The computational progress in linear, quadratic and mixed integer programming over the last twenty years has been nothing short of remarkable, enabling business, scientific and other applications that literally would have been unapproachable just a few short years ago.[1]

In the 1980s, it was difficult to solve a problem with just a hundred integer variables, but now it is possible to solve problems with millions of integer variables [Johnson et al., 2000]. There have been dramatic advancements in both hardware and software. Figure 1-2 shows the exponential increase in the speed of supercomputers developed since the 1980s, measured in billion floating point operations per second (gigaFLOPS).[2] Figure 1-3 shows the time taken to solve the linear optimization problem PILOT on different machines between the late 1980s to 2000.[3] The time decreased by a factor greater than 6000.

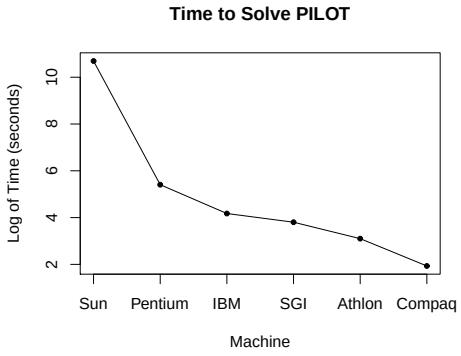
Algorithms for solving MIO problems have also steadily progressed. Techniques employed by modern solvers include branch-and-bound, cutting plane algorithms,

[1] http://www.gurobi.com/html/about.html
[2] http://ecelab.com/supercomputers-list.htm
[3] http://faculty.smu.edu/barr/ip/01ipBixby.PDF



Year	Supercomputer	gFLOPS
1984	M-13	2.4
1985	Cray-2/8	3.9
1989	ETA10-G/8	10.3
1990	NEC SX-3/44R	23.2
1993	Intel Paragon XP/S 140	143.4
1994	Fujitsu Num. Wind Tunnel	170.4
1996	Tsukuba CP-PACS/2048	368.2
1997	Intel ASCI Red/9152	$1.338 \cdot 10^3$
1999	Intel ASCI Red/9632	$2.380 \cdot 10^3$
2000	IBM ASCI White	$7.226 \cdot 10^3$
2002	NEC Earth Simulator	$35.86 \cdot 10^3$
2004	IBM Blue Gene/L	$70.72 \cdot 10^3$
2005	IBM Blue Gene/L	$280.6 \cdot 10^3$
2007	IBM Blue Gene/L	$478.2 \cdot 10^3$
2008	IBM Roadrunner	$1.105 \cdot 10^6$

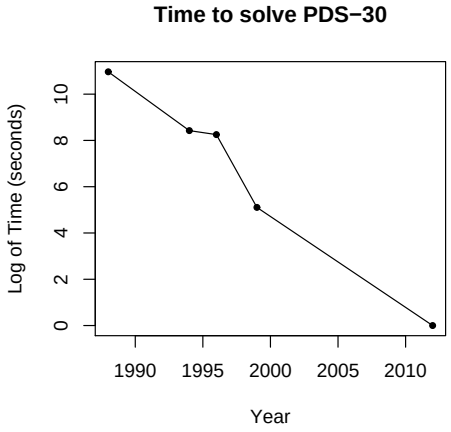
Figure 1-2: Speed of supercomputers.



	Machine	Time (s)
1980s	Sun 3/150	44064.0
.	Pentium PC (60 MHz)	222.6
.	IBM 590 Powerstation	65.0
.	SGI R8000 Power Challenge	44.8
	Athlon (650 MHz)	22.2
2000s	Compaq Alpha	6.9

Figure 1-3: Solver times for the PILOT problem (1141 constraints, 3652 variables).

constraint programming, Lagrangian duality, basis reduction, approximation algorithms, and heuristics [see Johnson et al., 2000, for a description of branch-and-bound and a comprehensive list of references for other integer optimization algorithms]. Solvers for MIO depend heavily on solving linear optimization problems, and the speed with which linear optimization problems can be solved has increased dramatically. For example, Figure 1-4 shows the time taken to solve a problem called PDS-30, which is well-known in the linear optimization community, using different versions of the CPLEX solver on the same machine (296 MHz Sun UltraSparc) [Bixby et al., 2000]. The times improved by a factor of 350 over eleven years, from approximately



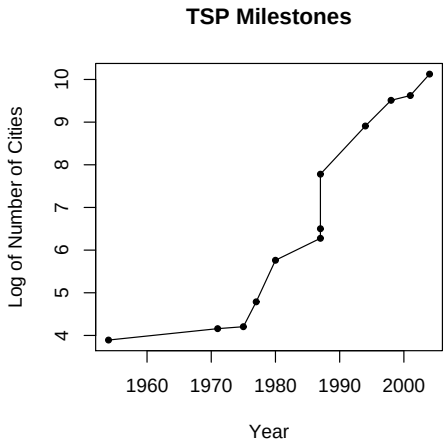
Year	Version	Time (s)
1988	CPLEX 1.0	57840
1994	CPLEX 3.0	4555
1996	CPLEX 5.0	3835
1999	CPLEX 6.5	165

Figure 1-4: Solver times for PDS-30 problem (49944 rows, 177628 columns).

sixteen hours with the first version of CPLEX in 1988, to under three minutes in 1999; extrapolated to the present, the runtime would take on the order of one second. With the advancements in both hardware and software, the size of MIO problems that are solvable is increasing exponentially. For example, consider the traveling salesman problem (TSP), which is the problem of finding the least expensive route for a salesman to visit each of a set of cities exactly once and return to his starting city. Table 1-5 shows the enormous progress made over the years in the size of TSP instances that can be solved, starting from the result of Dantzig et al. [1954] in solving a 49-city TSP.[4]

MIO methods are not commonly used to solve machine learning problems, partly due to a perception starting from the early 1970s that MIO is computationally intractable for most real-world problems [Bertsimas and Shioda, 2007]. However, despite the inherent hardness of MIO, all of the examples above illustrate the rapid and ongoing progress in our ability to solve MIO problems. Such progress encourages us to explore the possibility of using MIO in domains in which it has not been widely applied, including machine learning. Recent work that intersects machine learning and discrete optimization has consisted largely of either using concepts from machine learning to solve discrete optimization problems [e.g., Malagò et al., 2009, Furtlehner and Schoenauer, 2010, Hsu and McIlraith, 2010] or using heuristics from combina-

[4]http://www.tsp.gatech.edu/history/milestone.html



Year	# Cities
1954	49
1971	64
1975	67
1977	120
1980	318
1987	532
1987	666
1987	2,392
1994	7,397
1998	13,509
2001	15,112
2004	24,978

Figure 1-5: Milestones in solving the TSP (number of cities).

torial optimization that exploit problem structure to address machine learning tasks [e.g., Cevher and Krause, 2009, Lin and Bilmes, 2010], instead of using MIO formulations to directly solve machine learning problems. Still, MIO has already been shown to be effective in feature selection, as well as in classification and regression [Nguyen et al., 2009, Bertsimas and Shioda, 2007, Brooks, 2010]. This thesis is among these first efforts in developing MIO methods for machine learning.

In Chapter 2, we derive MIO formulations for supervised bipartite ranking tasks. There are a variety of ranking quality measures that we are interested in optimizing, and our formulations can exactly capture many of them. Chapter 3 discusses an application of supervised ranking in which we use MIO to reverse-engineer quality ranking models. In Chapter 4, we show how to use MIO for both mining association rules and combining the rules into an interpretable and accurate classifier. We conclude in Chapter 5.

Chapter 2

MIO for Supervised Ranking

Supervised ranking techniques can be used for a wide variety of prioritization tasks in many domains, including information retrieval, recommender systems, natural language processing, bioinformatics, and industrial maintenance. The ranking problem is essentially that of ordering a set of entities by their probabilities of possessing a certain attribute. For many applications, improving the quality of a ranked list by even a small percentage has significant consequences. For instance, a more accurate ranking of electrical grid components in New York City, in order of vulnerability to failures or dangerous events, helps prevent power outages as well as serious accidents from fires and explosions [Gross et al., 2009, Rudin et al., 2010]. In the area of drug screening, where developing a new drug costs over $1 billion, the ability to correctly rank the top of a list of millions of compounds according to the chance of clinical success produces significant savings, in terms of both time and money [Agarwal et al., 2010]. For Netflix, the accurate ranking of movies was sufficiently important that the company offered a $1 million prize in a contest to beat the accuracy of its recommendation system [Bennett and Lanning, 2007].[1]

In this chapter, we introduce an MIO approach for supervised ranking. The primary advantage of using MIO for ranking is that it allows for direct optimization of the true objective function rather than approximating with a heuristic choice of loss functions. This means that the objective we optimize with MIO is also the measure we

[1] http://www.netflixprize.com/

use to evaluate ranking quality. Our methods were designed to be able to optimize many common ranking objectives, or *rank statistics*, including the area under the ROC curve (AUC) [Metz, 1978, Bradley, 1997] and the discounted cumulative gain (DCG) measure used in information retrieval [Järvelin and Kekäläinen, 2000]. This work focuses on bipartite ranking problems, which are distinctly different from and more challenging than binary classification. Currently, supervised ranking methods are used almost exclusively for large scale problems that occur in the information retrieval domain [for example, see Xu, 2007, Cao et al., 2007, Matveeva et al., 2006, Lafferty and Zhai, 2001, Li et al., 2007, and the LETOR compilation of works[2]], and there are many works that discuss how to approximately solve extremely large ranking problems quickly [Freund et al., 2003a, Tsochantaridis et al., 2005, Joachims, 2002, Cossock and Zhang, 2006, Burges et al., 2006, Xu et al., 2008, Le and Smola, 2007, Ferri et al., 2002, Ataman et al., 2006]. In order to produce fast solutions, these methods all use heuristic loss functions or other approximations that may be very different from the true objectives.

On the other hand, not all ranking problems are large. Consider, for example, the re-ranking problem [Ji et al., 2006, Collins and Koo, 2003]. In re-ranking, the top N candidates are first generated by a classification model, and then the ranking algorithm is applied only to those top candidates. These problems may be small depending on the size of N, even if the original ranking problem is extremely large. Moreover, there is a growing body of work that addresses supervised ranking in domains where speed is not essential and a better solution is worth the extra computation time. Examples of such supervised ranking tasks include ranking manholes for the purpose of power grid maintenance [Rudin et al., 2010], ranking chemicals for the purpose of drug discovery [Agarwal et al., 2010], and ranking medical symptoms for the purpose of future symptom prediction [McCormick et al., 2011]. In Chapter 3, we use specialized MIO ranking methods to reverse-engineer quality ratings, where the dataset is a decade's worth of ratings data; this problem's size is still able to be handled effectively with MIO.

[2]http://research.microsoft.com/en-us/um/beijing/projects/letor/paper.aspx

Our approach makes the following contributions:

1. **Unification and extension of rank statistics:** We present a method that unifies a large class of rank statistics under the same formulation. This implies that we can use the same optimization method to exactly solve ranking problems, regardless of the specific rank statistic of interest. Further, by taking different cases of our general formulation, we can derive and optimize new rank statistics that are useful for specialized problems.
2. **Guarantee of optimality:** Our method is designed specifically to yield scoring functions with optimal ranking performance, with respect to a given objective function. For tasks for which the choice of algorithm makes little difference on the solution, the MIO method serves as a benchmark and provides a guarantee of optimality that other algorithms do not. Even if an MIO problem is too large to solve to provable optimality, solvers provide a bound on the optimal objective value, which may be a useful measure of closeness to optimality.

This chapter is organized as follows: Section 2.1 sets up our ranking notation and definitions, and Section 2.2 contains our MIO formulations. In particular, we give formulations that optimize two ranking objectives: the AUC, and a more general ranking objective that we call a "rank risk functional." In Section 2.3, we discuss a result showing that in many cases, we can use a nonexact formulation to exactly optimize the rank risk functional, the benefit of the nonexact formulation being that it solves faster. In Section 2.4, we show computational results comparing the performance of the MIO methods to that of several other methods. We give future directions of the work in Section 2.5 and conclude in Section 4.6.

2.1 Supervised Bipartite Ranking

In this section, we establish our notation and propose a new way of representing a general class of rank statistics. We also explain our means of handling ties and describe current approximate methods for supervised bipartite ranking tasks.

2.1.1 Notation

In *supervised bipartite ranking*, the data consist of labeled examples $\{(x_i, y_i)\}_{i=1}^n$, with each example x_i in some space $X \subset \mathcal{R}^d$ and each $y_i \in \{0, 1\}$. The examples with $y_i = 1$ are labeled "positive," and the examples with $y_i = 0$ are labeled "negative." These ranking problems are called "supervised" because the labels are known, and "bipartite" refers to the labels taking on two possible values. There are n_+ positive examples and n_- negative examples, with index sets $S_+ = \{i : y_i = 1\}$ and $S_- = \{k : y_k = 0\}$. To rank the examples, we use a scoring function $f : X \to \mathcal{R}$ to assign them real-valued scores $\{f(x_i)\}_{i=1}^n$. We define *minimum rank* as a function of f by the following formula:

$$\text{minrank}_f(x_i) = \sum_{k=1}^{n} \mathbf{1}_{[f(x_k) < f(x_i)]}, \quad \forall i = 1, \ldots, n.$$

The minimum rank of an example is the number of examples that score strictly below it. Note that examples with equal score are tied in their minimum ranks. We also assign *ranks* to the examples according to the following definition:

Definition 1. *The **rank** of example x_i according to scoring function f, denoted $\text{rank}_f(x_i)$, is a number between 0 and $n-1$ that obeys the following constraints:*

1. *The rank of an example is at least its minimum rank.*
2. *Each possible rank, 0 through $n-1$, may be assigned to only one example (even though multiple examples may all share the same minimum rank).*
3. *If a positive example and a negative example have the same score, then the negative example is assigned a higher rank.*

When there are no ties in score, the rank is equal to the minimum rank. The assignment of ranks to a set of examples is not necessarily unique; if two positive or two negative examples have the same score, then either of them could take the higher rank. Table 2.1 shows a ranked list of labeled examples along with their scores, minimum ranks, and a possible choice of ranks. A *misrank* occurs when a negative

Table 2.1: Demonstration of rank definitions.

Label	+	+	+	−	−	+	−	+	−
Score	6	6	5	4	3	3	2	2	1
MinRank	7	7	6	5	3	3	1	1	0
Rank	8	7	6	5	4	3	2	1	0

example scores equal to or higher than a positive example. We use linear scoring functions $f(x_i) = w^T x_i$, where $w \in \mathcal{R}^d$. Note that we can add features such as x_i^2, $\log(x_i)$, $x_i x_k$, or $\mathbf{1}_{[x_i>10]}$ to incorporate nonlinearity. Our goal is to generate a scoring function f such that the coefficients $w_1, \ldots, w_d$ are optimal with respect to a specified ranking quality measure.

2.1.2 Rank Statistics

There are several rank statistics used to measure ranking quality, the most popular of which is arguably the AUC. Counting ties as misranks, the AUC is defined by:

$$\mathrm{AUC}(f) = \frac{1}{n_+ n_-} \sum_{i \in S_+} \sum_{k \in S_-} \mathbf{1}_{[f(x_k) < f(x_i)]}.$$

There are $n_+ n_-$ positive-negative pairs of examples, or pairs with one positive example and one negative example. Thus, the AUC is simply the fraction of correctly ranked positive-negative pairs. We next introduce a general class of rank statistics:

Definition 2. *Let $a_1 \leq a_2 \leq \cdots \leq a_n$ be non-negative constants. A **rank risk functional** is of the form*

$$\mathrm{RRF}(f) = \sum_{i=1}^{n} y_i \sum_{\ell=1}^{n} \mathbf{1}_{[\mathrm{rank}_f(x_i)=\ell-1]} \cdot a_\ell. \tag{2.1}$$

This class captures a broad collection of rank statistics. The RRF equation coincides with the definition of conditional linear rank statistics [Clemençon and Vayatis, 2008] when there are no ties in score. Special members of this class include:

- $a_\ell = \ell$: Wilcoxon Rank Sum (WRS) – related to the AUC [see Clemençon et al., 2008, Clemençon and Vayatis, 2008].

- $a_\ell = \ell \cdot \mathbf{1}_{[\ell \geq t]}$ for some threshold t: local AUC – concentrates at the top of the list [Clemençon and Vayatis, 2007, 2008, Dodd and Pepe, 2003].
- $a_\ell = \mathbf{1}_{[\ell = n]}$: Winner Takes All (WTA) – concerned only with whether the top example in the list is positively-labeled [Burges et al., 2006].
- $a_\ell = \frac{1}{n-\ell+1}$: Mean Reciprocal Rank (MRR) [Burges et al., 2006].
- $a_\ell = \frac{1}{\log_2(n-\ell+2)}$: Discounted Cumulative Gain (DCG) – popular in information retrieval [Järvelin and Kekäläinen, 2000].
- $a_\ell = \frac{1}{\log_2(n-\ell+2)} \cdot \mathbf{1}_{[\ell \geq t]}$: DCG@N – concentrates at the top of the list [see, for instance, Le and Smola, 2007].
- $a_\ell = \ell^p$ for some $p > 0$: related to the P-Norm Push – concentrates on pushing negatives down from the top of the list [Rudin, 2009].

A different framework that unifies ranking measures is presented in Le and Smola [2007] and Le et al. [2010].

In addition to encompassing conventional rank statistics, (2.1) may be used to define new rank statistics. We introduce the *staircase rank statistic*, which is appropriate for problems in which the user wants to specify priorities between several tiers in a ranked list but does not discriminate within each tier. As a practical illustration of this statistic, consider the task of ranking manholes, or access points to an electric grid, in order of vulnerability, as faced by Rudin et al. [2010]. Suppose there is a repair truck that visits all manholes in the top tier of a ranked list at approximately the same time, and later on, the next tier of manholes all at approximately the same time, and so on. In this case, it does not matter how manholes are ranked *within* a particular tier because they will all be visited at about the same time, but the relative placement of manholes between tiers does matter.

Definition 3. *Let there be $T \leq n$ tiers with given rank thresholds $\{r_t\}_{t=1}^T$, and parameters $\{q_t\}_{t=1}^T$, where $q_t \in \mathcal{R}_+$. Here q_t represents the increase in the objective gained by placing a positive example in tier t rather than in tier $t-1$. The* ***staircase rank***

statistic *is:*

$$\mathrm{RS}_{stair}(f) = \sum_{i=1}^{n} y_i \sum_{t=1}^{T} q_t \mathbf{1}_{[\mathrm{rank}_f(x_i) \geq r_t]}.$$

For instance, suppose $n = 50$ and there are three tiers: top ten, top twenty, and top thirty. Assume that there are no ties. Then $T = 3$, $r_1 = 40$, $r_2 = 30$, and $r_3 = 20$. If $q_1 = 5$, $q_2 = 3$, and $q_3 = 1$, then a positive example x_i adds $9 = 5 + 3 + 1$ to the statistic if it is in the top ten, $4 = 3 + 1$ if it is in the top twenty, 1 if it is in the top thirty, and 0 otherwise. (2.1) represents the staircase rank statistic if we set:

$$a_\ell = \sum_{t=1}^{T} q_t \mathbf{1}_{[\ell - 1 \geq r_t]}.$$

2.1.3 Treatment of Ties

Nonparametric statistics textbooks typically remove tied observations or assign average ranks [Tamhane and Dunlop, 2000, Wackerly et al., 2002], and there has been some research in comparing different ways of handling ties [e.g., Putter, 1955]. However, the treatment of ties in rank is not critical in classical applications of statistics in the sense that there is no unified treatment of ties [Savage, 1957].

On the other hand, handling ties is of central importance when we wish not only to compute rank statistics, but also to *optimize* them. The key is to treat a tie between a positive example and a negative example pessimistically as a misrank. To see why this is essential, suppose tied positive-negative pairs were considered correctly ranked. Then using $w = 0$, there would be no misranks because all positive-negative pairs would be tied at score $f(x_i) = 0$. Having no misranks usually implies a perfect solution, but clearly $w = 0$ is not optimal in any reasonable sense. Thus, in our formulations, a tied positive-negative pair is penalized as a misrank. Our definitions reflect this in two specific places. First, the inequality in our definition of minimum rank is strict ($\sum_{k=1}^{n} \mathbf{1}_{[f(x_k) < f(x_i)]}$ instead of $\sum_{k=1}^{n} \mathbf{1}_{[f(x_k) \leq f(x_i)]}$). Second, if there is a tied positive-negative pair, we always give the negative example the higher rank, according to the third constraint in Definition 1.

Rank statistics typically assume that there are no ties. In the case of no ties, we

ensure that our formulas give the same value as the usual definitions. However, we also handle ties in the pessimistic way described above, so that increasing the number of ties between positive-negative pairs lowers ranking quality, which allows us to avoid trivial solutions when optimizing rank statistics.

2.1.4 Approximate Methods for Ranking

In this section, we contrast the discrete nature of rank statistics with current ranking methods that approximate rank statistics with convex surrogate loss functions. As an illustrative example, suppose that we want to minimize the misranking error, which is equivalent to maximizing the AUC. The misranking error is the fraction of pairs that are misranked:

$$\text{ERR}(f) = \frac{1}{n_+ n_-} \sum_{i \in S_+} \sum_{k \in S_-} \mathbf{1}_{[f(x_i) \leq f(x_k)]} = \frac{1}{n_+ n_-} \sum_{i \in S_+} \sum_{k \in S_-} \mathbf{1}_{[u_{ik} \leq 0]}, \tag{2.2}$$

where $u_{ik} = f(x_i) - f(x_k)$ for $i \in S_+, k \in S_-$. The 0-1 loss $g(u) = \mathbf{1}_{[u \leq 0]}$ is the step function in Figure 2-1. The RankBoost algorithm of Freund et al. [2003a] uses the exponential loss e^{-u} as an upper bound for the 0-1 loss. That is, the loss function for RankBoost is

$$\sum_{i \in S_+} \sum_{k \in S_-} e^{-(f(x_i) - f(x_k))}. \tag{2.3}$$

Support vector machine algorithms [e.g., Joachims, 2002, Herbrich et al., 2000, Shen and Joshi, 2003] use a piecewise-linear function, the hinge loss $g(u) = \max\{0, 1-u\}$, as the upper bound. For example, a possible SVM-style loss function is

$$\sum_{i \in S_+} \sum_{k \in S_-} \max\{0, 1 - (f(x_i) - f(x_k))\}. \tag{2.4}$$

As shown in Figure 2-1, the exponential loss and hinge loss are convex upper bounds for the misranking error. Instead of directly minimizing (2.2), current methods commonly minimize such upper bounds. In Section 2.2, we show how to use MIO to directly optimize the misranking error.

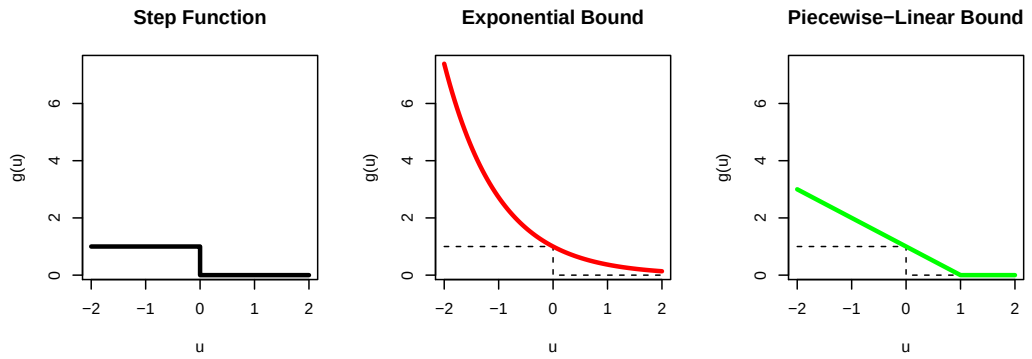


Figure 2-1: 0-1 loss $g(u) = \mathbf{1}_{[u \leq 0]}$ (left), exponential loss $g(u) = e^{-u}$ (center), and hinge loss $g(u) = \max\{0, 1 - u\}$ (right).

There are a variety of other ranking algorithms that similarly minimize convex loss functions, such as RankProp and RankNet [Caruana et al., 1996, Burges et al., 2005]. The P-Norm Push algorithm [Rudin, 2009] generalizes RankBoost by introducing an ℓ_p norm that acts as a soft-max, and is equivalent to RankBoost for $p = 1$; this algorithm minimizes

$$\sum_{k \in S_-} \left(\sum_{i \in S_+} e^{-(f(x_i) - f(x_k))} \right)^p .$$

Bipartite ranking is different from binary classification, that is, a minimizer of the ranking loss is not necessarily a minimizer of the classification loss and vice-versa. Nevertheless, algorithms that produce estimates of $P(y = 1|x)$, such as logistic regression, can plausibly be used for both classification and ranking, though logistic regression minimizes the following, which is not a rank statistic:

$$\sum_{i=1}^{n} \ln \left(1 + e^{-y_i f(x_i)}\right) . \tag{2.5}$$

To use logistic regression for ranking, we would use the probability estimates to rank the examples [e.g., Cooper et al., 1994, Fine et al., 1997, Perlich et al., 2003]. The results of Kotłowski et al. [2011] support the minimization of classification loss functions for ranking. Also see Ertekin and Rudin [2011] for a logistic regression-style ranking algorithm and a comparison of classification versus ranking methods.

2.2 MIO Formulations

In this section, we introduce MIO formulations for maximizing the AUC and RRF from Section 2.1.2. In our experiments, we also use the associated linear relaxations of the MIO formulations, in which the binary variables are allowed to take continuous values in $[0, 1]$. In this case, the objective value is no longer exactly the AUC or RRF, but the solution w is still useful for ranking.

2.2.1 Maximizing AUC

Let $v_i = f(x_i) = w^T x_i$ be the score for instance x_i. For each pair (x_i, x_k) such that $i \in S_+$ and $k \in S_-$, we want the binary variable z_{ik} to keep track of whether x_i is scored higher than x_k. That is, our formulation captures for all $i \in S_+$ and $k \in S_-$:

- If $v_i > v_k$, then $z_{ik} = 1$.
- If $v_i \leq v_k$, then $z_{ik} = 0$.

In what follows, we use (2.6) to refer to the entire MIO formulation and not just the objective; likewise we refer to the other formulations in this thesis by the numbers next to their objective functions. The formulation for maximizing the AUC is:

$$\max_{w,v,z} \quad \sum_{i \in S_+} \sum_{k \in S_-} z_{ik} \tag{2.6}$$

$$\begin{aligned} \text{s.t.} \quad & z_{ik} \leq v_i - v_k + 1 - \varepsilon, \quad \forall i \in S_+, k \in S_-, && (2.7)\\ & v_i = w^T x_i, \quad \forall i \in S_+, \\ & v_k = w^T x_k, \quad \forall k \in S_-, \\ & -1 \leq w_j \leq 1, \quad \forall j = 1, \ldots, d, \\ & z_{ik} \in \{0, 1\}, \quad \forall i \in S_+, k \in S_-, \end{aligned}$$

where $\varepsilon > 0$ is a small user-specified constant. The main constraint is (2.7). If $v_i - v_k \geq \varepsilon$, then the right-hand-side of (2.7) is at least 1, so the solver assigns $z_{ik} = 1$ because we are maximizing z_{ik}. On the other hand, if $v_i - v_k < \varepsilon$, then the right-hand-side is strictly less than 1, which forces $z_{ik} = 0$.

The purpose of constraining each w_j in the interval $[-1, 1]$ is to bound the feasible region over which the solver must search for the optimal solution, since a smaller solution space generally implies a shorter runtime. The purpose of using a small positive ε in the formulation is to force the strict inequality $v_i > v_k$. If we removed ε, then the constraints would be satisfied by $w_j = 0$ for all j and $z_{ik} = 1$ for all i, k, corresponding to the trivial solution discussed in Section 2.1.3. To prohibit this solution from being optimal, we need $\varepsilon > 0$. Note that the bounds on w_j and choice of ε are not completely independent, in the following sense: if we modified the bounds to be $-c \leq w_j \leq c$ for some $c > 0$, then we would have an essentially equivalent formulation as the original by replacing ε with $c\varepsilon$.

In order for the formulation to be exact, we must have that for all positive-negative example pairs in which the positive example scores higher, the difference between the two scores is at least ε, that is

$$\delta = \min_{\{i \in S_+, k \in S_- : v_i > v_k\}} (v_i - v_k) \geq \varepsilon.$$

This is straightforward to verify after solving (2.6) as we can simply take the optimal w and compute the AUC to check that it matches the objective value of (2.6). Larger values of ε may lead to suboptimal solutions. For example, there may be two feasible solutions w_1 and w_2, where $\delta = 0.003$ for w_1 and $\delta = 0.0003$ for w_2. It is possible that for $\varepsilon = 0.001$, w_1 would maximize (2.6), but that by lowering ε to 0.0001, w_2 would be optimal instead. In Section 2.4, we show the effect of varying ε.

2.2.2 Maximizing RRF

We aim now to maximize the general rank risk functional from Definition 2. We want the binary variable $t_{i\ell}$ to be 1 if $\text{rank}_f(x_i) \geq \ell - 1$ and 0 otherwise. Then $\text{rank}_f(x_i) = \ell - 1$ if and only if $t_{i\ell} - t_{i,\ell+1} = 1$ for $\ell = 1, \ldots, n-1$ and $\text{rank}_f(x_i) = n - 1$ if and only if $t_{in} = 1$. Thus, the objective to maximize is:

$$\sum_{i=1}^{n} y_i \sum_{\ell=1}^{n} a_\ell (t_{i\ell} - t_{i,\ell+1}),\ t_{i,n+1} = 0,\ \text{or equivalently}\ \sum_{i=1}^{n} y_i \sum_{\ell=1}^{n} (a_\ell - a_{\ell-1}) t_{i\ell},\ a_0 = 0.$$

Since we have $t_{i1} = 1$ for all i, the cost function is:

$$\sum_{i \in S_+} \left(\sum_{\ell=2}^{n} (a_\ell - a_{\ell-1}) t_{i\ell} + a_1 \right) = |S_+| a_1 + \sum_{i \in S_+} \sum_{\ell=2}^{n} (a_\ell - a_{\ell-1}) t_{i\ell}.$$

For all $i \in S_+$ and $\ell \geq 2$, the formulation sets $t_{i\ell} = 1$ if feasible because $a_\ell - a_{\ell-1} \geq 0$. Let $\tilde{a}_\ell = a_\ell - a_{\ell-1}$ and $S_2 = \{\ell \geq 2 : \tilde{a} > 0\}$. We further simplify the objective to be:

$$\sum_{i \in S_+} \sum_{\ell \in S_2} \tilde{a}_l t_{i\ell}. \tag{2.8}$$

We present two formulations to address the problem of maximizing the RRF (2.1). In the first, we exactly maximize (2.1), with the simplification in (2.8). We define variables $r_i \in [0, n-1]$ to represent the rank of each example x_i. As before, we use linear scoring functions, so the score of instance x_i is $w^T x_i$. The MIO formulation is:

$$\begin{aligned}
\max_{w,z,t,r} \quad & \sum_{i \in S_+} \sum_{\ell \in S_2} \tilde{a}_l t_{i\ell} && (2.9)\\
\text{s.t.} \quad & z_{ik} \leq w^T (x_i - x_k) + 1 - \varepsilon, \quad \forall i, k = 1, \ldots, n, && (2.10)\\
& z_{ik} \geq w^T (x_i - x_k), \quad \forall i, k = 1, \ldots, n, && (2.11)\\
& r_i - r_k \geq 1 + n(z_{ik} - 1), \quad \forall i, k = 1, \ldots, n, && (2.12)\\
& r_k - r_i \geq 1 - n z_{ik}, \quad \forall i \in S_+, k \in S_-, && (2.13)\\
& r_k - r_i \geq 1 - n z_{ik}, \quad \forall i, k \in S_+, i < k, && (2.14)\\
& r_k - r_i \geq 1 - n z_{ik}, \quad \forall i, k \in S_-, i < k, && (2.15)\\
& t_{i\ell} \leq \frac{r_i}{\ell - 1}, \quad \forall i \in S_+, \ell \in S_2, && (2.16)\\
& -1 \leq w_j \leq 1, \quad \forall j = 1, \ldots, d,\\
& 0 \leq r_i \leq n - 1, \quad \forall i = 1, \ldots, n,\\
& z_{ik} \in \{0, 1\}, \quad \forall i, k = 1, \ldots, n,\\
& t_{i\ell} \in \{0, 1\}, \quad \forall i \in S_+, \ell \in S_2.
\end{aligned}$$

Constraint (2.10) implies $z_{ik} = 0$ if $w^T x_i - w^T x_k < \varepsilon$, and (2.11) implies $z_{ik} = 1$ if

$w^T x_i > w^T x_k$. Note that this means a feasible solution cannot have the difference between two scores strictly between 0 and ε. Constraint (2.12) specifies that for any pair (x_i, x_k), $r_i \geq r_k + 1$ if $z_{ik} = 1$, that is, if $w^T x_i - w^T x_k \geq \varepsilon$. This constraint does not handle ties in scores, so we need the following: (2.13) implies that for a tied positive-negative pair, the negative example has higher rank; and (2.14) and (2.15) imply that for positive-positive pairs and negative-negative pairs with tied scores, the example with a higher index is (arbitrarily) assigned the higher rank. Constraint (2.16) sets $t_{i\ell} = 1$ when $r_i \geq \ell - 1$.

For an alternative formulation to (2.9), consider the quantity

$$\mathrm{RRF}_{\min}(f) = \sum_{i=1}^{n} y_i \sum_{\ell=1}^{n} \mathbf{1}_{[\mathrm{minrank}_f(x_i)=\ell-1]} \cdot a_\ell, \tag{2.17}$$

which is similar to (2.1) except it uses minimum ranks instead of ranks. We show in Section 2.3 that in many cases, a maximizer of (2.17) also maximizes (2.1), the advantage being that capturing minimum ranks with MIO requires fewer variables and constraints than capturing ranks. The following MIO formulation maximizes (2.17):

$$\begin{aligned}
\max_{w,z,t} \quad & \sum_{i\in S_+} \sum_{\ell \in S_2} \tilde{a}_l t_{i\ell} && (2.18)\\
\text{s.t.} \quad & z_{ik} \leq w^T(x_i - x_k) + 1 - \varepsilon, \quad \forall i \in S_+, k = 1, \ldots, n, && (2.19)\\
& t_{i\ell} \leq \frac{1}{\ell - 1} \sum_{k=1}^{n} z_{ik}, \quad \forall i \in S_+, \ell \in S_2, && (2.20)\\
& z_{ik} + z_{ki} = \mathbf{1}_{[x_i \neq x_k]}, \quad \forall i, k \in S_+, && (2.21)\\
& t_{i\ell} \geq t_{i,\ell+1}, \quad \forall i \in S_+, \ell \in S_2 \setminus \max(S_2), && (2.22)\\
& \sum_{i\in S_+} \sum_{\ell \in S_2} \tilde{a}_l t_{i\ell} \leq \sum_{\ell=1}^{n} a_\ell, && (2.23)\\
& z_{ik} = 0, \quad \forall i \in S_+, k = 1, \ldots, n, x_i = x_k, && (2.24)\\
& -1 \leq w_j \leq 1, \quad \forall j = 1, \ldots, d,\\
& t_{i\ell}, z_{ik} \in \{0, 1\}, \quad \forall i \in S_+, \ell \in S_2, k = 1, \ldots, n.
\end{aligned}$$

The minimum rank is $\sum_{k=1}^{n} z_{ik}$. Constraints (2.19) and (2.20) are similar to (2.10) and (2.16). Constraints (2.21) through (2.24) are not necessary, but they are intended to strengthen the linear relaxation and thus speed up computation. Note that we do not need (2.11) since here, maximizing the $t_{i\ell}$ directly implies maximizing the z_{ik}.

In Section 2.3, we discuss the conditions under which (2.9) and (2.18) should be used. Formulation (2.9) has $d + n^2 + n_+|S_2| + n$ variables, corresponding to w, z, t, and r respectively. Formulation (2.18) has $d + n_+ n + n_+|S_2|$ variables, corresponding to w, z, and t respectively. Thus (2.9) has an additional $n_- \cdot n + n$ variables compared to (2.18), which can be a significant difference when the negative class is large.

2.2.3 Alternative Formulations

In Sections 2.2.1 and 2.2.2, we presented formulations that we found worked well empirically for the AUC and RRF problems. Here we show alternative formulations to illustrate that there may be multiple correct formulations, as discussed in Chapter 1.

One alternative formulation for the AUC problem is:

$$\begin{aligned} \max_{w,z} \quad & \sum_{i \in S_+} \sum_{k \in S_-} z_{ik} \\ \text{s.t.} \quad & M(1 - z_{ik}) \geq w^T(x_k - x_i) + \varepsilon, \quad \forall i \in S_+, k \in S_-, \\ & z_{ik} \in \{0, 1\}, \quad \forall i \in S_+, k \in S_-, \end{aligned}$$

where M is a large constant. In MIO, this type of formulation is known as a big-M formulation. If $w^T(x_k - x_i) > -\varepsilon$, or $w^T(x_i - x_k) < \varepsilon$, then the first constraint would force $z_{ik} = 0$. If $w^T(x_i - x_k) \geq \varepsilon$, then z_{ik} would be 1 because we are maximizing. Thus, this is also an exact formulation for the AUC problem. However, this formulation is not as strong as (2.6) because the large coefficients tend to cause the linear relaxation to be far from the convex hull of integer feasible points.

It is possible to formulate the RRF problem using *special ordered set* (SOS) constraints [Beale and Tomlin, 1970]. SOS constraints are designed to improve the efficiency of the branch-and-bound process. There are two types: SOS1 constraints say

that at most one variable in an ordered set may be nonzero; SOS2 constraints say that at most two variables in an ordered set may be nonzero, and that if there are two nonzero variables, then they must be consecutive within the set.

The constraint we aim to replace in (2.18) is

$$t_{i\ell} \leq \frac{1}{\ell - 1} \sum_{k=1}^{n} z_{ik}, \quad \forall i \in S_+, \ell \in S_2.$$

Since we are maximizing the $t_{i\ell}$, this constraint captures the condition $t_{i\ell} = 1$ if and only if $z_{ik} \geq \ell - 1$. To capture the same relation using an SOS2 constraint, let

$$\begin{aligned} s_{i\ell} &= 1 - t_{i\ell}, \\ h_{i\ell} &= h^+_{i\ell} - h^-_{i\ell}, \\ h_{i\ell} &= \frac{1}{\ell - 1} \sum_{k=1}^{n} z_{ik} - 1, \\ h^+_{i\ell}, h^-_{i\ell} &\geq 0. \end{aligned}$$

If $\sum_{k=1}^{n} z_{ik} \geq \ell - 1$, then $h_{i\ell} \geq 0$. If $\sum_{k=1}^{n} z_{ik} < \ell - 1$, then $h_{i\ell} < 0$. Consider

$$h^+_{i\ell} + 2t_{i\ell} + 3s_{i\ell} + 4h^-_{i\ell} = \text{SOS2}, \quad \forall i \in S_+, \ell \in S_2,$$

which states that within the ordered set $\{h^+_{i\ell}, t_{i\ell}, s_{i\ell}, h^-_{i\ell}\}$, at most two variables may be nonzero, and that if two are nonzero, then they must be consecutive. Thus, $h^+_{i\ell}$ and $h^-_{i\ell}$ cannot both be nonzero, so $h^+_{i\ell} = \max\{0, h_{i\ell}\}$ and $h^-_{i\ell} = \max\{0, -h_{i\ell}\}$. If $\sum_{k=1}^{n} z_{ik} \geq \ell - 1$, then $h_{i\ell} \geq 0$, which implies $h^+_{i\ell} \geq 0$, so $t_{i\ell} = 1$. If $\sum_{k=1}^{n} z_{ik} < \ell - 1$, then $h_{i\ell} < 0$, which implies $h^-_{i\ell} > 0$, so $s_{i\ell} = 1$ or $t_{i\ell} = 0$. We can also add

$$z_{ik} + 2z_{ki} = \text{SOS1}, \quad \forall i, k \in S_+, i < k,$$

which is an SOS1 constraint that says that at most one of z_{ik} and z_{ki} can be nonzero for all positive-positive pairs. This set of constraints is not necessary but strengthens the linear relaxation.

The MIO minimum rank formulation with SOS constraints is:

$$
\begin{aligned}
\max_{w,z,t,s,h^+,h^-} \quad & \sum_{i\in S_+}\sum_{\ell\in S_2} \tilde{a}_l t_{i\ell} \\
\text{s.t.} \quad & z_{ik} \le w^T(x_i - x_k) + 1 - \varepsilon, \quad \forall i \in S_+, k = 1,\dots,n, \\
& h^+_{i\ell} - h^-_{i\ell} = \frac{1}{\ell - 1}\sum_{k=1}^{n} z_{ik} - 1, \quad \forall i \in S_+, \ell \in S_2, \\
& s_{i\ell} = 1 - t_{i\ell}, \quad \forall i \in S_+, \ell \in S_2, \\
& h^+_{i\ell} + 2t_{i\ell} + 3s_{i\ell} + 4h^-_{i\ell} = \text{SOS2}, \quad \forall i \in S_+, \ell \in S_2, \\
& z_{ik} + 2z_{ki} = \text{SOS1}, \quad \forall (i,k) \in S_+, i < k, \\
& -1 \le w_j \le 1, \quad \forall j = 1,\dots,d, \\
& h^+_{i\ell}, h^-_{i\ell} \ge 0, \quad \forall i \in S_+, \ell \in S_2, \\
& t_{i\ell}, z_{ik} \in \{0,1\}, \quad \forall i \in S_+, \ell \in S_2, k = 1,\dots,n.
\end{aligned}
$$

Our preliminary tests for this formulation suggest that using SOS constraints shortens runtimes on smaller problems, but that for larger problems, the additional variables required for the SOS formulation take too much memory for the problem to be solved on most computers; we have found that the formulations in Sections 2.2.1 and 2.2.2 are substantially more practical.

2.2.4 Generalization Beyond the Bipartite Case

So far we have discussed only the bipartite case. In fact it is possible to extend the MIO methods to the general case of pairwise preferences [see, for example, Freund et al., 2003a]. Let the preference function $\pi(x_i, x_k) = \pi_{ik}$ capture the true ranking of x_i relative to x_k for each pair of examples (x_i, x_k). That is, let

$$
\pi_{ik} = \begin{cases} 1, & \text{if } x_i \text{ is ranked strictly higher than } x_k, \\ 0, & \text{otherwise.} \end{cases}
$$

Also let

$$\Pi = \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik}.$$

We want to find a scoring function that reproduces the rankings as close as possible to the true rankings. This is the setting of the supervised ranking task in Chapter 3 of this thesis. Consider the rank statistic

$$\text{AUC}_\pi(f) = \frac{1}{\Pi} \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik} \mathbf{1}_{[f(x_i) > f(x_k)]}.$$

This statistic is related to the disagreement measure introduced by Freund et al. [2003a], as well as Kendall's τ coefficient [Kendall, 1938]. The highest possible value of $\text{AUC}_\pi(f)$ is 1. We achieve this value if our scoring function f satisfies $f(x_i) > f(x_k)$ for all pairs (x_i, x_k) such that $\pi_{ik} = 1$. We can use the following MIO formulation to maximize AUC_π:

$$\begin{aligned}
\max_{w,z} \quad & \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik} z_{ik} \\
\text{s.t.} \quad & z_{ik} \leq w^T (x_i - x_k) + 1 - \varepsilon, \quad \forall i, k = 1, \ldots, n, \\
& -1 \leq w_j \leq 1, \quad \forall j = 1, \ldots, d, \\
& z_{ik} \in \{0, 1\}, \quad \forall i, k = 1, \ldots, n.
\end{aligned}$$

The AUC_π statistic is quite general. For example, it encompasses the case of k-partite or multipartite ranking [Rajaram and Agarwal, 2005, Fürnkranz et al., 2009], which is similar to ordinal regression [Herbrich et al., 2000]. In particular, suppose that there are C classes and that we would like Class 1 to be ranked above Class 2, Class 2 above Class 3, and so on. Then denoting the class of example x_i by $\text{Class}(x_i)$, we would set

$$\pi_{ik} = \begin{cases} 1, & \text{if } \text{Class}(x_i) > \text{Class}(x_k), \\ 0, & \text{otherwise.} \end{cases}$$

If $C = 2$, then this formulation simply maximizes the WRS statistic, with the positive class as Class 1 and the negative class as Class 2.

Table 2.2: Pathological case.

Label	+	+	−	+	+	+	+
Feature	3	3	2	1	1	1	1
MinRank $w = 1$	5	5	4	0	0	0	0
Rank $w = 1$	6	5	4	3	2	1	0
MinRank $w = -1$	0	0	2	3	3	3	3
Rank $w = -1$	0	1	2	3	4	5	6

Table 2.3: Local AUC ($\sum_{i=1}^{n} y_i \sum_{\ell=1}^{n} \mathbf{1}_{[r_f(x_i)=\ell-1]} \cdot \ell \cdot \mathbf{1}_{[\ell \geq t]}$) with $t = 5$ for ranked examples in Table 2.2, defined using $r_f = \text{minrank}_f$ (left) and $r_f = \text{rank}_f$ (right).

	Local AUC with minranks	Local AUC with ranks
$w = 1$	6+6=12	7+6=13
$w = -1$	0	7+6+5=18

2.3 Rank versus Minimum Rank

The minimum rank formulation (2.18) from Section 2.2.2 does not work for certain pathological cases, namely those for which the same examples appear many times in the data. For instance, suppose there are seven examples, each with just a single feature, as shown in Table 2.2. If the scoring function is $f(x) = wx$, where $w \in \mathcal{R}$ since there is only one feature, then there are two solutions that are unique up to a constant positive factor: $w = 1$ and $w = -1$. Let the objective function be the local AUC from Section 2.1.2 with $t = 5$. Table 2.3 shows calculations for the local AUC when it is defined using rank and minimum rank, as in (2.1) and (2.17) respectively. If we define the local AUC using minimum rank, then it is 12 for $w = 1$ and 0 for $w = -1$. However, $w = -1$ is intuitively the better solution because it puts more positive examples at the top of the list. We avoid this contradiction if we use rank to define the local AUC. That is, when we use rank instead of minimum rank, the local AUC is higher for $w = -1$ than for $w = 1$, which agrees with our intuition.

Thus, for such pathological cases, we should use the rank formulation (2.9). However, (2.9) has more variables than (2.18) and empirically has been difficult to solve except for small problems. We show that in many cases, solving (2.18) also solves (2.9), which allows us to scale up the size of problems that we can handle. We use $\text{RRF}(f)$ and $\text{RRF}_{\min}(f)$ to denote the objectives of (2.1) and (2.17) respectively.

2.3.1 Using Minimum Ranks for Distinct Examples

We want to find $f^* \in \text{argmax}_{\{f \text{ linear}\}}\text{RRF}(f)$, where $\{f \text{ linear}\}$ is the set of linear scoring functions, but since it is difficult to solve (2.9), an alternative is to find $f^* \in \text{argmax}_{\{f \text{ linear}\}}\text{RRF}_{\text{min}}(f)$ using (2.18). The main result is stated in Theorem 1.

Theorem 1. *If the examples are distinct, that is, $x_i \neq x_k$ for $i, k \in \{1, \ldots, n\}$, $i \neq k$, and $f^* \in \text{argmax}_{\{f \text{ linear}\}}\text{RRF}_{\text{min}}(f)$, then*

$$f^* \in \text{argmax}_{\{f \text{ linear}\}}\text{RRF}(f).$$

The proof is presented in two steps. The first step, given in Lemma 2, shows that a maximizer for (2.17) also maximizes (2.1) if there is a maximizer $\bar{f}$ of (2.1) such that there are no ties in score, that is, $\bar{f}(x_i) \neq \bar{f}(x_k)$ for all $i \neq k$. The second step, given in Lemma 3, shows that for linear scoring functions, this condition is satisfied when the examples are distinct, meaning $x_i \neq x_k$ for all $i \neq k$. Note that since $\text{RRF}_{\text{min}}(f)$ and $\text{RRF}(f)$ take a discrete set of values, bounded between 0 and $\sum_{\ell=1}^{n} a_\ell$, maximizers for both functions always exist. The following lemma establishes basic facts about the two objectives:

Lemma 1. *The following relationships always hold.*

a. For any f, $\text{RRF}_{\text{min}}(f) \leq \text{RRF}(f)$.

b. For any f such that there are no ties in score, $\text{RRF}_{\text{min}}(f) = \text{RRF}(f)$.

Proof. Fix a scoring function f.

a. The first part of Definition 1 says that $\text{minrank}_f(x_i) \leq \text{rank}_f(x_i)$ for all $i = 1, \ldots, n$. Since the a_ℓ are non-decreasing with ℓ,

$$\begin{aligned} \sum_{\ell=1}^{n} \mathbf{1}_{[\text{minrank}_f(x_i)=\ell-1]} \cdot a_\ell &= a_{(\text{minrank}_f(x_i)+1)} \\ &\leq a_{(\text{rank}_f(x_i)+1)} = \sum_{\ell=1}^{n} \mathbf{1}_{[\text{rank}_f(x_i)=\ell-1]} \cdot a_\ell \quad \forall i. \end{aligned} \tag{2.25}$$

Combining this result with (2.1) and (2.17), we have $\text{RRF}_{\text{min}}(f) \leq \text{RRF}(f)$.

b. If there are no ties in score, then it is clear from Definition 1 that we have $\text{minrank}_f(x_i) = \text{rank}_f(x_i)$ for all i. Thus, the inequality in (2.25) becomes an equality, and $\text{RRF}_{\min}(f) = \text{RRF}(f)$.

□

In the following, the argmax is with respect to a particular set of scoring functions.

Lemma 2. *Let* $\bar{f} \in \text{argmax}_f \text{RRF}(f)$ *such that there are no ties in score, that is,* $\bar{f}(x_i) \neq \bar{f}(x_k)$ *for all* $i \neq k$. *If* $f^* \in \text{argmax}_f \text{RRF}_{\min}(f)$, *then*

$$f^* \in \text{argmax}_f \text{RRF}(f).$$

Proof. Assume there exists $\bar{f} \in \text{argmax}_f \text{RRF}(f)$ such that there are no ties in score. Let f^* maximize $\text{RRF}_{\min}(f)$, which implies $\text{RRF}_{\min}(f^*) \geq \text{RRF}_{\min}(\bar{f})$. We know $\text{RRF}_{\min}(\bar{f}) = \text{RRF}(\bar{f})$ by Lemma 1b.

Suppose f^* does not maximize $\text{RRF}(f)$, so $\text{RRF}(\bar{f}) > \text{RRF}(f^*)$. Then

$$\text{RRF}_{\min}(f^*) \geq \text{RRF}_{\min}(\bar{f}) = \text{RRF}(\bar{f}) > \text{RRF}(f^*).$$

This contradicts Lemma 1a, so $f^* \in \text{argmax}_f \text{RRF}(f)$. □

It is interesting to note that under the condition of Lemma 2, namely that $\bar{f}$ maximizes $\text{RRF}(f)$ without any ties in score, we can also show $\bar{f} \in \text{argmax}_f \text{RRF}_{\min}(f)$. By both parts of Lemma 1, we have that for any f,

$$\text{RRF}_{\min}(f) \leq \text{RRF}(f) \leq \text{RRF}(\bar{f}) = \text{RRF}_{\min}(\bar{f}).$$

Thus, any f that maximizes $\text{RRF}(f)$ without any ties in score maximizes $\text{RRF}_{\min}(f)$.

The results above did not use the structure of our scoring functions, in particular the linear form $f(x) = w^T x$. It used only the properties in Lemma 1. In what follows, we incorporate the additional structure, which allows us to show that if the examples are distinct—which happens with probability one if they are drawn from a continuous

distribution—and if w yields a scoring function with ties, then we can find a corrected $\hat{w}$ that has no ties and achieves at least as high a value for $\mathrm{RRF}(f)$. This implies that there exists a maximizer of $\mathrm{RRF}(f)$ such that there are no ties in score, which satisfies the condition of Lemma 2. From this point on, assume we are considering only linear scoring functions; for example, $\mathrm{argmax}_f \mathrm{RRF}(f)$ means $\mathrm{argmax}_{\{f \text{ linear}\}} \mathrm{RRF}(f)$.

Recall that the number of features is d, that is $x_i \in \mathcal{R}^d$. We also use the fact that for distinct examples $\{x_i\}_{i=1}^n$, vectors that are orthogonal to any of the vectors $x_i - x_k$, $i \neq k$, are in a set of measure zero. Thus, a vector u such that $u^T(x_i - x_k) \neq 0$ for all $i \neq k$ always exists.

Lemma 3. *Assume the data lie in a bounded box, that is, there exists M such that $x_{ij} \in [-M, M]$ for all i, j. Also assume the examples are distinct, so that for any x_i and x_k, $i \neq k$, the vector $x_i - x_k \in \mathcal{R}^d$ has at least one nonzero entry. Consider $\bar{f} \in \mathrm{argmax}_f \mathrm{RRF}(f)$ that yields a scoring function $\bar{f}(x) = \bar{w}^T x$ with ties. Construct $\hat{w}$ as follows:*

$$\hat{w} = \bar{w} + \gamma u,$$

where u is a vector in $\mathcal{R}^d$ with $\|u\|_2 = 1$ and $u^T(x_i - x_k) \neq 0$ for all $i \neq k$, and γ is a fixed real number such that $0 < \gamma < \frac{\delta}{2M\sqrt{d}}$, with

$$\delta = \min_{\{i,k:\bar{f}(x_i) > \bar{f}(x_k)\}} \left(\bar{f}(x_i) - \bar{f}(x_k)\right).$$

Then $\hat{f}(x) = \hat{w}^T x$ preserves all pairwise orderings of $\bar{f}$ but does not have ties. That is, $\bar{f}(x_i) > \bar{f}(x_k) \Rightarrow \hat{f}(x_i) > \hat{f}(x_k)$ and $\hat{f}(x_i) \neq \hat{f}(x_k)$ for all $i \neq k$.

Proof. First we show that $\hat{f}$ preserves all pairwise orderings for examples that are not tied. Consider any pairwise ordering by choosing two examples x_1 and x_2 such that $\bar{f}(x_1) > \bar{f}(x_2)$. Now

$$\begin{aligned}\hat{f}(x_1) - \hat{f}(x_2) &= (\bar{w} + \gamma u)^T(x_1 - x_2) = \bar{w}^T(x_1 - x_2) + \gamma u^T(x_1 - x_2) \\ &= \bar{f}(x_1) - \bar{f}(x_2) + \gamma u^T(x_1 - x_2) \geq \delta + \gamma u^T(x_1 - x_2). \qquad (2.26)\end{aligned}$$

We know that:

$$\|x_1 - x_2\|_2 = \left(\sum_{j=1}^{d}(x_{1j} - x_{2j})^2\right)^{1/2} \leq \left(\sum_{j=1}^{d}(2M)^2\right)^{1/2} = 2M\sqrt{d}. \tag{2.27}$$

Using the Cauchy-Schwarz inequality and then using (2.27), the fact that $||u||_2 = 1$, and the bound on γ from the statement of the lemma:

$$\left|\gamma u^T(x_1 - x_2)\right| \leq \gamma\|u\|_2\|x_1 - x_2\|_2 \leq \gamma \cdot 2M\sqrt{d} < \frac{\delta}{2M\sqrt{d}} \cdot 2M\sqrt{d} = \delta.$$

This implies

$$\gamma u^T(x_1 - x_2) > -\delta. \tag{2.28}$$

Combining (2.26) with (2.28),

$$\hat{f}(x_1) - \hat{f}(x_2) \geq \delta + \gamma u^T(x_1 - x_2) > \delta - \delta = 0.$$

Thus, all pairwise orderings are preserved, that is, $\bar{f}(x_1) > \bar{f}(x_2) \longrightarrow \hat{f}(x_1) > \hat{f}(x_2)$. Next we prove that $\hat{w}$ yields a scoring function with no ties. Take x_1 and x_2 such that their scores according to $\bar{f}$ are tied: $\bar{f}(x_1) = \bar{f}(x_2)$. Then,

$$\begin{aligned}
|\hat{f}(x_1) - \hat{f}(x_2)| &= \left|(\bar{w} + \gamma u)^T(x_1 - x_2)\right| \\
&= \left|\bar{w}^T(x_1 - x_2) + \gamma u^T(x_1 - x_2)\right| \\
&= \left|0 + \gamma u^T(x_1 - x_2)\right| \\
&= |\gamma|\left|u^T(x_1 - x_2)\right| > 0,
\end{aligned}$$

where the last inequality follows since $\gamma > 0$ and $u^T(x_1 - x_2) \neq 0$ by assumption. This implies that the corrected scores are not tied. □

Now we prove Theorem 1, restated here: If the examples are distinct, that is, $x_i \neq x_k$ for $i, k \in \{1, \ldots, n\}$, $i \neq k$, and $f^* \in \text{argmax}_{\{f \text{ linear}\}}\text{RRF}_{\min}(f)$, then

$$f^* \in \text{argmax}_{\{f \text{ linear}\}}\text{RRF}(f).$$

Proof. We need only to satisfy the condition of Lemma 2, which says that (2.9) has a maximizer with no ties. Let $\bar{f} \in \text{argmax}_{\{f \text{ linear}\}}\text{RRF}(f)$. If $\bar{f}(x) = \bar{w}^T x$ is a scoring function with no ties, then we are done. Otherwise, the vector $\hat{w} = \bar{w} + \gamma u$ constructed according to Lemma 3 produces a scoring function with no ties. It only remains to show that $\hat{f}(x) = \hat{w}^T x$ is also optimal. We prove here that $\text{RRF}(\hat{f}) \geq \text{RRF}(\bar{f})$, which means $\text{RRF}(\hat{f}) = \text{RRF}(\bar{f})$ since $\bar{f}$ is optimal. Let $\text{rank}_{\bar{f}}(x_i)$ and $\text{rank}_{\hat{f}}(x_i)$ be the ranks of x_i according to $\bar{f}$ and $\hat{f}$ respectively. We have

$$\text{RRF}(\bar{f}) = \sum_{i=1}^{n} y_i \sum_{\ell=1}^{n} \mathbf{1}_{[\text{rank}_{\bar{f}}(x_i)=\ell-1]} \cdot a_\ell = \sum_{i \in S_+} a_{(\text{rank}_{\bar{f}}(x_i)+1)},$$

$$\text{RRF}(\hat{f}) = \sum_{i=1}^{n} y_i \sum_{\ell=1}^{n} \mathbf{1}_{[\text{rank}_{\hat{f}}(x_i)=\ell-1]} \cdot a_\ell = \sum_{i \in S_+} a_{(\text{rank}_{\hat{f}}(x_i)+1)}.$$

Since $a_1 \leq a_2 \leq \cdots \leq a_n$, it suffices to show that the ranks occupied by the positive examples under $\hat{f}$ are at least as high as the ranks occupied by the positives under $\bar{f}$.

First consider the examples that are not tied with any other examples under $\bar{f}$. The ranks of these examples are still the same under $\hat{f}$ because all pairwise orderings of examples that are not tied are preserved by Lemma 3. Now consider a set of examples that are tied under $\bar{f}$. The ranks of these examples will be permuted within the set. If all of the examples are positive, or all of the examples are negative, then the objective value remains the same since the positive examples in the set still occupy the exact same ranks. On the other hand, suppose there are both positive and negative examples in the tied set. By Definition 1, the negative examples have the highest ranks in the set under $\bar{f}$. Once the ties are broken under $\hat{f}$, the objective value changes only if a positive example moves into a position in the ranked list that was previously the position of a negative example. Thus, the positive examples can occupy only higher ranks under $\hat{f}$, not lower ranks. This proves $\text{RRF}(\hat{f}) \geq \text{RRF}(\bar{f})$, which implies $\hat{f} \in \text{argmax}_f \text{RRF}(f)$ has no ties, satisfying the condition of Lemma 2. $\square$

2.4 Computational Results

In this section, we demonstrate the performance of two of the MIO formulations presented above: (2.6) and (2.18). Unlike (2.18), (2.9) currently may be too hard for most computers to solve, except for small or sparse problems.

2.4.1 Proof-of-Concept

We begin with a proof-of-concept study, using an artificial dataset called ROC Flexibility.[3] Before describing the dataset, we briefly explain ROC (receiver operating characteristic) curves: To plot an ROC curve for a given ranking of examples, start at the origin, and for each example in order of the ranking, trace the curve one unit up if the example is positive and one unit to the right if the example is negative. The axes are normalized so that the curve ends at the point (1,1). Figure 2-2 shows ROC curves corresponding to a perfect ranking (all positive examples on top of all negatives), a perfect misranking (all negative examples on top of all positives), and a random ranking (each example in the ranked list is positive with probability 0.5 and negative with probability 0.5). The area under the ROC curve (AUC) is one for a perfect ranking and zero for a perfect misranking.

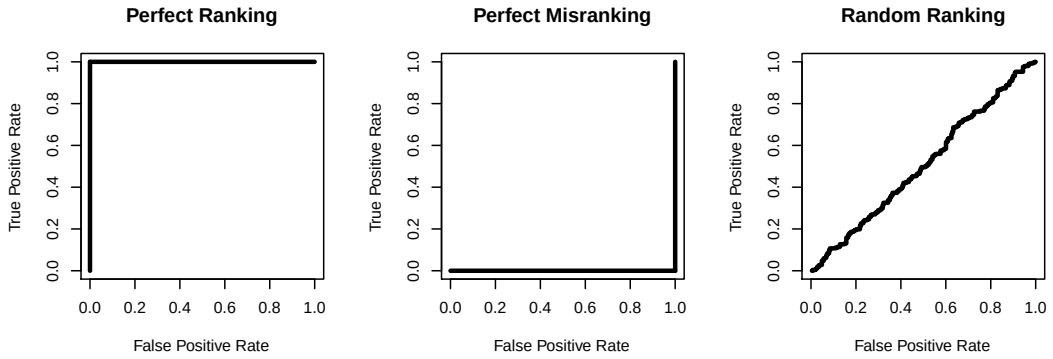
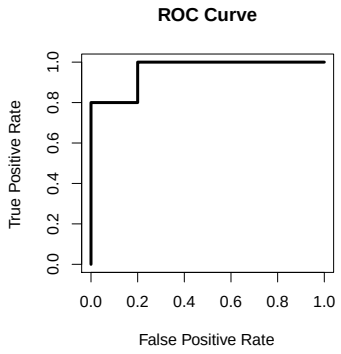


Figure 2-2: ROC curves for perfect ranking, perfect misranking, and random ranking.

One interpretation of an ROC curve is as a visualization of how the false positive and true positive rates of a binary classifier change as its discrimination threshold is

[3]Available at: http://web.mit.edu/rudin/www/ROCFlexibilityData/ROCFlexibilityData.html.

varied. For instance, Figure 2-3 shows the ROC curve for ten examples, five positive and five negative, corresponding to the ranking in the adjacent table. Consider using the ranking function for classification instead, and consider different settings of the discrimination threshold. If the threshold were such that only the highest example is classified as positive and the rest as negative, then 0.2 of the positive examples are true positives and 0 of the negative examples are false positives, so the false positive and true positive rates are 0 and 0.2 respectively. The rates for the other settings of the threshold are shown in the table in Figure 2-3. This example shows how plotting the false positive rate versus true positive rate also yields the ROC curve.



i	Label	False Pos. Rate	True Pos. Rate
1	+	0	0.2
2	+	0	0.4
3	+	0	0.6
4	+	0	0.8
5	−	0.2	0.8
6	+	0.2	1
7	−	0.4	1
8	−	0.6	1
9	−	0.8	1
10	−	1	1

Figure 2-3: ROC curve with corresponding false positive and true positive rates for discrimination thresholds between examples i and $i + 1$.

The ROC Flexibility dataset was designed so that there would be flexibility in the performance of different algorithms. To illustrate what is meant by flexibility, we plot the ROC curves that correspond to ranking the examples by each of the five features, that is, for each feature in turn, treating the feature value as the score and ranking the examples by this score. Figure 2-4 shows the ROC curves corresponding to ranking by each of the five features of the ROC Flexibility dataset. For example, in the first plot, we use the first feature as the score and construct the corresponding ROC curve. The sixth plot in the figure that overlays all five ROC curves shows that there is a "step" in each of the curves, and the position of each step is distinct, so that there is a portion of the ROC curve for each feature that does not overlap with the ROC curve of any other feature. Thus, linear combinations of the features can

correspond to rankings that lead to a wide variety of ROC curves, and we expect different algorithms to perform differently from each other.

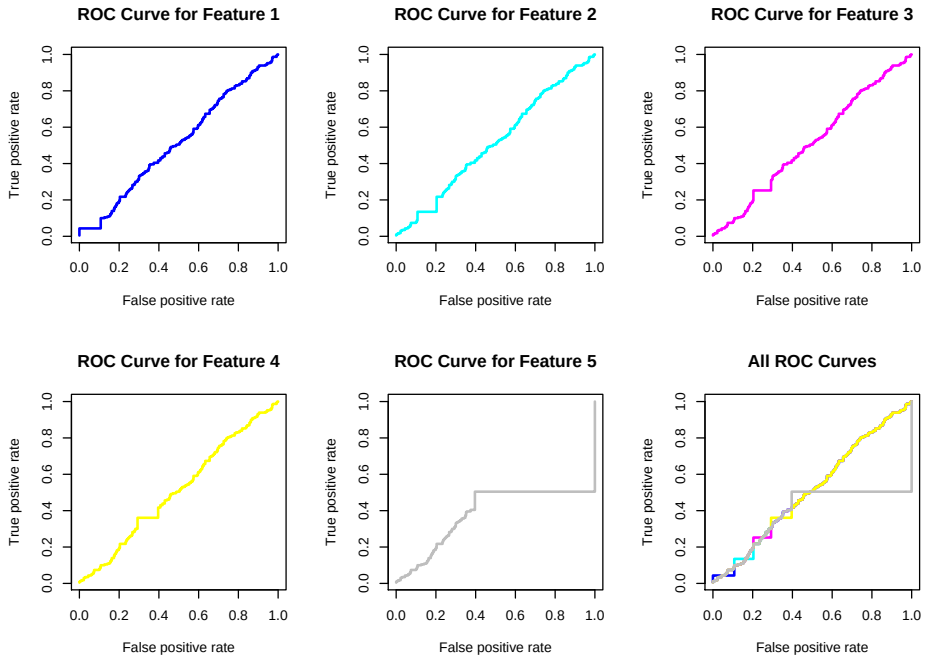


Figure 2-4: ROC curves for individual features of ROC Flexibility data.

We compare the performance of our AUC method—both the MIO formulation and its linear relaxation (LP)—to that of three algorithms: RankBoost (RB), logistic regression (LR), and a support vector machine (SVM)-style ranking algorithm, corresponding to minimizing (2.3), (2.5), and (2.4) respectively. There are other possible ranking algorithms, such as the others listed in Section 2.1.4, but we chose these three as a sample of practical and widely used methods. Note that for these algorithms, we minimize only the loss functions, without regularization terms in the objectives. In this work, we do not tune regularization parameters for any algorithm since our main goal is to investigate the advantage of optimizing the exact loss function over heuristic loss functions; by tuning parameters, it would be unclear whether the advantage is from the loss function or from the regularization.

The approximate methods were all run using MATLAB 7.8.0. To solve the MIO and LP versions of (2.6), we used ILOG AMPL 11.210 with the CPLEX 11.2.1 solver. For each algorithm, we randomly divided the dataset into 250 training and 250 test

Table 2.4: Mean AUC (%) on ROC Flexibility data (approximate algorithms).

	RB	**LR**	**SVM**	**LP**
Train	71.0959	72.0945	71.1840	70.6327
Test	65.9028	67.4607	67.7844	67.1797

Table 2.5: Mean AUC (%) on ROC Flexibility data (MIO algorithm).

	MIO ($\varepsilon = 10^{-6}$)	**MIO** ($\varepsilon = 10^{-5}$)	**MIO** ($\varepsilon = 10^{-4}$)	**MIO** ($\varepsilon = 10^{-3}$)
Train	78.8738	80.6163	80.6163	80.5848
Test	78.8464	81.7706	81.7706	81.6525
Time (s)	69.331	173.992	206.172	356.878

examples, and after generating the model using the training data, computed the AUC for both sets of data. We repeated this process ten times. Table 2.4 shows the mean training and test AUC values over the ten trials for RB, LR, SVM, and LP, which all had negligible runtimes; the LP results are presented for $\varepsilon = 10^{-4}$ as this value resulted in better performance than 10^{-3}, 10^{-5}, and 10^{-6}. Table 2.5 shows the mean AUC values for the MIO method with various values of ε. We note the following:

- Increasing ε results in slower runtimes for the MIO algorithm. Also, as explained in Section 2.2.1, the MIO formulation may terminate with a suboptimal solution if ε is too large because then the formulation is no longer exact. For $\varepsilon = 10^{-3}$ in Table 2.5, eight of the ten trials produced the same training AUC as $\varepsilon = 10^{-4}$ and $\varepsilon = 10^{-5}$, but the other two produced a lower AUC.
- Decreasing ε expands the solution space, so theoretically the optimal value can only increase with smaller ε for the MIO algorithm. However, ε must be large enough for the solver to recognize it as nonzero; if ε is too small, the numerical instability may cause the solver to terminate with suboptimal solutions. For $\varepsilon = 10^{-6}$ in Table 2.5, half of the ten trials produced the same training AUC as $\varepsilon = 10^{-4}$ and $\varepsilon = 10^{-5}$, but the other half produced a lower AUC.
- The MIO algorithm for $\varepsilon = 10^{-4}$ and $\varepsilon = 10^{-5}$ performed dramatically better than the approximate algorithms. The mean MIO training AUC was about 11.8% higher than the best of its competitors (LR), and the mean MIO test AUC was about 20.6% higher than the best of its competitors (SVM).

Results on the ROC Flexibility data show that the MIO has the potential to perform substantially better than other methods. This dataset was designed so that optimizing a performance metric using different algorithms can lead to dramatically different ranked lists. It appears that when a dataset has this level of flexibility, optimizing the objective exactly can have a substantial benefit. In the remainder of this section, we show computational results using other datasets. The experiments using (2.6) and (2.18) were run using ILOG AMPL 11.210 on a computer with two Intel quad core Xeon E5440 2.83GHz processors and 32GB of RAM. The LP solutions were generated using the CPLEX 11.2.1 solver with $\varepsilon = 10^{-4}$. The MIO solutions were generated using the Gurobi 3.0.0 solver with $\varepsilon = 10^{-6}$. These choices of solver and value of ε were based on results from Section 2.4.1 and preliminary experiments for each dataset. For the experiments in Sections 2.4.2 and 2.4.3, the Gurobi solver was numerically stable with $\varepsilon = 10^{-6}$. ROC Flexibility was the only dataset for which CPLEX performed better than Gurobi on the MIO problem, thus we used Gurobi for all other datasets. The approximate methods were all run using MATLAB 7.8.0.

2.4.2 Maximizing AUC

We solved (2.6) using five other datasets: FourClass and SVMGuide1 are from the LIBSVM collection,[4] and the others are from the UCI Machine Learning Repository [Asuncion and Newman, 2007]. For each dataset, we randomly divided the data into training and test sets, and compared the performance of RB, LR, SVM, LP, and MIO. This experiment was repeated ten times. As an example of the ROC curves that occur in these datasets, Figure 2-5 shows the curves for the Liver, SVMGuide1, and MAGIC data. There is generally more overlap between the ROC curves of the features than in the ROC Flexibility dataset. For the SVMGuide1 data, there are in fact two overlapping features that both correspond to nearly perfect ranking. Thus, we expect less variation in the performance of the different methods, but we can still use the MIO method as a benchmark to corroborate their ranking quality.

Table 2.6 shows the mean training and test AUC over the ten trials for each dataset

[4] http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/

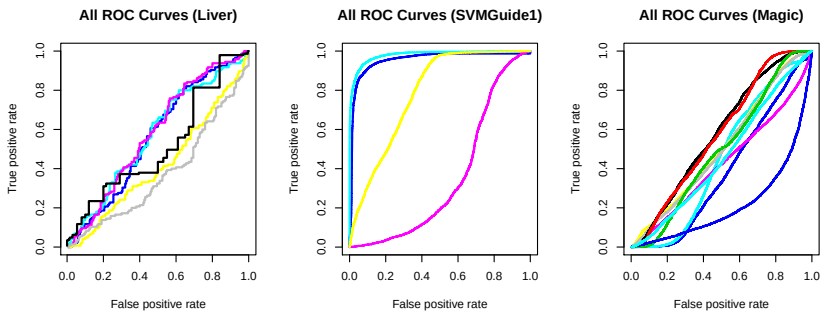


Figure 2-5: ROC curves for Liver, SVMGuide1, and MAGIC datasets.

and algorithm; for completeness we included the ROC Flexibility dataset in the table. Bold indicates the value is not significantly smaller than the highest value in its row at the 0.05 significance level, according to a matched pairs t-test. In particular, for each row in the table, let μ_1 be the population mean AUC for the algorithm with the largest sample mean AUC, where the sample consists of the ten AUC values recorded for that algorithm. Then for each of the algorithms, denote by μ_2 the population mean AUC for that algorithm, and test the null hypothesis $H_0 : \mu_1 = \mu_2$ against the alternative $H_1 : \mu_1 > \mu_2$. We use boldface for that algorithm's entry in the table whenever the test does not reject H_0. It is possible for multiple numbers in a row to be bold if they are all not statistically smaller than the highest in the row. It is also possible for numbers to not be bold even though they are greater than others in the row if the corresponding t-statistic is sufficiently large to reject H_0.

The MIO achieved the statistically highest mean AUC for all training and test sets. The LP also performed well on the test data. Table 2.7 shows for each dataset the number of times out of ten that each method performed best, that is, achieved the highest AUC, on the training and test data; bold indicates the highest count in each row. The counts in the MIO column clearly dominate the counts of the other methods. Not all of the training AUC values are highest for the MIO since not all of the problems solved to optimality. For the Liver data, there were some trials for which certain algorithms tied for the highest test AUC, and for the MAGIC data, there was one trial for which the LP and MIO algorithms tied for the highest training AUC, thus the counts in these rows sum to greater than ten.

Table 2.6: Mean AUC (%) on training and test.

Dataset		**RB**	**LR**	**SVM**	**LP**	**MIO**
Liver Disorders	train	73.6621	74.0374	74.2802	74.2816	**75.3802**
	test	**70.6463**	**70.8567**	**70.9705**	**70.9691**	**71.0257**
ROC Flexibility	train	71.0959	72.0945	71.1840	70.6327	**80.6163**
	test	65.9028	67.4607	67.7844	67.1797	**81.7706**
FourClass	train	83.0278	82.9907	83.1853	83.1857	**83.2230**
	test	82.8790	82.8050	83.0438	**83.0492**	**82.9861**
SVMGuide1	train	99.1929	99.2255	99.2330	99.2327	**99.2384**
	test	**99.0520**	**99.0563**	99.0649	**99.0651**	**99.0642**
Abalone	train	91.3733	91.4723	91.5078	91.5067	**91.5135**
	test	90.5580	90.6139	**90.6415**	**90.6411**	**90.6419**
MAGIC	train	84.0105	84.0280	84.4880	84.4903	**84.4943**
	test	83.5273	83.5984	**83.9349**	**83.9365**	**83.9370**

Table 2.7: Problem dimensions and number of times each method performs best.

Dataset	n_{train}	n_{test}	d		**RB**	**LR**	**SVM**	**LP**	**MIO**
Liver Disorders	172	173	6	train	0	0	0	0	**10**
				test	1	1	3	3	**5**
ROC Flexibility	250	250	5	train	0	0	0	0	**10**
				test	0	0	0	0	**10**
FourClass	431	431	2	train	0	0	0	1	**9**
				test	1	1	0	3	**5**
SVMGuide1	700	6389	4	train	0	2	0	0	**8**
				test	1	3	0	2	**4**
Abalone	1000	3177	10	train	0	0	0	0	**10**
				test	1	1	2	2	**4**
MAGIC	1000	18020	10	train	0	0	0	3	**8**
				test	0	0	2	**4**	**4**

2.4.3 Maximizing RRF

As illustrated in Section 2.1.2, the RRF formulation encompasses many rank statistics. For this set of experiments, we choose one rank statistic—DCG@N—and solve the formulation with three datasets. The first was the ROC Flexibility dataset we solved for the previous formulation, and the objective of interest was the DCG over the top 30% of the ranked list. The other two were the Haberman's Survival and Pima Indians Diabetes datasets from the UCI Machine Learning Repository. The objective of interest for both of these datasets was the DCG over the top 10% of the ranked list. Note that the more we focus at the top of the list, the faster the MIO

solves since more of the a_ℓ coefficients are zero. We used the top 30% instead of top 10% for the ROC Flexibility dataset because it contains so many identical examples that out of 250 examples in a training set, there would be no examples with minimum ranks between 225 and 249.

For each dataset, we randomly divided the data into training and test sets, and compared the performance of RankBoost (RB), the P-Norm Push with $p = 2, 4, 8$ (P2, P4, P8), logistic regression (LR), the support vector machine (SVM)-style ranking algorithm, the linear relaxation (LP), and the MIO. This experiment was repeated ten times, using $\varepsilon = 10^{-4}$ to solve all LPs and $\varepsilon = 10^{-6}$ to solve all MIOs. Again, we chose these parameters based on insights from Section 2.4.1 and preliminary experiments on the datasets. Note that the Haberman's Survival and Pima Indians Diabetes problems were solved using (2.18) as shown in Section 2.2.2, but the ROC Flexibility problem was solved using a slightly different formulation—namely, we omitted (2.21) through (2.23), and replaced (2.19) with $v_i = w^T x_i$ for all i and $z_{ik} \leq v_i - v_k + 1 - \varepsilon$ for $i \in S_+, k \in 1, \ldots, n$; this choice of constraints ran faster for this particular dataset.

In the ROC Flexibility dataset, there are 500 examples in total, but only 20 unique examples. In the Haberman's Survival dataset, there are 306 examples in total, and 283 unique examples. In the Pima Indians Diabetes dataset, there are 768 unique examples. We evaluate solutions from (2.18) in terms of two quality measures: DCG@N defined with minimum ranks and DCG@N defined with ranks. The difference between the two objectives is largest for the ROC Flexibility data because the percentage of unique examples is smallest in this dataset. The two objectives are close for the Haberman's Survival data since most examples are unique, and they are exactly equal for the Pima Indians Diabetes data since all examples are unique, in accordance with Theorem 1.

Tables 2.8 and 2.9 show the mean training and test objectives over the ten trials for each dataset and algorithm; bold indicates the value is not significantly smaller than the highest value in its row at the 0.05 significance level, according to a matched pairs t-test. Tables 2.10 and 2.11 show for each dataset the number of times out of ten that each method performed best on the training and test data; bold indicates the

Table 2.8: Mean RRF (with minimum ranks) on training (top) and test (bottom).

Data	RB	P2	P4	P8	LR	SVM	LP	MIO
ROC Flexibility	7.5845	7.9456	8.3329	9.3744	8.1970	8.1262	6.7895	**11.7136**
	6.9096	7.3402	7.8322	8.1412	7.6468	7.1133	6.8940	**12.0470**
Haberman	5.1133	5.1890	5.1768	5.2478	5.0423	5.0381	4.8950	**5.8080**
	4.7644	4.7170	4.6961	4.7508	**4.7901**	**4.7877**	**4.8679**	**5.1701**
Pima Diabetes	7.1586	7.3360	7.4221	**7.5265**	7.1732	6.8701	6.7943	**7.5849**
	10.7014	**10.6961**	**10.7544**	**10.6886**	**10.7582**	**10.7116**	10.3859	10.0964

Table 2.9: Mean RRF (with ranks) on training (top) and test (bottom).

Data	RB	P2	P4	P8	LR	SVM	LP	MIO
ROC Flexibility	13.8694	14.3971	14.8264	15.3331	14.1901	13.5106	8.1689	**15.9534**
	12.6536	13.3732	13.9773	14.9208	13.0913	12.3905	8.8177	**16.2293**
Haberman	5.1230	5.2003	5.1908	5.2954	5.0517	5.0503	4.9309	**5.8227**
	4.8127	4.7567	4.7836	4.7886	**4.8136**	**4.8254**	**4.8907**	**5.1756**
Pima Diabetes	7.1586	7.3360	7.4221	**7.5265**	7.1732	6.8701	6.7943	**7.5849**
	10.7014	**10.6961**	**10.7544**	**10.6886**	**10.7582**	**10.7116**	10.3859	10.0964

highest count in each row. For some trials, there were multiple algorithms that tied for the best, so the counts in the rows do not necessarily sum to ten. Both sets of tables show an advantage of the MIO over the other methods for the ROC Flexibility and Haberman's Survival data. The MIO performed well on the Pima Indians Diabetes training data too, but there were not enough examples for the results to generalize to the test data. Overall, our experiments support the hypothesis that MIO yields useful solutions, and has a competitive advantage over other methods.

2.4.4 Computational Speed

Table 2.12 shows for each MIO and LP problem the mean and standard deviation of time (in seconds) taken to find the final solutions for the ten trials. Certain problems solved to optimality, so the time recorded was the time until the optimal solution was found. Most of the MIOs did not solve to optimality before the cutoff time indicated in Table 2.12, in which case the time recorded was the time until the last (possibly

Table 2.10: Number of times each method performs best (RRF with minimum ranks).

Data	n_{train}	n_{test}	d		RB	P2	P4	P8	LR	SVM	LP	MIO
ROC Flexibility	250	250	5	train	0	0	0	0	1	1	0	**8**
				test	0	0	0	1	1	1	0	**8**
Haberman	153	153	3	train	0	1	1	0	0	0	0	**10**
				test	0	0	0	0	1	1	1	**7**
Pima Diabetes	250	518	8	train	0	0	0	4	0	0	0	**6**
				test	2	0	1	2	**3**	2	1	0

Table 2.11: Number of times each method performs best (RRF with ranks).

Data	n_{train}	n_{test}	d		**RB**	**P2**	**P4**	**P8**	**LR**	**SVM**	**LP**	**MIO**
ROC Flexibility	250	250	5	Train	1	2	2	3	2	1	0	**8**
				Test	0	1	2	2	1	1	0	**9**
Haberman	153	153	3	Train	1	2	2	2	1	1	0	**10**
				Test	0	0	0	0	1	1	1	**7**
Pima Diabetes	250	518	8	Train	0	0	0	4	0	0	0	**6**
				Test	2	0	1	2	**3**	2	1	0

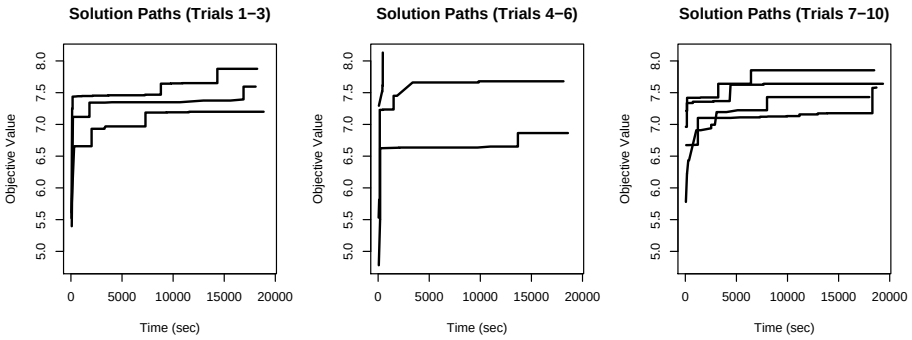


Figure 2-6: Solution paths for RRF problem on Pima Indians Diabetes data.

suboptimal) solution was found. A star in place of a cutoff time for a particular dataset and problem signifies that all ten trials solved to optimality; a star next to a cutoff time signifies that at least one of the trials solved to optimality.

Figure 2-6 shows how the objective value changed over time as we solved the MIO for the ten trials of the RRF minimum rank problem using the Pima Indians Diabetes data. For most of the trials, the solution did not change much after two hours. There was one trial that solved to optimality after 460 seconds. Note that it is often the case that after a solver finds the optimal solution of an MIO problem, it can take an inordinately long time to prove optimality. Thus, it may be that for many problems in our experiments, the solver did in fact find the optimal solution, but was not able to prove optimality before the time limit. Table 2.12 and Figure 2-6 both show that there can be huge variation in the amount of time it takes to solve an MIO problem, even for instances of the same size and from the same dataset.

Table 2.12: Cutoff times and mean times (± one standard deviation) until final solution (in seconds).

	Dataset	Method	Cutoff Time	Mean Time To Final Sol.
AUC	Liver Disorders	MIO	3600 (1 hr)	1950.1 ± 1124.3
		LP	*	0.34 ± 0.08
AUC	ROC Flexibility	MIO	*	174.0 ± 54.4
		LP	*	0.56 ± 0.07
AUC	FourClass	MIO	10800 (3 hrs)	5031.3 ± 3394.8
		LP	*	4.0 ± 2.2
AUC	SVMGuide1	MIO	10800 (3 hrs)	3529.5 ± 3896.2
		LP	*	29.1 ± 8.4
AUC	Abalone	MIO	25200 (7 hrs)	6199.2 ± 7528.8
		LP	*	2103.8 ± 618.7
AUC	MAGIC	MIO	36000 (10 hrs)	20630.7 ± 13267.6
		LP	*	2178.0 ± 2249.2
RRF	ROC Flexibility	MIO	14400 (4 hrs)	2990.9 ± 1445.2
		LP	*	2546.7 ± 550.4
RRF	Haberman Surv.	MIO	3600 (1 hr)*	1361.7 ± 1348.6
		LP	*	6.1 ± 0.6
RRF	Pima Diabetes	MIO	18000 (5 hrs)*	10752.7 ± 5431.8
		LP	*	13.3 ± 2.4

*Solved to optimality: all LPs, all MIOs for ROC Flexibility AUC, 3 MIOs for Haberman Survival RRF, 1 MIO for Pima Indians Diabetes RRF

2.5 Future Work

In order to scale the MIO methods to larger problems, it may be possible to find conditions–that is, characteristics of the data–under which the linear relaxation is close to the convex hull of integer feasible solutions. In these cases, the LP may yield particularly high-quality solutions. Since the LP can already be solved for larger problems, this would be a way to leverage the advantages of MIO to get higher quality solutions on the large scale. Another possible direction is to experiment with different types of regularization that are natural for MIO. A simple example is to vary ε and/or the bounds on the w_j's. This may serve two purposes at once; with these forms of regularization we might be able to gain better test accuracy as well as better computational speed.

2.6 Summary

We have developed a new approach to address supervised ranking tasks in machine learning. We have defined the class of rank risk functionals (RRF), which includes a number of established linear rank statistics as well as novel measures such as the staircase rank statistic. We have introduced methods for maximizing the AUC and the RRF. Our methods take advantage of the modeling power of MIO, and we have presented promising evidence for the ability of MIO to solve ranking problems. Since the MIO approach directly optimizes the objective functions of interest instead of using surrogates, optimal solutions to the MIO formulations achieve higher training objective values than solutions generated from other machine learning methods. The MIO methods also demonstrate an ability to generalize to test data, and they compete well against other methods.

Chapter 3

MIO for Reverse-Engineering Quality Rankings

Many organizations depend on the top ratings given to their products or services by quality rating companies. For instance, the reputations of undergraduate and graduate programs at colleges and universities depend heavily on their *U.S. News and World Report* rankings. Similarly, mortgage providers rely on the models of credit rating agencies such as *Experian, Equifax* and *TransUnion*, while businesses rely on *Standard and Poor's, Moody's*, and *Dunn and Bradstreet* credit ratings, and mutual funds rely on *Morningstar* ratings. For electronics, rating companies include *CNET* and *PCMag*; and for vehicles, they include *What Car?, JDPower, Edmunds, Kelley Blue Book*, and *Car and Driver*. Most of these rating companies use a formula to score products, and few of them make their complete rating formulas public. If organizations were able to recreate these formulas, they would better understand the standards by which their products were being judged, which would potentially allow them to produce better products, or at least products with better ratings. Furthermore, rating companies that are aware of reverse-engineering may be motivated to re-evaluate the accuracy of their formulas in representing the quality of products.

In this chapter, we introduce a method for reverse-engineering product ranking models. The method integrates knowledge about the way many such ranking models are commonly constructed, which is summarized in the following points:

- **Point 1 (Linear scoring functions):** The rating company states publicly that its product rankings are based on real-valued scores given to each product, and that the score is a weighted linear combination of a known set of factors. The precise values for some factors can be obtained directly, but other factors have been discretized into a number of "stars" between 1 and 5 and are thus noisy versions of the true values. For example, the National Highway Traffic Safety Administration discretizes factors pertaining to vehicle safety ratings.

- **Point 2 (Category structure):** Products are organized into *categories*, and within each category there are one or more *subcategories*. For example, a computer rating company may have a laptop category with subcategories such as netbooks and tablets. Products within a category share the same scoring system, but the ranking of each product is with respect to its subcategory.

- **Point 3 (Ranks over scores):** It is not as essential for organizations to be able to reproduce the scores assigned by rating companies as it is to reproduce the ranks, since consumers pay more attention to product ranks than to scores or to differences in score. Moreover, sometimes only the ranks are available in the data, and not the scores.

- **Point 4 (Focus on top products):** Consumers generally focus on top-ranked products, so a model that can reproduce the top of each subcategory's ranked list accurately is more valuable than one that better reproduces the middle or bottom of the list.

Reverse-engineering product quality rankings is a new application for machine learning, and the algorithm we provide for this task matches the application in conforming to the four points above. We use linear combinations of the same factors used by the rating company, and generate a separate model for each category, in accordance with Points 1 and 2. The reverse-engineered model for a given category is provided by a supervised ranking algorithm that uses discrete optimization to force the ranks produced by our algorithm to be similar to the ranks from the rating company; note that the

algorithm reproduces the ranks, not the scores, as in Point 3. Specifically, the model is constructed to obey certain preference relationships in accordance with Point 4, that is, within each subcategory, the rankings of the rating companies' top-k products should match the top-k rankings from our model. When there are not enough data within a category to reliably determine the ranking model for that category, our algorithm draws strength across categories by using data from other categories as a type of regularization. Our experimental results indicate an advantage in sharing information across product categories, modeling ranks rather than scores, and using discrete optimization to maximize the exact rank statistic of interest rather than a convex proxy, similar to the results of the previous chapter.

Note that even though Point 1 makes the assumption of known factors, it is also possible to use our method for problems in which the factors are unknown. As long as the factors in our model encompass the information used for the rating system, our algorithm can be applied regardless of whether or not the factors are precisely the same as those used by the rating company. For instance, a camera expert might know all of the potential camera characteristics that could contribute to camera quality, which we could then use as the factors in our model.

After the model has been reverse-engineered, we can use it to determine the most cost-effective way to increase product rankings, and we present discrete optimization algorithms for this task. These algorithms may be used independently of the reverse-engineering method. That is, if the reverse-engineered formula were obtained using a different method from ours, or if the formula were made public, we could still use these algorithms to cost-effectively increase a product's rank.

We describe related work in Section 3.1. In Section 3.2, we derive a ranking quality objective that encodes the preference relationships discussed above. In Section 3.3 we provide the machine learning algorithm, based on discrete optimization, that exactly maximizes the ranking quality objective. In Section 3.4, we establish new measures that can be used to evaluate the performance of our model. In Section 3.5, we derive several baseline algorithms for reverse-engineering that all involve convex optimization. Section 3.6 contains results from a proof-of-concept experiment, and Section 3.7

provides experimental results using rating data from a major quality rating company. Section 3.8 discusses the separate problem of how to cost-effectively increase the rank of a product. We conclude in Section 3.9. The main contributions are: the application of machine learning to reverse-engineering product quality rankings; our method of encoding the preference relationships in accordance with Points 1 through 4; using data from other product categories as regularization; the design of novel evaluation measures; and the mechanism to cost-effectively achieve a highly ranked product.

3.1 Related Work

We have considered the reverse-engineering task as an application of *supervised ranking* (see Chapter 2 for ranking references). This problem is related to the area of conjoint analysis in marketing [Green et al., 2001]. Conjoint analysts aim to model how a consumer chooses one brand over another, with the goal of learning which product characteristics are most important to consumers.

Reverse-engineering and approximation of rating models has been done in a number of industries, albeit not applied to rankings for consumer products with the category/subcategory structure. This work has mostly been published within blogs and deals with the problem of approximating the ranking function with a smaller number of variables, rather than using the exact factors in the rating company's formula. For instance, Chandler [2006] approximated the U.S. News and World Report Law School rankings using symbolic regression to obtain a formula with four factors and another with seven. Hammer et al. [2007] approximated credit rating models using Logical Analysis of Data. In the sports industry, there has been work in reverse-engineering Elias Sports Bureau rankings, which are used to determine compensation for free agents [Bajek, 2008]. The search engine optimization (SEO) industry aims to be able to boost the search engine rank of a web page by figuring out which features have high influence in the ranking algorithm. For instance, Su et al. [2010] used a linear optimization model to approximate Google web page rankings.

If the ratings are accurate measures of quality, then making the rating model more

transparent could have a uniformly positive impact: it would help companies to make better rated products, and it would encourage rating companies to receive feedback as to whether their rating systems fairly represent quality. However, problems may arise if the ratings are not accurate measures of quality. Unethical manipulation of reverse-engineered credit rating models heavily contributed to the 2007-2010 financial crisis [Morgenson and Story, 2010]. These ratings permitted some companies to sell "junk bonds" with very high ratings. In such cases, the transparency gained from reverse-engineering may encourage the rating companies to align their formulas more closely with quality as defined according to the public interest.

3.2 Encoding Preferences for Quality Rating Data

We derive a rank statistic that serves as our objective for reverse-engineering. Maximizing this objective yields estimates of the weights on each of the factors in the rating company's model. We start with the simple case of one category with one subcategory. Then, we generalize to the case of multiple categories and subcategories. Our method can be used to reverse-engineer quality rankings whether or not the underlying scores are made available; we need only to know the ranks.

3.2.1 One Category, One Subcategory

Let n denote the number of products to be ranked. We represent product i by a vector of d factors $x_i \in \mathcal{X}$, where $\mathcal{X} \subset \mathcal{R}^d$. The rating company assigns a score $\zeta_i \in \mathcal{R}$ to each product i, which translates into a rank. Higher scores imply higher ranks, so that a product with rank 0 is at the bottom of the list with the lowest quality. For all pairs of products, let the preference function $\pi : \mathcal{X} \times \mathcal{X} \to \{0, 1\}$ capture the true pairwise preferences according to the scores ζ_i. That is, let:

$$\pi(x_i, x_k) := \pi_{ik} := \mathbf{1}_{[\zeta_i > \zeta_k]},$$

where $\mathbf{1}_q$ is the indicator function that equals 1 if condition q holds and 0 otherwise. In other words, if product i is ranked higher than product k by the rating company, then π_{ik} is 1. Even if the ζ_i are not available, we assume that the ranks are known, so we can derive the π_{ik}. Our goal is to generate a scoring function $f : \mathcal{X} \to \mathcal{R}$ that assigns real-valued scores $f(x_i)$ to each product x_i such that the π_{ik} values match as closely as possible our model associated preferences $\mathbf{1}_{[f(x_i)>f(x_k)]}$.

Let $\Pi = \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik}$. We first consider a rank statistic that generalizes the area under the ROC curve (AUC):

$$\mathrm{AUC}_\pi(f) := \frac{1}{\Pi} \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik} \mathbf{1}_{[f(x_i)>f(x_k)]}. \tag{3.1}$$

This statistic is related to the disagreement measure introduced by Freund et al. [2003b], as well as Kendall's τ coefficient [Kendall, 1938]. That is, in the absence of ties, the disagreement measure is $1 - \mathrm{AUC}_\pi(f)$ and Kendall's τ is $2\mathrm{AUC}_\pi(f) - 1$. The highest possible value of $\mathrm{AUC}_\pi(f)$ is 1, which is achieved if the scoring function f satisfies $f(x_i) > f(x_k)$ for all pairs (x_i, x_k) such that $\pi_{ik} = 1$. There is a formulation in Chapter 2 that maximizes $\mathrm{AUC}_\pi(f)$. Here we modify $\mathrm{AUC}_\pi(f)$ to capture additional information about product quality rankings.

$\mathrm{AUC}_\pi(f)$ does not put any emphasis on the top of the ranked list; a product at the bottom of the ranked list can contribute the same amount to $\mathrm{AUC}_\pi(f)$ as a product at the top. However, as noted in Point 4 in the introduction, it is often more important to accurately reproduce rankings at the top of the list than in the middle or at the bottom. Suppose we want to concentrate on the top $\bar{T}$ products within the subcategory. In particular, we want to weigh the top $\bar{T}$ products $1 + \theta$ times more than the rest of the list, where $\theta \geq 0$. To do this, we first define the rank of product i, with respect to scoring function f, to be the number of products it is scored strictly above:

$$\mathrm{rank}_f(x_i) := \sum_{k=1}^{n} \mathbf{1}_{[f(x_i)>f(x_k)]}.$$

Note that this is the minimum rank from Chapter 2; we assume in this chapter that

the different products being ranked are distinct, so that the minimum rank equals the rank. The top $\bar{T}$ products have rank at least $T := n - \bar{T}$. For example, if $n = 10$, then assuming no ties in rank, the top $\bar{T} = 4$ products have ranks at least $T = 6$, that is, their ranks are 6, 7, 8, and 9. We consider the objective function:

$$\mathrm{AUC}_{\pi}^{\mathrm{top}}(f) := \frac{1}{\Pi(\theta)} \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik} \mathbf{1}_{[f(x_i) > f(x_k)]} \left(1 + \theta \mathbf{1}_{[\mathrm{rank}_f(x_i) \geq T]}\right),$$

where we normalize by

$$\Pi(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik} \left(1 + \theta \mathbf{1}_{[\sum_{k=1}^{n} \pi_{ik} \geq T]}\right).$$

Note that $\mathrm{AUC}_{\pi}^{\mathrm{top}}(f)$ varies between 0 and 1 since the largest possible value of the summation in $\mathrm{AUC}_{\pi}^{\mathrm{top}}(f)$ is $\Pi(\theta)$, which is achieved if f ranks all pairs (x_i, x_k) correctly. Each pair of products (x_i, x_k) contributes $\frac{1}{\Pi(\theta)} \pi_{ik} \mathbf{1}_{[f(x_i) > f(x_k)]}(1 + \theta)$ to the objective if the rank of x_i is at least T, and contributes $\frac{1}{\Pi(\theta)} \pi_{ik} \mathbf{1}_{[f(x_i) > f(x_k)]}$ otherwise. If either $\theta = 0$ or $T = 0$, then maximizing this objective is equivalent to maximizing $\mathrm{AUC}_{\pi}(f)$, which does not focus at the top.

3.2.2 Multiple Categories and Subcategories

We assume that different categories have different ranking models, as stated in the introduction. Even so, these models may be similar enough that knowledge obtained from other categories can be used to "borrow strength" when there are limited data in the category of interest. Thus, as we derive the objective for reverse-engineering the model f for one prespecified category, we use data from all of its subcategories as well as from the subcategories in other categories.

Let S_{sub} be the set of all subcategories across all categories, including the category of interest, and let there be n_s products in subcategory s. Analogous to our previous notation, $x_i^s \in \mathcal{R}^d$ represents product i in subcategory s, $\zeta_i^s \in \mathcal{R}$ is the score assigned to product i in subcategory s, and π_{ik}^s is 1 if $\zeta_i^s > \zeta_k^s$ and is 0 otherwise. The threshold T_s defines the top of the list for subcategory s.

Our general objective is a weighted sum of $\mathrm{AUC}_\pi^{\mathrm{top}}(f)$ over all subcategories:

$$\mathrm{AUC}_\pi^{\mathrm{top,sub}}(\theta, C)(f) = \sum_{s \in S_{\mathrm{sub}}} \frac{C_s}{\Pi_s(\theta)} \sum_{i=1}^{n_s} \sum_{k=1}^{n_s} \pi_{ik}^s \mathbf{1}_{[f(x_i^s) > f(x_k^s)]} \left(1 + \theta \mathbf{1}_{[\mathrm{rank}_f^s(x_i^s) \geq T_s]}\right), \tag{3.2}$$

where

$$\mathrm{rank}_f^s(x_i^s) = \sum_{k=1}^{n_s} \mathbf{1}_{[f(x_i^s) > f(x_k^s)]}. \tag{3.3}$$

The normalization constants are

$$\Pi_s(\theta) = \sum_{r \in \mathrm{cat}(s)} \sum_{i=1}^{n_r} \sum_{k=1}^{n_r} \pi_{ik}^r \left(1 + \theta \mathbf{1}_{[\sum_{k=1}^{n_r} \pi_{ik}^r \geq T_r]}\right), \tag{3.4}$$

where $\mathrm{cat}(s)$ denotes the category to which subcategory s belongs. The values C_s determine how much influence each subcategory has on the model. It is logical in general for C_s to be the same for all subcategories within a certain category. If there is a sufficient number of rated products in the category of interest, relative to the total number d of factors, then we can train the model with only these data. In that case, we would set $C_s = 1$ for subcategories within the category of interest and $C_s = 0$ for subcategories in all other categories. On the other hand, if the number of products in the category of interest is too small to permit the model to generalize, then we can regularize by setting $C_s \in (0, 1]$ for subcategories of other categories, choosing the values of C_s by cross-validation.

Note that $\Pi_s(\theta)$ is the same for all subcategories s within the same category, instead of being proportional to the size of the subcategory. This is because we want each pair of products within the same category to have the same influence on the objective function. Consider if the normalization constants were alternatively

$$\Pi_s(\theta) = \sum_{i=1}^{n_s} \sum_{k=1}^{n_s} \pi_{ik}^s \left(1 + \theta \mathbf{1}_{[\sum_{k=1}^{n_s} \pi_{ik}^s \geq T_s]}\right).$$

Then for a particular category, there may be subcategories with large values of $\Pi_s(\theta)$

and others with small values. But in this case, assuming C_s is the same for all subcategories in this category, a misranked pair lowers the objective by much more in the subcategories with small $\Pi_s(\theta)$ values than with large values (since $\Pi_s(\theta)$ is in the denominator). Thus in some sense, normalizing this way puts more weight on accurately ranking within the smaller subcategories. To avoid this issue, we use (3.4) to normalize. Conventional ranking methods do not address the subcategory/category structure of our product ranking problem in this manner, and in fact it can be difficult to take the normalization into account accurately if the learning algorithm is limited to convex optimization. We show in Section 3.3 how our algorithm incorporates this form of normalization in an exact way.

We assume a linear form for the model. That is, we assume that the scoring function has the form $f(x) = w^T x$, so that $w \in \mathcal{R}^d$ is a vector of variables in our formulation, and the objective in (3.2) is a function of w. Note that we can also capture nonlinear rating systems using a linear model with nonlinear factors.

3.3 Optimization

We now provide an algorithm to reverse-engineer quality rankings that exactly maximizes (3.2). The algorithm is called MIO-RE—Mixed Integer Optimization for Reverse-Engineering, and expands on the technique in Chapter 2 for supervised ranking in machine learning. Recall from the previous chapter that this type of approach has an advantage over other machine learning techniques in that it exactly optimizes the objective, which tends to achieve higher levels of performance. This advantage is counterbalanced by a sacrifice in computational speed, but for the rating problem, new data come out occasionally (e.g., yearly, monthly, weekly) whereas the computation time is generally on the order of hours, depending on the number of products in the training data and the number of factors. In this case, the extra computation time needed to produce a better solution is worthwhile.

3.3.1 Model for Reverse-Engineering

The variable v_i^s represents the model's score $w^T x_i^s$ of product i in subcategory s, and the binary variable z_{ik}^s captures the decision $\mathbf{1}_{[v_i^s > v_k^s]}$, as in (3.3). The strict inequality is numerically defined using a small positive constant ε, that is:

$$z_{ik}^s = \mathbf{1}_{[v_i^s - v_k^s \geq \varepsilon]}. \tag{3.5}$$

Thus $\text{rank}_f^s(x_i^s) = \sum_{k=1}^{n_s} z_{ik}^s$. To keep track of which products are in the top, we want the binary variable t_i^s to be 1 only if $\text{rank}_f^s(x_i^s)$ is at least T_s:

$$t_i^s = \mathbf{1}_{[\sum_{k=1}^{n_s} z_{ik}^s \geq T_s]}. \tag{3.6}$$

Also, we want the binary variable u_{ik}^s to be 1 only if both $v_i^s - v_k^s \geq \varepsilon$ and $\text{rank}_f^s(x_i^s) \geq T_s$, which is equivalent to:

$$u_{ik}^s = \min\{z_{ik}^s, t_i^s\}. \tag{3.7}$$

Here is the MIO formulation that maximizes (3.2):

$$\max_{w,v,z,t,u} \quad \sum_{s \in S_{\text{sub}}} \frac{C_s}{\Pi_s(\theta)} \sum_{i=1}^{n_s} \sum_{k=1}^{n_s} \pi_{ik}^s (z_{ik}^s + \theta u_{ik}^s) \tag{3.8}$$

$$\text{s.t.} \quad v_i^s = w^T x_i^s, \quad \forall s, i,$$

$$z_{ik}^s \leq v_i^s - v_k^s + 1 - \varepsilon, \quad \forall s, i, k, \tag{3.9}$$

$$T_s t_i^s \leq \sum_{k=1}^{n_s} z_{ik}^s, \quad \forall s, i, k, \tag{3.10}$$

$$u_{ik}^s \leq z_{ik}^s, \quad \forall s, i, k, \tag{3.11}$$

$$u_{ik}^s \leq t_i^s, \quad \forall s, i, k, \tag{3.12}$$

$$0 \leq w_j, u_{ik}^s \leq 1, \quad \forall j, s, i, k,$$

$$z_{ik}^s, t_i^s \in \{0, 1\}, \quad \forall s, i, k.$$

As in Chapter 2, we use (3.8) to refer to the entire formulation. Constraints (3.9) through (3.12) capture (3.5) through (3.7). If $v_i^s - v_k^s \geq \varepsilon$, then the right-hand-side

of (3.9) is at least 1, so the solver sets $z_{ik}^s = 1$ because it is maximizing z_{ik}^s. Otherwise, the right-hand-side is strictly smaller than 1, so the solver sets $z_{ik}^s = 0$. Similarly, if $\sum_{k=1}^{n_s} z_{ik}^s \geq T_s$, then (3.10) implies $t_i^s = 1$; note that since we are maximizing u_{ik}^s in (3.8), we are also maximizing t_i^s because of (3.12). And if both $v_i^s - v_k^s \geq \varepsilon$ and $\text{rank}_f^s(x_i^s) \geq T_s$, then $z_{ik}^s = t_i^s = 1$, so (3.11) and (3.12) imply $u_{ik}^s = 1$. We do not need to explicitly specify u_{ik}^s as a binary variable because u_{ik}^s is the minimum of two binary variables; if either z_{ik}^s or t_i^s is 0, then u_{ik}^s is 0, and otherwise it is 1.

We enforce that the weights $\{w_j\}_{j=1}^d$ are nonnegative, in accordance with our knowledge of how most quality ratings are constructed. If there is a case in which a factor is negatively correlated with rank, then we would simply use the negative of the factor, so that the corresponding weight would be positive. Also, if w^* maximizes (3.2), then so does γw^*, for any constant $\gamma > 0$; thus we can constrain each w_j to be in the interval $[0, 1]$ without loss of generality. The primary purpose of this constraint is to reduce the size of the region of feasible solutions, which is intended to speed up the computation. There is a single parameter $\varepsilon > 0$ that the user specifies. Since increasing ε tends to increase runtimes, as shown in the experimental results of the previous chapter, we choose ε to be 10^{-6}, which is just large enough to be recognized as nonzero by the solver.

After the optimization problem (3.8) is solved for our category of interest, we use the maximizing weights w^* to determine the score $f(x) = w^{*T}x$ of a new product x within the same category.

3.4 Evaluation Metrics

In the case of our rating data, one goal is to predict, for instance, whether a new product that has not yet been rated will be among the top-k products that have already been rated. That is, the training data are included in the assessment of test performance. This type of evaluation is contrary to common machine learning practice in which evaluations on the training and test sets are separate, and thus it is not immediately clear how these evaluations should be performed.

Table 3.1: Notation for evaluation metrics. Note that ζ^s and f_w^s are computed from only the training data.

ζ_i^s	=	true score for product x_i^s (training or test)
ζ^s	=	true score of product in position $\bar{T}_s$ within the training set, where products are ranked according to true scores ζ_i^s
f_w^s	=	model score of product in position $\bar{T}_s$ within the training set, where products are ranked according to model scores $w^T x_i^s$
S^s_{train}	=	$\{i : \text{product } x_i^s \text{ is in the training set}\}$
S^s_{test}	=	$\{j : \text{product } x_j^s \text{ is in the test set}\}$
S^s_{all}	=	$S^s_{\text{train}} \cup S^s_{\text{test}}$
$S^s_{\text{train,top}}$	=	$\{i : i \in S^s_{\text{train}} \text{ and } \zeta_i^s \geq \zeta^s\}$
$S^s_{\text{test,top}}$	=	$\{j : j \in S^s_{\text{test}} \text{ and } \zeta_j^s \geq \zeta^s\}$
$S^s_{\text{all,top}}$	=	$S^s_{\text{train,top}} \cup S^s_{\text{test,top}}$

In this section, we define three measures that are useful for ranking problems in which test predictions are gauged relative to the training set. The measures are first computed separately for each subcategory and then aggregated over the subcategories to produce a concise result. We focus on the top $\bar{T}_s$ products in subcategory s, and use the notation in Table 3.1, where $f(x) = w^T x$ is a given scoring function.

Measure 1: Fraction of correctly ranked pairs among top of ranked list

This is the most useful and important of the three measures because it specifically captures ranking quality at the top of the list. Using the same notation as in (3.8), let $\pi_{ik} = 1$ if $\zeta_i^s > \zeta_k^s$ and 0 otherwise, and $z_{ik} = 1$ if $w^T x_i > w^T x_k$ and 0 otherwise. The evaluation measures for the training and test data are:

$$\text{M1}_{\text{train}}(s) = \frac{\sum_{i,k \in S^s_{\text{train,top}}} \pi_{ik} z_{ik}}{\sum_{i,k \in S^s_{\text{train,top}}} \pi_{ik}},$$

$$\text{M1}_{\text{test}}(s) = \frac{\sum_{i,k \in S^s_{\text{all,top}}} \pi_{ik} z_{ik} - \sum_{i,k \in S^s_{\text{train,top}}} \pi_{ik} z_{ik}}{\sum_{i,k \in S^s_{\text{all,top}}} \pi_{ik} - \sum_{i,k \in S^s_{\text{train,top}}} \pi_{ik}}.$$

The M1 metric does not require the actual values of the true scores ζ_i^s; it suffices to know the pairwise preferences π_{ik}. Note that $\text{M1}_{\text{test}}(s)$ is the fraction of correctly ranked pairs among both training and test products, excluding pairs for which both products are in the training set.

Measure 2: Fraction of correctly ranked pairs over entire ranked list

This measure is similar to Measure 1, except that instead of considering only the top of the ranked list, it considers the entire list.

$$\mathrm{M2}_{\mathrm{train}}(s) = \frac{\sum_{i,k \in S^s_{\mathrm{train}}} \pi_{ik} z_{ik}}{\sum_{i,k \in S^s_{\mathrm{train}}} \pi_{ik}},$$
$$\mathrm{M2}_{\mathrm{test}}(s) = \frac{\sum_{i,k \in S^s_{\mathrm{all}}} \pi_{ik} z_{ik} - \sum_{i,k \in S^s_{\mathrm{train}}} \pi_{ik} z_{ik}}{\sum_{i,k \in S^s_{all}} \pi_{ik} - \sum_{i,k \in S^s_{\mathrm{train}}} \pi_{ik}}.$$

Note that $\mathrm{M2}_{\mathrm{train}}$ is the same as AUC_π in (3.1).

Measure 3: Fraction of correctly classified products

This evaluation metric is the fraction of products that are correctly classified in terms of being among the top of the list:

$$\mathrm{M3}_{\mathrm{train}}(s) = \frac{1}{|S^s_{\mathrm{train}}|} \sum_{i \in S^s_{\mathrm{train}}} \left(\mathbf{1}_{[\zeta^s_i \geq \zeta^s \text{ and } w^T x_i \geq f^s_w]} + \mathbf{1}_{[\zeta^s_i < \zeta^s \text{ and } w^T x_i < f^s_w]} \right),$$
$$\mathrm{M3}_{\mathrm{test}}(s) = \frac{1}{|S^s_{\mathrm{test}}|} \sum_{j \in S^s_{\mathrm{test}}} \left(\mathbf{1}_{[\zeta^s_j \geq \zeta^s \text{ and } w^T x_j \geq f^s_w]} + \mathbf{1}_{[\zeta^s_j < \zeta^s \text{ and } w^T x_j < f^s_w]} \right).$$

Although $\mathrm{M3}_{\mathrm{test}}(s)$ measures quality on the test set, the values ζ^s and f^s_w depend on the true scores and model scores from the training set. If the true scores ζ^s_i are not available, then it suffices to know the rank of each product relative to the product in position $\bar{T}_s$ in the training set in order to compute this metric.

Aggregation of measures

To produce a single numerical evaluation for each of the three measures, we aggregate by taking a weighted sum of the measures over subcategories in a given category, where the weights are proportional to the sizes of the subcategories. The three evaluation measures defined above all have the form: $\mathrm{M}(s) = \frac{\mathrm{numer}(s)}{\mathrm{denom}(s)}$. The version of evaluation measure M aggregated over subcategories for either the training set or the test set is:

$$\mathrm{M} = \frac{\sum_s \mathrm{numer}(s)}{\sum_s \mathrm{denom}(s)} = \frac{\sum_s \mathrm{denom}(s)\mathrm{M}(s)}{\sum_s \mathrm{denom}(s)}.$$

3.5 Other Methods for Reverse-Engineering

We compare our approach with several other methods. The first set of methods are based on least squares regression, and the second set are convex relaxations of the MIO method.

3.5.1 Least Squares Methods for Reverse-Engineering

The organization that provides our rating data currently uses a proprietary method to reverse-engineer the ranking model, the core of which is very similar to least squares regression on the scores. If the scores were not available—for instance, when working with data from a different rating company—the organization would conceivably use least squares regression on the ranks. Thus, our baselines are variations on least squares regression, minimizing:

$$\sum_{s \in S_{\text{sub}}} \frac{C_s}{N_s} \sum_{i=1}^{n_s} (y_i^s - (w_0 + w^T x_i^s))^2,$$

where N_s is the number of products in the category to which subcategory s belongs:

$$N_s = \sum_{r \in \text{cat}(s)} n_r,$$

and y_i^s can be one of three quantities:

1. the true score ζ_i^s for product x_i^s (method LS1),
2. the rank over all training products, that is, the number of training products that are within subcategories r such that $C_r > 0$ and are ranked strictly below x_i^s according to the true scores ζ_i^s (method LS2),
3. the rank within the subcategory, that is, the number of training products in the same subcategory as x_i^s that are ranked strictly below x_i^s according to the true scores ζ_i^s (method LS3).

3.5.2 The ℓ_p Reverse-Engineering Algorithm

As another point of comparison, we introduce a new method called "ℓ_p Reverse-Engineering" (ℓ_pRE) that generalizes the P-Norm Push algorithm for supervised ranking [Rudin, 2009]. This algorithm minimizes an objective with two terms, one that "pushes" low-quality products to the bottom of the list, and another that "pulls" high-quality products to the top. To derive this algorithm, we first consider the following loss function:

$$\text{Loss}_{s,p,\text{low},0-1}(f) := \left(\sum_{k=1}^{n_s}\left(\sum_{i=1}^{n_s}\pi_{ik}^s \mathbf{1}_{[f(x_i^s)\leq f(x_k^s)]}\right)^p\right)^{1/p}.$$

In order to interpret $\text{Loss}_{s,p,\text{low},0-1}(f)$, consider that $\sum_{i=1}^{n_s}\pi_{ik}^s\mathbf{1}_{[f(x_i^s)\leq f(x_k^s)]}$ is the number of products i that should be ranked higher than k (that is, $\pi_{ik}^s = 1$), but are ranked lower by f (that is, $\mathbf{1}_{[f(x_i^s)\leq f(x_k^s)]}$). This quantity is large when k is a low-quality product that is near the top of the ranked list. In other words, the largest terms in the sum $\sum_{k=1}^{n_s}\left(\sum_{i=1}^{n_s}\pi_{ik}^s\mathbf{1}_{[f(x_i^s)\leq f(x_k^s)]}\right)^p$ correspond to low quality products that are highly ranked. Thus, minimizing $\text{Loss}_{s,p,\text{low},0-1}(f)$ tends to "push" low-quality products towards the bottom of the list.

Instead of minimizing $\text{Loss}_{s,p,\text{low},0-1}(f)$ directly, we can minimize the following convex upper bound:

$$\text{Loss}_{s,p,\text{low}}(f) := \left(\sum_{k=1}^{n_s}\left(\sum_{i=1}^{n_s}\pi_{ik}^s e^{-\left(f(x_i^s)-f(x_k^s)\right)}\right)^p\right)^{1/p}.$$

We reverse the sums over i and k to define another quantity:

$$\text{Loss}_{s,p,\text{high}}(f) := \left(\sum_{i=1}^{n_s}\left(\sum_{k=1}^{n_s}\pi_{ik}^s e^{-\left(f(x_i^s)-f(x_k^s)\right)}\right)^p\right)^{1/p}.$$

Minimizing $\text{Loss}_{s,p,\text{high}}(f)$ tends to "pull" high-quality products towards the top of the list. The ℓ_pRE method uses both $\text{Loss}_{s,p,\text{low}}(f)$ and $\text{Loss}_{s,p,\text{high}}(f)$. The loss function

minimized by ℓ_pRE is:

$$\sum_{s \in S_{\text{sub}}} \frac{C_s}{N_{s,p}} \Big(\text{Loss}_{s,p,\text{low}}(f) + C_{\text{high}} \cdot \text{Loss}_{s,p,\text{high}}(f) \Big),$$

where the normalization factor $N_{s,p}$ is:

$$N_{s,p} = \sum_{r \in \text{cat}(s)} \left(\left(\sum_{k=1}^{n_r} \left(\sum_{i=1}^{n_r} \pi_{ik}^r \right)^p \right)^{1/p} + C_{\text{high}} \left(\sum_{i=1}^{n_r} \left(\sum_{k=1}^{n_r} \pi_{ik}^r \right)^p \right)^{1/p} \right),$$

and C_s and C_{high} are user-specified parameters that control the relative importance of each subcategory, and the importance of $\text{Loss}_{s,p,\text{high}}(f)$ relative to $\text{Loss}_{s,p,\text{low}}(f)$ respectively. We use $p = 1$ and $p = 2$, and denote the corresponding methods by ℓ_1RE and ℓ_2RE respectively.

3.6 Proof of Concept

As a preliminary experiment, we tested the methods using an artificial dataset.[1] Figure 3-1 shows for each of the five factors of this dataset, a scatterplot of the factor values versus the scores. The sixth plot in the figure shows all five factors versus the scores in the same window. For each factor, there is one set of products for which there is perfect correlation between the factor values and scores, another set for which there is perfect negative correlation, and the remainder for which the factor value is constant. By constructing the dataset in this manner, we expect there to be significant variation in the ranking performance of the different methods. This dataset is similar to the ROC Flexibility data from Chapter 2.

There is only one category with one subcategory. There are 200 products total, and we randomly divided the data into 100 products for training and 100 products for testing. We tested five methods: LS1, LS2, ℓ_1RE, ℓ_2RE, and MIO-RE; LS3 is equivalent to LS2 since there is only one subcategory.[2] We ran the methods for three

[1] Dataset available at: http://web.mit.edu/rudin/www/ReverseEngineering_Flex_Data.csv.

[2] All least-squares methods were implemented using R 2.8.1, and all ℓ_pRE methods using MATLAB 7.8.0, on an Intel Core 2 Duo 2GHz processor with 1.98GB of RAM. MIO-RE was implemented

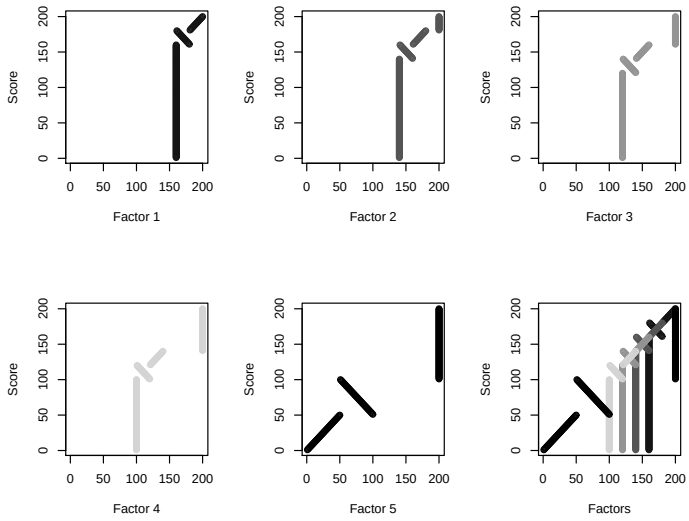


Figure 3-1: Factor values vs. scores for artificial dataset.

Table 3.2: Train and test values for M1, M2, and M3 on artificial dataset (top 60).

Algorithm		**M1**	**M2**	**M3**
LS1, ℓ_1RE, ℓ_2RE	train	0.878	0.912	**0.780**
	test	0.892	0.909	**0.770**
LS2	train	0.909	0.923	**0.780**
	test	0.915	0.918	**0.770**
MIO-RE	train	**0.925**	**0.928**	**0.780**
	test	**0.943**	**0.929**	**0.770**

cases: concentrating on the top 60, the top 45, and the top 25, that is, $T = 40$, $T = 55$, and $T = 75$ respectively. We ran ℓ_1RE with $C_{\text{high}} = 0$; ℓ_2RE with $C_{\text{high}} = 0, 0.5$, and 1; and MIO-RE with $\theta = 9$. MIO-RE found the final solutions within three minutes for each case, and the other methods ran within seconds. Tables 3.2, 3.3, and 3.4 show the results. The highest training and test measures across the methods are highlighted in bold. LS1, ℓ_1RE, and ℓ_2RE (with $C_{\text{high}} = 0, 0.5$, and 1) always produced the same values for the three evaluation measures.

The methods all performed similarly according to the classification measure M3. MIO-RE had a significant advantage with respect to M2, regardless of the threshold we used for top of the list (top 60 in Table 3.2, top 45 in Table 3.3, or top 25 in

using ILOG AMPL 11.210 with the Gurobi 3.0.0 solver on two Intel quad core Xeon E5440 2.83GHz processors with 32GB of RAM. We always used $\varepsilon = 10^{-6}$ for MIO-RE.

Table 3.3: Train and test values for M1, M2, and M3 on artificial dataset (top 45).

Algorithm		**M1**	**M2**	**M3**
LS1, ℓ_1RE, ℓ_2RE	train	0.880	0.912	**0.920**
	test	0.898	0.909	**0.930**
LS2	train	0.935	0.923	**0.920**
	test	0.942	0.918	**0.930**
MIO-RE	train	**0.964**	**0.928**	**0.920**
	test	**0.994**	**0.929**	**0.930**

Table 3.4: Train and test values for M1, M2, and M3 on artificial dataset (top 25).

Algorithm		**M1**	**M2**	**M3**
LS1, ℓ_1RE, ℓ_2RE	train	0.907	0.912	**1.000**
	test	0.899	0.909	0.980
LS2	train	0.907	0.923	**1.000**
	test	0.899	0.918	0.980
MIO-RE	train	**1.000**	**0.928**	**1.000**
	test	**1.000**	**0.929**	**1.000**

Table 3.4). For M1, MIO-RE performed substantially better than the others, and its advantage over the other methods was more pronounced as the evaluation measure concentrated more on the top of the list. One can see this by comparing the M1 column in Tables 3.2, 3.3, and 3.4. In Table 3.4, MIO-RE performed better than the other methods by 10.3% on training and 11.3% on testing. Using exact optimization rather than approximations, the MIO-RE method was able to find solutions that none of the other methods could find. This study demonstrates the potential of MIO-RE to substantially outperform other methods.

3.7 Experiments on Rating Data

For our main experiments, the dataset contains approximately a decade's worth of rating data from a major rating company, compiled by an organization that is aiming to reverse-engineer the ranking model. The values of the factors are discretized versions of the true values. The rating company periodically makes ratings for new products available, and our goal is to predict, with respect to the products that are already rated: where each new product is within the top-k (M1), where it is in the

full list, even if not in the top-k (M2), and whether each new product falls within the top-k (M3). We generate a scoring function for one category, "Category A," regularizing with data from "Category B." Category A has eight subcategories with a current total of 209 products, and Category B has eight subcategories with a total of 212 products. There are 19 factors.

The size of the dataset is small and thus challenging to deal with from a machine learning perspective. The small size causes problems with accurate reverse-engineering in training and evaluating generalization ability in testing. That is, for all algorithms, the variance of the test evaluation measures is high compared to the difference in training performance. The worst performing algorithm in training sometimes has the best test performance, and vice versa. We aim to determine whether MIO-RE has consistently good performance, compared to other algorithms that sometimes perform very poorly.

3.7.1 Experimental Setup

For this set of experiments, we divided the data for Category A into four folds, and used each fold in turn as the test set. The first fold had 53 products, and the other three folds each had 52 products. Our experiment was as follows, where M1, M2, and M3 refer to the three aggregate evaluation measures, computed using just data from Category A and not Category B, though data from both categories were used for training:

1. For each set of parameters, perform three-fold cross-validation using the first three folds as follows:

    a. Train using Folds 1 and 2, and Category B, and validate using Fold 3. Compute M1, M2, and M3 for training and validation.

    b. Train using Folds 1 and 3, and Category B, and validate using Fold 2. Compute M1, M2, and M3 for training and validation.

    c. Train using Folds 2 and 3, and Category B, and validate using Fold 1. Compute M1, M2, and M3 for training and validation.

d. Compute the average over the three folds of the training and validation values for each of M1, M2, and M3.

Note that when we compute M1, M2, and M3 on validation data, this also takes into account the training data, as in Section 3.4.

2. Sum the three average validation measures, and choose the parameters corresponding to the largest sum.
3. Train using Folds 1, 2, and 3, and Category B, together with the parameters chosen in the previous step, and test using Fold 4. Compute M1, M2, and M3 for training and testing.
4. Repeat steps 1 through 3 using Folds 1, 2, and 4 for cross-validation and Fold 3 for the final test set.
5. Repeat steps 1 through 3 using Folds 1, 3, and 4 for cross-validation and Fold 2 for the final test set.
6. Repeat steps 1 through 3 using Folds 2, 3, and 4 for cross-validation and Fold 1 for the final test set.

We followed this experimental procedure for each algorithm, repeating the same steps four times to avoid the possibility that by chance our results would be good or bad because of our choice of training data.

For all algorithms, we set $C_s = 1$ for all subcategories s in Category A. The regularization parameter $C_s = C$ for all subcategories s in Category B varied for each method in a range such that the contribution in the objective function from Category B was smaller than the contribution from Category A. Table 3.5 shows the different parameter values tested for each algorithm. For ℓ_1RE, the two terms of the objective function are identical, so we chose C_{high} to be 0. For ℓ_2RE, we chose C_{high} to be 0, 0.5, or 1. For MIO-RE, we chose θ to be 0 or 9, so that the top of the list was weighed by a factor of 1 or 10 respectively.

In total, for the cross-validation step, there were $6 \times 3 = 18$ problems to solve for LS1, LS2, and LS3; $6 \times 2 = 12$ problems for ℓ_1RE, $6 \times 2 \times 3 = 36$ problems for

Table 3.5: Parameter values tested for each algorithm.

Algorithm	**Parameter1**	**Parameter2**
LS1	C=0, 0.1, or 0.2	
LS2	C=0, 0.1, or 0.2	
LS3	C=0, 0.025, or 0.05	
ℓ_1RE	C=0 or 0.1	C_{high}=0
ℓ_2RE	C=0 or 0.1	C_{high}=0, 0.5, or 1
MIO-RE	C=0 or 0.5	θ=0 or 9

ℓ_2RE, and $6 \times 2 \times 2 = 24$ problems for MIO-RE. (For each method, the total number of problems was the number of different parameter settings times six, which is the number of ways to choose two out of four folds for training.) For the test step, there were an additional four problems for each method. This set of experiments required approximately 163 hours of computation time.

3.7.2 Results

There are four rounds of the experiment in which we train on three folds and test on the fourth fold (step 3 in the procedure above), with the parameter values found through cross-validation. Tables 3.6 through 3.9 show the training and test values of M1, M2, and M3 in each of these four rounds. The highest training and test measures are highlighted in bold. The integer number next to each measure is the rank of the method, that is, the number of other methods below it for the particular measure and dataset (training or test). Note that 0 is the lowest possible rank by this definition.

Figure 3-2 is a visualization of Tables 3.6 through 3.9 and shows barplots of M1, M2, and M3 from each of the four rounds; note that for each algorithm, the bars for the three measures have been stacked for compactness. The bar heights are relative instead of absolute; for example, the bar heights for the dark bars (M1) in the top left plot were computed as follows:

1. Let M1_m be the value of M1 for method m, where m is either LS1, LS2, LS3, ℓ_1RE, ℓ_2RE, or MIO-RE. Note that these are the M1 values from training on Folds 1, 2, and 3.

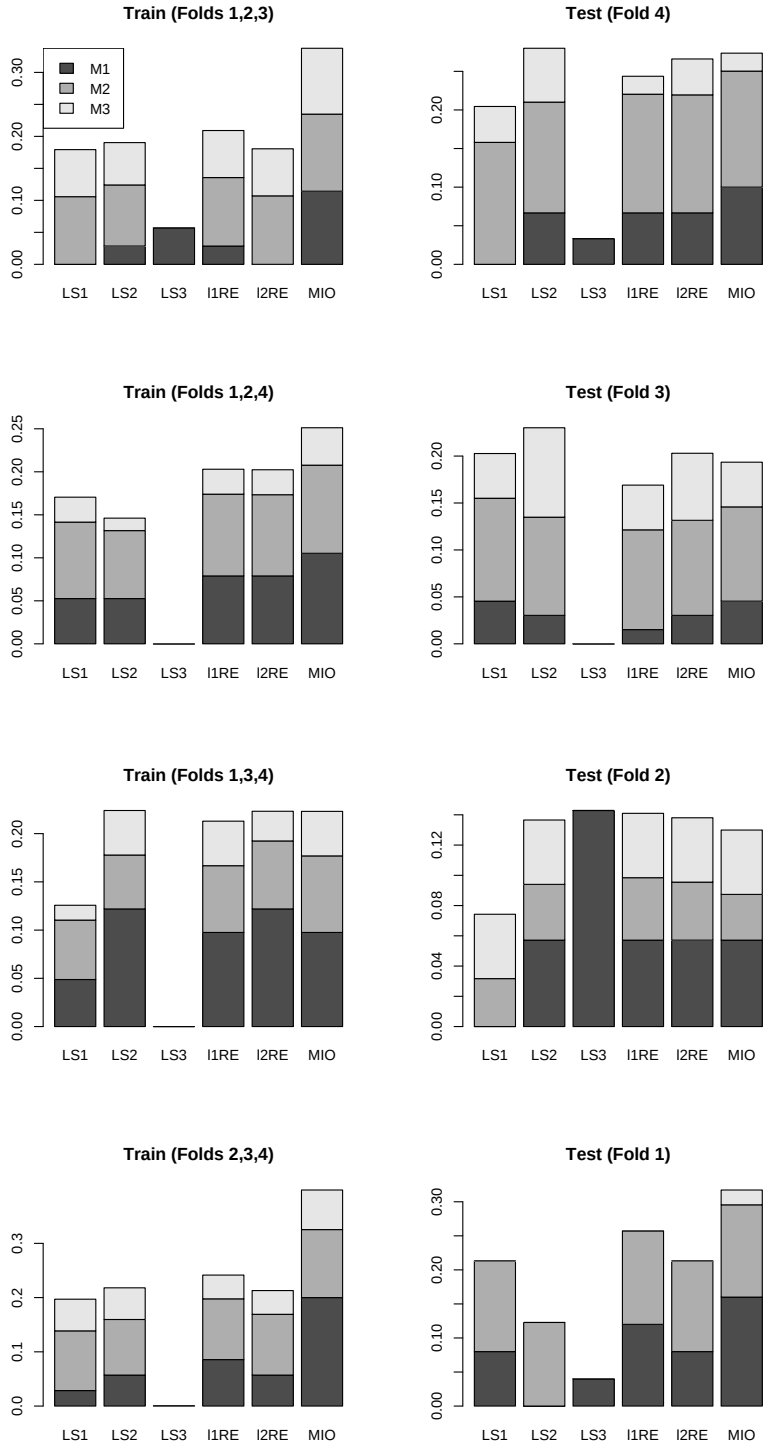


Figure 3-2: Barplot summary of results from four rounds of training on three folds and testing on the fourth: M1 (dark), M2 (medium), and M3 (light).

Table 3.6: Training and test values of M1, M2, and M3 on ratings data, and ranks of algorithms (train on Folds 1, 2, and 3; test on Fold 4).

Algorithm		**M1**		**M2**		**M3**	
LS1	train	0.686	0	0.920	2	0.930	2
$C = 0$	test	0.714	0	**0.922**	5	0.865	3
LS2	train	0.706	2	0.911	1	0.924	1
$C = 0.1$	test	0.762	2	0.911	1	**0.885**	5
LS3	train	0.725	4	0.832	0	0.866	0
$C = 0.05$	test	0.738	1	0.797	0	0.827	0
ℓ_1RE	train	0.706	2	0.921	3	0.930	2
$C = 0.1$, $C_{\text{high}} = 0$	test	0.762	2	0.919	4	0.846	1
ℓ_2RE	train	0.686	0	0.921	3	0.930	2
$C = 0.1$, $C_{\text{high}} = 1$	test	0.762	2	0.918	3	0.865	3
MIO-RE	train	**0.765**	5	**0.932**	5	**0.955**	5
$C = 0.5, \theta = 0$	test	**0.786**	5	0.916	2	0.846	1

2. Let $\text{M1}_{\min}$ be the minimum of the six M1_m values.

3. The bar height for method m is the percentage increase of M1_m from $\text{M1}_{\min}$:

$$\frac{\text{M1}_m - \text{M1}_{\min}}{\text{M1}_{\min}}.$$

The method for which $\text{M1}_m = \text{M1}_{\min}$ has bar height 0. The other bar heights were computed similarly; for each measure, there is at least one method for which the bar height is 0. Thus it is easy from the figure to see, within each barplot, the relative magnitudes of the three measures across all algorithms. For instance, in the top left barplot, MIO-RE clearly is largest in terms of dark bars (M1) and light bars (M3), though it is about the same as all other algorithms in terms of medium bars (M2).

As stated in Section 3.4, we are most interested in M1, which measures ranking quality at the top of the list. Figure 3-2 shows that with respect to M1, though not always the best, MIO-RE performed consistently well for both training and testing. In contrast, the other algorithms may have performed well for some training or test cases but also performed poorly for other cases. Table 3.10 shows just the M1 metric from Tables 3.6 through 3.9, averaged for each algorithm over the four rounds. MIO-RE has a clear advantage over the other methods according to these sums.

Table 3.7: Training and test values of M1, M2, and M3 on ratings data, and ranks of algorithms (train on Folds 1, 2, and 4; test on Fold 3).

Algorithm		**M1**		**M2**		**M3**	
LS1	train	0.833	1	0.925	2	0.904	2
$C = 0$	test	**0.784**	4	**0.922**	5	0.846	1
LS2	train	0.833	1	0.917	1	0.892	1
$C = 0.2$	test	0.773	2	0.918	3	**0.885**	5
LS3	train	0.792	0	0.850	0	0.879	0
$C = 0.05$	test	0.750	0	0.831	0	0.808	0
ℓ_1RE	train	0.854	3	0.930	4	0.904	2
$C = 0.1$, $C_{\text{high}} = 0$	test	0.761	1	0.919	4	0.846	1
ℓ_2RE	train	0.854	3	0.930	3	0.904	2
$C = 0$, $C_{\text{high}} = 0$	test	0.773	2	0.915	2	0.865	4
MIO-RE	train	**0.875**	5	**0.937**	5	**0.917**	5
$C = 0.5, \theta = 0$	test	**0.784**	4	0.914	1	0.846	1

Table 3.8: Training and test values of M1, M2, and M3 on ratings data, and ranks of algorithms (train on Folds 1, 3, and 4; test on Fold 2).

Algorithm		**M1**		**M2**		**M3**	
LS1	train	0.843	1	0.913	2	0.841	1
$C = 0.1$	test	0.778	0	0.925	2	**0.942**	1
LS2	train	**0.902**	4	0.908	1	**0.866**	3
$C = 0$	test	0.822	1	0.929	3	**0.942**	1
LS3	train	0.804	0	0.860	0	0.828	0
$C = 0.05$	test	**0.889**	5	0.896	0	0.904	0
ℓ_1RE	train	0.882	2	0.919	3	**0.866**	3
$C = 0.1$, $C_{\text{high}} = 0$	test	0.822	1	**0.933**	5	**0.942**	1
ℓ_2RE	train	**0.902**	4	0.920	4	0.854	2
$C = 0.1$, $C_{\text{high}} = 1$	test	0.822	1	0.931	4	**0.942**	1
MIO-RE	train	0.882	2	**0.928**	5	**0.866**	3
$C = 0.5, \theta = 0$	test	0.822	1	0.923	1	**0.942**	1

Table 3.9: Training and test values of M1, M2, and M3 on ratings data, and ranks of algorithms (train on Folds 2, 3, and 4; test on Fold 1).

Algorithm		**M1**		**M2**		**M3**	
LS1	train	0.706	1	0.932	2	0.929	3
$C = 0$	test	0.900	2	0.902	2	0.868	0
LS2	train	0.725	2	0.925	1	0.929	3
$C = 0.2$	test	0.833	0	0.894	1	0.868	0
LS3	train	0.686	0	0.839	0	0.878	0
$C = 0.05$	test	0.867	1	0.796	0	0.868	0
ℓ_1RE	train	0.745	4	0.933	3	0.917	1
$C = 0.1$, $C_{\text{high}} = 0$	test	0.933	4	**0.906**	5	0.868	0
ℓ_2RE	train	0.725	2	0.933	3	0.917	1
$C = 0.1$, $C_{\text{high}} = 0.5$	test	0.900	2	0.902	2	0.868	0
MIO-RE	train	**0.824**	5	**0.944**	5	**0.942**	5
$C = 0.5, \theta = 0$	test	**0.967**	5	0.904	4	**0.887**	5

Table 3.10: Average of M1 metric over four rounds for each algorithm.

Algorithm	**M1 (train)**	**M1 (test)**
LS1	0.767	0.794
LS2	0.792	0.798
LS3	0.752	0.811
ℓ_1RE	0.797	0.820
ℓ_2RE	0.792	0.814
MIO-RE	**0.836**	**0.840**

To view the results in a nonparametric way, we use the ranks in Tables 3.6 through 3.9. There are four sets of ranks corresponding to the four rounds of training and testing. In Table 3.11, we sum up the ranks over the four rounds. The consistently high performance of MIO-RE is also reflected in this table, particularly in its advantage in terms of training and testing for M1.

Note that LS1 has an inherent advantage over the other five methods in that it uses information—namely the true scores—that is not available to the other methods that use only the ranks. As discussed earlier, in many cases the true scores may not be available if the rating company does not provide them. Even if the scores are available, our experiment demonstrates that it is possible for methods that encode only the ranks, such as MIO-RE, to have comparable or better performance than methods that directly use the scores. For example, in all but the third round of

Table 3.11: Sums of ranks over four rounds for each algorithm.

		LS3	**LS1**	**LS2**	**ℓ_2RE**	**ℓ_1RE**	**MIO-RE**
Train	M1	4	3	9	9	11	17
	M2	0	8	4	13	13	20
	M3	0	8	8	7	8	18
Total		**4**	**19**	**21**	**29**	**32**	**55**
Test	M1	7	6	5	7	8	15
	M2	0	14	8	11	18	8
	M3	0	5	11	8	3	8
Total		**7**	**25**	**24**	**26**	**29**	**31**

our experiment, it appears that there was a particularly good solution that none of the approximate methods found, but that MIO-RE did, similar to the results in Section 3.6. This is the major advantage of exactly optimizing the objective function rather than using a convex proxy.

3.7.3 Example of Differences Between Methods on Evaluation Measures

It is not immediately clear how a difference in evaluation measures corresponds to differences between ranked lists in our experiment. To illustrate this, we directly compare ranked lists corresponding to the test set in the fourth round (train on Folds 2, 3, and 4; test on Fold 1). The ranked lists shown in Table 3.12 were generated by scoring the products using MIO-RE and LS3, and are divided into the eight subcategories in Category A. For confidentiality purposes, the actual product names have been replaced by the names of various wineries in eight different regions of California.[3]

As indicated by the test measures, reproduced in Table 3.13, MIO-RE and LS3 were comparable in terms of correctly classifying products as either in the top or not in the top (M3). However, MIO-RE performed much better in terms of pairwise rankings (M1 and M2). For example, MIO-RE correctly ranked all products in the Lake County subcategory while LS3 switched the first and third products; MIO-RE switched the first two products in the Southern California subcategory while LS3 also

[3]http://www.cawinemall.com/region.shtml

Table 3.12: Example of ranked lists produced by different algorithms, corresponding to metrics in Table 3.13.

True	MIO-RE	LS3
	LakeCounty	
Brassfield	Brassfield	Wildhurst
Langtry	Langtry	Langtry
Wildhurst	Wildhurst	Brassfield
	NorthCoast	
Alpen	Alpen	Alpen
Fieldbrook	Fieldbrook	Fieldbrook
Winnett	Winnett	Winnett
	SouthCali	
Faulkner	Lenora	Lenora
Lenora	Faulkner	Faulkner
Peralta	Peralta	Peralta
Salerno	Salerno	Thompkin
Thompkin	Thompkin	Salerno
	Mendocino	
Baxter	Navarro	Navarro
Goldeneye	Baxter	Baxter
Navarro	Goldeneye	Goldeneye
Skylark	Skylark	Skylark
	CentralCoast	
Blackstone	Blackstone	Morgan
Estancia	Estancia	Blackstone
Jenkins	Morgan	Ronan
Morgan	Parsonage	Estancia
Newell	Newell	Ventana
Parsonage	Jenkins	Jenkins
Ronan	Ronan	Newell
Ventana	Ventana	Parsonage

True	MIO-RE	LS3
	CentralVal	
Accardi	Accardi	Accardi
Baywood	Baywood	Mariposa
Cantiga	Mariposa	Trimble
Harmony	Cantiga	Harmony
Mariposa	Omega	Cantiga
Omega	Watts	Omega
Trimble	Harmony	Watts
Watts	Trimble	Baywood
	SierraFoot	
Auriga	Auriga	Auriga
Chevalier	Chevalier	Paravi
Dillian	Paravi	Chevalier
Fitzpatrick	Dillian	Solomon
Hatcher	Fitzpatrick	Oakstone
Montevina	Hatcher	Hatcher
Oakstone	Montevina	Fitzpatrick
Paravi	Oakstone	Dillian
Renwood	Solomon	Renwood
Solomon	Renwood	Montevina
Venezio	Venezio	Venezio
	NapaValley	
Carter	Falcor	Falcor
Falcor	Carter	Carter
Ilsley	Ilsley	Kelham
Kelham	Kelham	Ilsley
Mason	Mason	Mason
Oberon	Oberon	Oberon
Quintessa	Relic	Quintessa
Relic	Quintessa	Trefethen
Sawyer	Sawyer	Relic
Trefethen	Varozza	Sawyer
Varozza	Trefethen	Varozza

Table 3.13: Comparison of MIO-RE and LS3 (train on Folds 2, 3, and 4; test on Fold 1), corresponding to ranked lists in Table 3.12.

Algorithm	**M1**	**M2**	**M3**
MIO-RE	0.967	0.904	0.887
LS3	0.867	0.796	0.868

switched the last two; and the MIO-RE rankings for the Central Valley subcategory were not inaccurate by more than three places for any product while LS3 ranked the second product in the eighth position and the eighth product in the third position. There are several other differences between the ranked lists that help to explain the differences in the evaluation measures.

3.8 Determining a Cost-Effective Way to Achieve Top Rankings

Having reverse-engineered the ranking model, it is useful to investigate the following: given a current product x, how can its features be cost-effectively modified so that the new product achieves a top ranking? For instance, suppose we would like to find the most cost-effective way to achieve a top ranking point-and-shoot digital camera. In particular, let there be L ways to change a current product, where multiple changes could potentially be made simultaneously. For example, we can change a current digital camera by enlarging the battery and by making it out of heavier material. Let the decision variable α_ℓ encode whether change ℓ is implemented. The α_ℓ are binary, that is, either the change is implemented or not:

$$\alpha_\ell = \begin{cases} 1, & \text{if change } \ell \text{ is implemented,} \\ 0, & \text{otherwise.} \end{cases}$$

If change ℓ is implemented, then there is an associated cost, denoted c_ℓ, and factor j of product x will increase by an amount $\delta_{j\ell}(x)$:

$$\alpha_\ell = 1 \quad \Longrightarrow \quad x_j \leftarrow x_j + \delta_{j\ell}(x).$$

It is possible that implementing change ℓ can affect more than one factor. Making a digital camera out of heavier material affects its weight and perhaps also its ability to handle shake, for example. Moreover, some of the $\delta_{j\ell}$ values and costs may be

negative, as the most cost-effective way to increase the ranking of a product may be to decrease some factors while increasing others. That is, it might be economical to spend less on one factor and instead fund another change that contributes more to increasing the score. The total change in factor j of product x is

$$\sum_{\ell=1}^{L} \alpha_\ell \delta_{j\ell}(x).$$

There may be possible changes that conflict with each other, and we take this into account as follows: let there be M index sets of changes where at most one of these changes is allowed, and let S_m denote the m^{th} set. Then we have the exclusivity constraints

$$\sum_{\ell \in S_m} \alpha_\ell \leq 1, \quad \forall m = 1, \ldots, M.$$

For instance, we cannot increase a camera's resolution both by one megapixel and by two megapixels; at most one of these two changes can occur.

Let the current score of product x be

$$v_0(x) = w^T x = \sum_{j=1}^{d} w_j x_j.$$

For a given vector of changes $\alpha \in \{0,1\}^L$, the new score of product x after the changes are made is

$$\begin{aligned} v_{\text{new}}(x) &= \sum_{j=1}^{d} w_j \left(x_j + \sum_{\ell=1}^{L} \alpha_\ell \delta_{j\ell}(x) \right) = \sum_{j=1}^{d} w_j x_j + \sum_{j=1}^{d} w_j \left(\sum_{\ell=1}^{L} \alpha_\ell \delta_{j\ell}(x) \right) \\ &= v_0(x) + \sum_{j=1}^{d} w_j \sum_{\ell=1}^{L} \alpha_\ell \delta_{j\ell}(x) = v_0(x) + \sum_{\ell=1}^{L} \alpha_\ell \sum_{j=1}^{d} w_j \delta_{j\ell}(x) \\ &= v_0(x) + \sum_{\ell=1}^{L} \alpha_\ell W_\ell(x), \end{aligned}$$

where $W_\ell(x) = \sum_{j=1}^{d} w_j \delta_{j\ell}(x)$. Note that $W_\ell(x)$ is the change in score that would

result from making change ℓ. Then

$$v_{\text{diff}}(x) = v_{\text{new}}(x) - v_0(x) = \sum_{\ell=1}^{L} \alpha_\ell W_\ell(x)$$

is the total score difference. The total cost associated with the changes in α is

$$c_{\text{diff}}(\alpha) = \sum_{\ell=1}^{L} c_\ell \alpha_\ell.$$

The cost trades off with the change in score. In what follows, we show how to both maximize the change in score on a fixed budget, and how to minimize the cost to achieve a certain change in score.

3.8.1 Two Formulations

Maximizing score on a fixed budget: The first problem is to fix the budget for making changes and maximize the new score of product x, which is equivalent to maximizing v_{diff}. That is, we want to maximize $\sum_{\ell=1}^{L} \alpha_\ell W_\ell(x)$ while not exceeding some bound on the cost, denoted $\bar{c}$. The integer optimization formulation to solve this problem is given by:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{\ell=1}^{L} \alpha_\ell W_\ell(x) \\ \text{s.t.} \quad & \sum_{\ell=1}^{L} c_\ell \alpha_\ell \leq \bar{c}, \\ & \sum_{\ell \in S_m} \alpha_\ell \leq 1, \quad \forall m = 1, \ldots, M, \\ & \alpha_\ell \in \{0, 1\}, \quad \forall \ell = 1, \ldots, L. \end{aligned} \tag{3.13}$$

Minimizing cost with a fixed target score: Suppose the target score is v_{tar}, so that the desired score difference is $v^*_{\text{diff}} = v_{\text{tar}} - v_0(x)$. The integer optimization

Table 3.14: Point-and-shoot digital camera factors.

1	2	3	4	5
Resolution	Weight	Photo Quality	Video Quality	Response Time
6	7	8	9	10
Handling Shake	Versatility	LCD Quality	Widest Angle	Battery Life

formulation is given by:

$$\begin{aligned} \min_{\alpha} \quad & \sum_{\ell=1}^{L} c_\ell \alpha_\ell \\ \text{s.t.} \quad & \sum_{\ell=1}^{L} \alpha_\ell W_\ell(x) \geq v^*_{\text{diff}}, \\ & \sum_{\ell \in S_m} \alpha_\ell \leq 1, \quad \forall m = 1, \ldots, M, \\ & \alpha_\ell \in \{0, 1\}, \quad \forall \ell = 1, \ldots, L. \end{aligned} \tag{3.14}$$

By solving the first formulation for a range of budgets, or by solving the second formulation for a range of target scores, we can map out an efficient frontier of maximum score for minimum cost. This concept is best explained through the following example.

3.8.2 Fictitious Example

We use the example of finding the most cost-effective way to increase the rank of a point-and-shoot digital camera. The data are fictitious. There are 10 factors, shown in Table 3.14. Resolution is in number of megapixels, weight is in ounces, widest angle is in millimeters, and battery life is in number of shots. All other factors take values between 1 and 5, in increments of 0.5, with 1 representing poor quality and 5 representing excellent quality. Let the coefficients of the scoring function $f(x) = w^T x$ be as shown in Table 3.15. The coefficient corresponding to camera weight is negative since it is desirable to have a lighter camera. Table 3.16 shows the scores of two different cameras according to this scoring function.

There are twelve possible changes that we can make to a particular hypothetical

Table 3.15: Coefficients of scoring function for digital cameras.

w_1	w_2	w_3	w_4	w_5
0.5842	−0.5706	4.3421	2.9256	3.7692
w_6	w_7	w_8	w_9	w_{10}
1.1374	1.4423	2.8960	0.0054	0.0006

Table 3.16: Scores of two example cameras.

Camera	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	Score
1	14	5	5	5	5	5	5	5	35	500	88.38
2	12	5	4	4	4	3	4	4	30	300	69.41

digital camera x. Table 3.17 shows the cost of making each change in dollars per camera, as well as the effect $\delta_{j\ell}$ each change ℓ has on factor j. A dot indicates the effect is 0. Table 3.17 does not apply to all cameras, and the $\delta_{j\ell}$'s might need to be constructed individually for each camera. In particular, we assume that none of the six integer factors of camera x would exceed the upper bound of 5 if any of the changes were implemented. For instance, x could not be the first camera in Table 3.16 since factors 2 through 8 are already at their maximum possible value, but it could be the second.

Table 3.18 shows the conflict sets S_m. For instance, the changes "Add 1 Megapixel" (change 2) and "Add 2 Megapixels" (change 6) are mutually exclusive. These conflicts are incorporated in (3.13) and (3.14) in the exclusivity constraints. We represent the conflict between changes 2 and 6 as

$$\alpha_2 + \alpha_6 \leq 1, \quad \text{or} \quad \sum_{\ell \in S_1} \alpha_\ell \leq 1,$$

where $S_1 = \{2, 6\}$. Table 3.19 gives an alternative way to represent the conflicts and shows for each of the twelve changes, which of the other changes conflict with it.

The points in Figures 3-3 and 3-4 correspond to the 512 feasible changes or combinations of changes. The coordinates of each point indicate its cost and effect on the score. We can trace out a frontier of solutions that lead to maximum changes in score for minimum cost. For example, suppose that we fix the maximum cost at 7. Figure 3-3 shows that for a cost of 7, the maximum difference in score is 5.097, which

Table 3.17: Change information for a digital camera.

	Change	δ_1	δ_2	δ_3	δ_4	δ_5	δ_6	δ_7	δ_8	δ_9	δ_{10}	Cost
1	Larger Battery	·	·	·	·	·	·	·	·	·	50	2
2	Add 1 Megapixel	1	·	·	·	·	·	·	·	·	·	3
3	Better LCD	·	·	·	·	·	·	·	0.5	·	·	4
4	More Modes	·	·	·	·	·	·	1	·	·	·	4
5	Wider Angle	·	·	0.5	·	·	·	·	·	2	·	5
6	Add 2 Megapixels	2	·	0.5	·	·	·	·	·	·	·	5
7	Heavier Material	·	1	·	·	·	1	·	·	·	·	5
8	Better Video	·	·	·	1	·	·	·	·	·	·	6
9	Faster Response	·	·	·	·	0.5	·	·	·	·	·	6
10	Better Lens	·	·	0.5	1	·	·	·	·	·	·	7
11	Fastest Response	·	·	·	·	0.5	1	·	·	·	·	7
12	Most Modes	·	·	1	·	0.5	·	1	·	·	·	9

Table 3.18: Conflict sets ($M = 6$).

m	S_m
1	$\{2, 6\}$
2	$\{5, 6, 10, 12\}$
3	$\{8, 10\}$
4	$\{9, 11, 12\}$
5	$\{7, 11\}$
6	$\{4, 12\}$

Table 3.19: Conflicts between changes.

	Change	Conflicts
1	Larger Battery	·
2	Add 1 Megapixel	6
3	Better LCD	·
4	More Modes	12
5	Wider Angle	6, 10, 12
6	Add 2 Megapixels	2, 5, 10, 12
7	Heavier Material	11
8	Better Video	10
9	Faster Response	11, 12
10	Better Lens	5, 6, 8, 12
11	Fastest Response	7, 9, 12
12	Most Modes	4, 5, 6, 9, 10, 11

corresponds to the single change "Better Lens." Note that for a maximum cost of 8, the best solution stays the same. That is, even if we were willing to spend up to 8, the maximum difference in score would be achieved by the same solution as if we were willing to spend only up to 7. These results address the first problem in Section 3.8.1. Figure 3-4 addresses the second problem in Section 3.8.1. For instance, suppose that we specify that the difference in score is at least 2. There are two ways to achieve this difference with the minimum cost of 5, namely by the changes "Wider Angle," which corresponds to an actual score difference of 2.182, or "Add 2 Megapixels," which corresponds to a higher score difference of 3.339.

For large datasets, (3.13) and (3.14) provide an efficient way to generate the

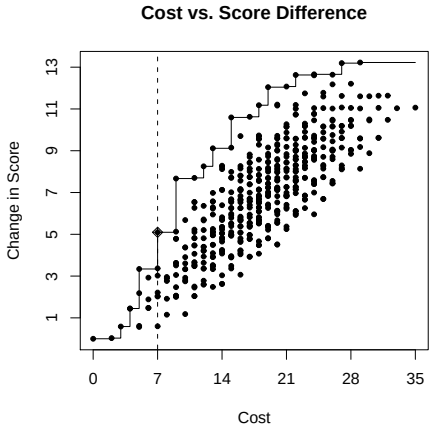


Figure 3-3: If we fix the maximum allowed cost at 7 (dotted line), then the highest possible change in score is 5.097 (one optimum, indicated by a diamond).

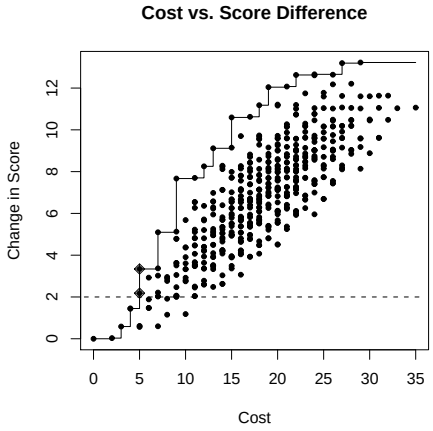


Figure 3-4: If we fix the minimum allowed change in score at 2 (dotted line), then the lowest possible cost is 5 (two optima, indicated by diamonds).

frontier without having to enumerate all possible solutions as we did in Figures 3-3 and 3-4. There may be multiple optima, but it is straightforward to find them by iteratively solving (3.13) or (3.14), and adding a constraint in each iteration that makes the previous optimum infeasible, until the optimal cost changes.

3.9 Summary

We have presented a new approach to reverse-engineering ranking models. The formulation encodes a specific preference structure and categorical organization of the products. Another contribution of our work is the introduction of evaluation measures that take into account the rank of a new product relative to the products that have already been ranked. Finally, we showed how to use a reverse-engineered ranking model to find a cost-effective means of modifying a current product so that the modified product achieves a high rank.

Chapter 4

MIO for Associative Classification

Our goal in this chapter is to develop classification models that are on par in terms of accuracy with the top classification algorithms, yet are interpretable, or easily understood, by humans. This work thus addresses a dichotomy in the current state-of-the-art for classification: On the one hand, there are algorithms such as support vector machines (SVM) [Vapnik, 1995] that are highly accurate but not interpretable; for instance, trying to explain a support vector kernel to a medical doctor is not likely to persuade him to use an SVM-based diagnostic system. On the other hand, there are algorithms such as decision trees (CART) [Breiman et al., 1984] that are highly interpretable but not as accurate; the popularity of decision trees is primarily due to their intutiveness.

Our models are designed to be interpretable from multiple perspectives. First, the models are designed to be *convincing*: for each prediction, the algorithm also provides the reasons for why this particular prediction was made, highlighting exactly which data were used to make that prediction. To achieve this, we use "association rules" to build the models into "decision lists," that is, ordered sets of rules. The second way our models are interpretable involves their size: these models are designed to be *concise*. Specifically, our formulations include two types of regularization. The first encourages rules to have small left-hand-sides, so that the reasons given for each prediction are as sparse as possible. The second encourages the decision list to be shorter. That is, the regularization essentially pulls the default rule (the rule that

applies if none of the rules above it apply) as high as possible in the list. There is no single correct way to measure interpretability, as it is necessarily subjective. Nevertheless, psychologists have long studied human ability to process data, and have shown that humans can simultaneously process only a handful of cognitive entities, and are able to estimate relatedness of only a few variables at a time [e.g. Miller, 1956, Jennings et al., 1982]. We aim in this work to construct a convincing and concise model that captures relationships between variables, which limits the reasoning required by humans to understand and believe its predictions. These models allow predictions to more easily be communicated in words, rather than in equations.

The principal methodology we use in this work is mixed integer optimization (MIO), which helps our classification algorithm achieve high accuracy. Rule learning problems suffer from combinatorial explosion, in terms of both searching through a database for rules and managing a massive pile of potentially interesting rules. A dataset with even a modest number of items can contain thousands of rules, thus making it difficult to find useful ones. Moreover, for a set of L rules, there are $L!$ ways to order them into a decision list. On the other hand, MIO solvers are designed *precisely* to handle combinatorial problems, and the application of MIO to rule learning problems is reasonable given the discrete nature of rules. We create MIO formulations for both the problem of mining rules and the problem of learning to rank them, and our experiments show predictive accuracy on a collection of datasets at approximately the same level as some of the top current algorithms in machine learning, including support vector machines with Gaussian kernels and boosted decision trees.

In Section 4.1, we discuss related work. In Section 4.2, we state our notation and derive MIO formulations for association rule mining. In Section 4.3, we present a learning algorithm, also an MIO formulation, that uses the generated rules to build a classifier. In Section 4.4, we show results on classification accuracy, and in Section 4.5, we demonstrate the interpretability of our classifiers. We conclude in Section 4.6.

4.1 Related Work

Association rule mining was introduced by Agrawal et al. [1993] to aid market-basket analysis, the purpose of which was to discover sets of items, or *itemsets*, that were often purchased together, such as the well-known (though probably fictitious) correlation between sales of beer and diapers [Büchter and Wirth, 1998]. To help increase store profit and customer satisfaction, these easy-to-understand patterns could be used to guide the management of store layout, customer segmentation, and items for sale. Consider the rule $\{i, j\} \Rightarrow k$, where $s\%$ of customers purchased items i, j, and k; and $c\%$ of customers who purchased items i and j also purchased item k. In this case, $\{i, j\}$ is the *body* of the rule, k is the *head*, s is the *support*, and c is the *confidence*. In general, the most challenging part of rule mining is to first generate all itemsets with support exceeding a specified threshold, called *frequent itemsets*. Frequent itemsets have a downward closure property, that is, any subset of a frequent itemset must also be frequent. Even so, the problem of counting the number of maximal frequent itemsets, or itemsets that are not subsets of other frequent itemsets, is #P-complete, suggesting that the problem of enumerating all frequent itemsets can in general be hard [Yang, 2004]. Since the introduction of the Apriori method by Agrawal and Srikant [1994], researchers have proposed many algorithms for frequent pattern mining that apply various heuristic techniques to traverse the search space, which grows exponentially with the number of items in the database [Han et al., 2007, Hipp et al., 2000, Goethals, 2003].

Frequent itemset generation often leads to an overwhelming number of rules, making it difficult to distinguish the most useful rules. To make sense of such an enormous collection of rules, users typically rank them by a measure of "interestingness," which can be defined in many different ways. There is a large body of literature on interestingness measures, such as lift, conviction, Laplace, and gain [review articles include those of Tan and Kumar, 2000, McGarry, 2005, Geng and Hamilton, 2006]. The existence of so many interestingness measures introduces another problem of how to select an interestingness measure for a particular task. Bayardo and Agrawal [1999]

showed that if the head of the rule is fixed, then a number of metrics, including those listed above, are optimized by rules that lie along the upper support-confidence border, where a rule on this border has the highest confidence among rules with equal or higher support. They proposed an algorithm to mine only this border, which indeed produces a reduced set of rules. In this paper, we extend the idea of an optimal border to general rules, not just the case of rules with fixed heads, and we use MIO to find the border.

Association rules were originally designed for data exploration, and later *associative classification* developed as a framework to use the rules for classification, with algorithms such as CBA, CMAR, and CPAR [Liu et al., 1998, Li et al., 2001, Yin and Han, 2003, Simon et al., 2011]. Reviews of the different approaches are given by Thabtah [2007], Rückert [2008], and Vanhoof and Depaire [2010]. Methods to build a classifier using a sorted set of association rules fall into two categories: those that predict based on multiple rules, and those that predict based on a single rule in a ranked list of rules. The first category uses more information by classifying based on a sort of majority vote of rules, but typically has two major disadvantages: first, it ignores the dependency between rules, so even two rules that are almost exactly the same have two separate votes instead of one; and second, the model loses interpretability by combining rules together. Boosted decision trees share a related problem–they no longer have the interpretability of single decision trees. Examples of rule ensemble classifiers are in Friedman and Popescu [2008] and Meinshausen [2010]. These models are similar to the Logical Analysis of Data (LAD) model [Boros et al., 2000], though the LAD model uses only rules that have confidence equal to one, so that even rules with confidence 0.99 are discarded, which could lead to overfitting. The second category of sorted-rule-based classification algorithms produces decision lists, and are related to the "teleo-reactive programs" introduced by Nilsson [1994]. These classifiers are simple to understand and use the highest ranked rules for prediction. However, if the list is not properly ordered, it may not yield an accurate classifier. There is a small literature of theoretical work on decision lists [see Rivest, 1987, Klivans and Servedio, 2006, Sokolova et al., 2003, Anthony, 2005, Long and Servedio, 2007, Marchand and

Sokolova, 2005, Rudin et al., 2011]. Decision lists can be created by ordering rules according to an interestingness measure. Alternatively, the ordering of rules can be learned from data, which is the approach we take here. Learning the rule list has the potential to be substantially more accurate in terms of misclassification error than ranking rules by an arbitrary choice of interestingness measure. As far as we know, there are no other mathematical programming approaches to creating decision lists in the literature.

4.2 Mining Optimal Association Rules

In this section, we describe an MIO method to generate the rules that form the building blocks for the classifier. First, we derive constraints that characterize the full set of possible rules for a database. Then, we present an MIO algorithm to find a set of general rules. Finally, we address the special case of mining rules for binary classification, for which the rules have a particular form.

4.2.1 Interestingness and the Frontier

We use the following standard notation: let $\mathcal{I} = \{1, \ldots, d\}$ be a set of items, and $X \subseteq \mathcal{I}$ be an itemset. Let $\mathcal{D}$ be a database of itemsets. Each itemset or row in the database is called a *transaction*. An association rule has the form $X \Rightarrow Y$, where $X, Y \subseteq \mathcal{I}$ and $X \cap Y = \emptyset$.

Suppose there are n transactions in the database $\mathcal{D}$, and let $t_i \in \{0,1\}^d$ represent transaction i:

$$t_{ij} = \mathbf{1}_{[\text{transaction } i \text{ includes item } j]}, \qquad 1 \leq i \leq n, \quad 1 \leq j \leq d.$$

The t_i are data. Now we introduce the decision variables. Let $b, h \in \{0,1\}^d$ represent the body and head of a given rule $X \Rightarrow Y$. That is, for $j = 1, \ldots, d$, let

$$b_j = \mathbf{1}_{[j \in X]} \quad \text{and} \quad h_j = \mathbf{1}_{[j \in Y]}.$$

We also use decision variables x_i, y_i, and z_i, for $i = 1, \ldots, n$, where

$$x_i = \mathbf{1}_{[\text{transaction } i \text{ includes } X]},$$
$$y_i = \mathbf{1}_{[\text{transaction } i \text{ includes } Y]},$$
$$z_i = \mathbf{1}_{[\text{transaction } i \text{ includes } X \text{ and } Y]}.$$

The following constraints define the space $\mathcal{P}$ of possible association rules. Each constraint is explained below.

$$b_j + h_j \leq 1, \quad \forall j, \tag{4.1}$$
$$x_i \leq 1 + (t_{ij} - 1)b_j, \quad \forall i, j, \tag{4.2}$$
$$x_i \geq 1 + (t_i - e_d)^T b, \quad \forall i, \tag{4.3}$$
$$y_i \leq 1 + (t_{ij} - 1)h_j, \quad \forall i, j, \tag{4.4}$$
$$y_i \geq 1 + (t_i - e_d)^T h, \quad \forall i, \tag{4.5}$$
$$z_i \leq x_i, \quad \forall i, \tag{4.6}$$
$$z_i \leq y_i, \quad \forall i, \tag{4.7}$$
$$z_i \geq x_i + y_i - 1, \quad \forall i, \tag{4.8}$$
$$b_j, h_j \in \{0, 1\}, \quad \forall j, \tag{4.9}$$
$$0 \leq x_i, y_i, z_i \leq 1, \quad \forall i. \tag{4.10}$$

Note that e_d is the d-vector of ones. Since an item cannot be in both the body and head of a rule ($X \cap Y = \emptyset$), b and h must satisfy (4.1). To understand (4.2), consider the two cases $b_j = 0$ and $b_j = 1$. If $b_j = 0$, then the constraint is just $x_i \leq 1$, so the constraint has no effect. If $b_j = 1$, then the constraint is $x_i \leq t_{ij}$. That is, if $b_j = 1$ (item j is in X) but $t_{ij} = 0$ (item j is not in transaction i), then $x_i = 0$. This set of constraints implies that $x_i = 0$ if transaction i does not include X. We need (4.3) to enforce that $x_i = 1$ if transaction i includes X. Note that $t_i^T b$ is the number of items in the intersection of transaction i and X, and $e_d^T b$ is the number of items in X.

Table 4.1: The body X of the rule is in transaction i since (4.2) and (4.3) are satisfied.

	j				
	1	2	3	4	5
t_i (1 if item j in transaction i, 0 otherwise)	1	0	1	1	0
b (1 if item j in body of rule, 0 otherwise)	1	0	0	1	0

Constraint (4.3) is valid because

$$t_i^T b = \sum_{j=1}^{d} t_{ij} b_j \leq \sum_{j=1}^{d} b_j = e_d^T b,$$

where equality holds if and only if transaction i includes X and otherwise $t_i^T b \leq e_d^T b - 1$. Table 4.1 helps to clarify (4.2) and (4.3). Constraints (4.4) and (4.5) capture the y_i in the same way that (4.2) and (4.3) capture the x_i. The z_i are 1 if and only if $x_i = y_i = 1$, which is captured by (4.6) through (4.8). Constraints (4.9) and (4.10) specify that b and h are restricted to be binary, while the values of x, y, and z are restricted only to be between 0 and 1.

Each point in $\mathcal{P}$ corresponds to a rule $X \Rightarrow Y$, where $X = \{j : b_j = 1\}$ and $Y = \{j : h_j = 1\}$. There are $2d$ binary variables, $3n$ continuous variables, and $d + 2nd + 5n$ constraints. Computationally, it is favorable to reduce the number of integer variables, and here we explain why x, y, and z are not also restricted to be integral. There are two cases when deciding whether X is in transaction i. If it is, then (4.3) says $x_i \geq 1$, which implies $x_i = 1$. If it is not, then there exists j such that $t_{ij} = 0$ and $b_j = 1$, so (4.2) says $x_i \leq 0$ for some j, which implies $x_i = 0$. Thus, in either case, x_i is forced to be an integer, regardless of whether we specify it as an integer variable. The argument is similar for y_i. For z_i, there are two cases when deciding whether X and Y are both in transaction i. If they are, then $x_i = y_i = 1$, so (4.8) says $z_i \geq 1$, which implies $z_i = 1$. If they are not, then either (4.6) or (4.7) says $z_i \leq 0$, which implies $z_i = 0$. Thus, z_i is also always integral.

The number of feasible points in $\mathcal{P}$ grows exponentially in the number of items $d = |\mathcal{I}|$. It includes the full set of association rules, which is many more than we usually need or wish to collect. In order to capture only the potentially interesting

Table 4.2: Interestingness measures.

Measure	**Definition**	
Coverage	$P(X)$	s_X
Prevalence	$P(Y)$	s_Y
Support	$P(X \cup Y)$	s
Confidence/Precision	$P(Y\|X)$	$\frac{s}{s_X}$
Recall	$P(X\|Y)$	$\frac{s}{s_Y}$
Accuracy	$P(X \cup Y) + P(X^c \cup Y^c)$	$1 - s_X - s_Y + 2s$
Lift/Interest	$\frac{P(X \cup Y)}{P(X)P(Y)}$	$\frac{s}{s_X s_Y}$
Conviction	$\frac{P(X)P(Y^c)}{P(X \cup Y^c)}$	$\frac{1 - s_Y}{1 - s/s_X}$
Laplace Correction	$\frac{nP(X \cup Y) + 1}{nP(X) + k}$, k is number of classes	$\frac{ns + 1}{ns_X + k}$
Piatetsky-Shapiro	$P(X \cup Y) - P(X)P(Y)$	$s - s_X s_Y$

rules, we judge each rule according to three of its fundamental properties, namely

$$s_X = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad s_Y = \frac{1}{n}\sum_{i=1}^{n} y_i, \quad \text{and} \quad s = \frac{1}{n}\sum_{i=1}^{n} z_i,$$

called *coverage*, *prevalence*, and *support* respectively. When we refer to these measures for a particular rule r, we use the notation $s_X(r)$, $s_Y(r)$, and $s(r)$; we omit the parenthetical "(r)" when referring to them in general. Using s_X, s_Y, and s, we can capture many interestingness measures in addition to coverage, prevalence, and support, some of which are shown in Table 4.2. The notation $P(A)$ means the fraction, or empirical probability, of transactions containing itemset A.

We define a partial order $\leq_p$ over the set of possible of rules. Given two rules r and r^*, we have $r \leq_p r^*$ if and only if:

$$s_X(r) \geq s_X(r^*), \quad s_Y(r) \geq s_Y(r^*), \quad \text{and} \quad s(r) \leq s(r^*). \tag{4.11}$$

Moreover, $r =_p r^*$ if and only if $s_X(r) = s_X(r^*)$, $s_Y(r) = s_Y(r^*)$, and $s(r) = s(r^*)$. In words, "$r \leq_p r^*$" means that the coverage and prevalence of r^* are no greater than that of r, but the support of r^* is at least that of r. For intuition, consider the interestingness measure of confidence, which is the empirical probability of Y given X.

Table 4.3: Number of transactions containing certain items.

	Case 1	**Case 2**
{chips}	8	10
{cookies}	8	8
{chips, guacamole}	7	5
{cookies, milk}	5	5

Suppose we have two rules—**a**: {chips}⇒{guacamole} and **b**: {cookies}⇒{milk}—and refer to the data in Table 4.3. In Case 1, we have:

$$s_X(\mathbf{a}) = s_X(\mathbf{b}) = 8, \quad s(\mathbf{a}) = 7, \quad \text{and} \quad s(\mathbf{b}) = 5,$$

so the two rules have equal coverage, but the support is higher for **a**. The confidence of **a** and **b** are $\frac{7}{8}$ and $\frac{5}{8}$ respectively, thus **a** dominates **b** in confidence. In Case 2, we have:

$$s_X(\mathbf{a}) = 10, \quad s_X(\mathbf{b}) = 8, \quad \text{and} \quad s(\mathbf{a}) = s(\mathbf{b}) = 5,$$

so the rules have equal support, but the coverage is lower for **b**. The confidence of **a** and **b** are $\frac{5}{10}$ and $\frac{5}{8}$ respectively, thus **b** dominates **a**. This example shows that higher support and lower coverage increase the confidence of a rule; for other measures, lower prevalence also often increases the interestingness.

Let F^* be the set of rules that are not dominated by any other rules, that is,

$$\mathcal{F}^* = \{r : \text{ There does not exist any } \bar{r} \text{ such that } r <_p \bar{r}.\}.$$

The rules $r \in \mathcal{F}^*$ fall along a three dimensional frontier in s_X, s_Y, and s. Many interestingness measures, including those in Table 4.2, increase with decreasing s_X (holding s_Y and s constant), decreasing s_Y (holding s_X and s constant), and increasing s (holding s_X and s_Y constant). Thus, the rules that optimize each of these measures are in $\mathcal{F}^*$. Since we do not wish to generate all possible rules, we choose to focus on mining this particular frontier because it contains the most "interesting" rules according to a variety of measures.

4.2.2 MIO Algorithm for General Association Rule Mining

We can find each rule on the frontier $\mathcal{F}^*$ corresponding to $\leq_p$ by putting upper bounds on both s_X and s_Y, and then maximizing s. We vary the bounds over all possible values to produce the entire frontier. In particular, Formulation (4.12) maximizes the "scaled support" $(n \cdot s)$ for a certain choice $\bar{s}_X$ and $\bar{s}_Y$, which denote the user-specified upper bounds on the "scaled coverage" $(n \cdot s_X)$ and "scaled prevalence" $(n \cdot s_Y)$ respectively.

$$\begin{aligned} \max_{b,h,x,y,z} \quad & \sum_{i=1}^{n} z_i - R_{\text{gen\_xy}} \left(\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i \right) - R_{\text{gen\_bh}} \left(\sum_{j=1}^{d} b_j + \sum_{j=1}^{d} h_j \right) \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i \leq \bar{s}_X, \\ & \sum_{i=1}^{n} y_i \leq \bar{s}_Y, \\ & (b, h, x, y, z) \in \mathcal{P}. \end{aligned} \tag{4.12}$$

The first term in the objective is the scaled support. The second set of terms $\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i$ correspond to the coverage s_X and prevalence s_Y; if there are multiple rules with optimal support, we want those with smaller coverage and prevalence since otherwise we would be generating rules not on the frontier. The third set of terms $\sum_{j=1}^{d} b_j + \sum_{j=1}^{d} h_j$ are for regularization, and correspond to the sparsity of the rule; if there are multiple rules that maximize s and have equal s_X and s_Y, we want those with smaller bodies and heads, that is, more zeros in b and h. The parameters $R_{\text{gen\_xy}}$ and $R_{\text{gen\_bh}}$ control the weight of these terms in the objective, where the former ensures that we properly trace out the frontier, and the latter could potentially trade off sparsity for closeness to the frontier.

Solving (4.12) once for each possible pair $(\bar{s}_X, \bar{s}_Y)$ does not yield the entire frontier since there may be multiple optimal rules at each point on the frontier. To find other optima, we add constraints making each solution found so far infeasible, so that they cannot be found again when we re-solve. Specifically, for each pair $(\bar{s}_X, \bar{s}_Y)$, we

iteratively solve the formulation as follows: Let (h^*, b^*) be the first optimum we find for (4.12). In each iteration, we add the constraint

$$\sum_{j:b_j^*=0} b_j + \sum_{j:b_j^*=1} (1 - b_j) + \sum_{j:h_j^*=0} h_j + \sum_{j:h_j^*=1} (1 - h_j) \geq 1 \tag{4.13}$$

to the formulation. This constraint says that in either the vector b or the vector h, at least one of the components must be different from in the previous solution; that is, at least one of the zeros must be one or one of the ones must be zero. The previous solution $b_j = b_j^*$ and $h_j = h_j^*$ is infeasible since it would yield $0 \geq 1$ in (4.13). After adding this constraint, we solve again. If the optimal value of $\bar{s} = \sum_{i=1}^{n} z_i$ decreases, then we exit the loop. Otherwise, we have a new optimum, so we repeat the step above to generate another constraint and re-solve.

4.2.3 MIO Algorithm for Associative Classification

As our main goal is to use association rules to construct a decision list for binary classification, we show in this section how to use MIO to mine rules for this purpose. In this case, the rules are of a specific form, either $X \Rightarrow 1$ or $X \Rightarrow -1$. That is, we prespecify the heads Y of the rules to be a class attribute, 1 or -1. Our rule generation algorithm mines two separate frontiers of rules, one frontier for each class.

Suppose we want to generate rules on the frontier for class $y \in \{-1, 1\}$. Let $S = \{i : \text{ transaction } i \text{ has class label } y\}$. Then $s = \frac{1}{n} \sum_{i \in S} x_i$. Since $s_Y = |S|$ is equal for all rules of interest, we simplify the partial order (4.11) so that given two rules r and r^*, we have $r \leq_p r^*$ if and only if:

$$s_X(r) \geq s_X(r^*) \quad \text{and} \quad s(r) \leq s(r^*).$$

Also, $r =_p r^*$ if and only if $s_X(r) = s_X(r^*)$ and $s(r) = s(r^*)$. The corresponding two dimensional frontier in s_X and s can be found by upper bounding s_X and maximizing s. Since Y is fixed, we do not need the h, y, or z variables from (4.12).

Formulation (4.14) finds a rule with maximum s for a given upper bound $\bar{s}_X$ on $n \cdot s_X$.

$$\begin{aligned}
\max_{b,x} \quad & \sum_{i \in S} x_i - R_{\text{gen_x}} \sum_{i=1}^{n} x_i - R_{\text{gen_b}} \sum_{j=1}^{d} b_j \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_i \leq \bar{s}_X, \\
& x_i \leq 1 + (t_{ij} - 1) b_j, \quad \forall i, j, \\
& x_i \geq 1 + (t_i - e_d)^T b, \quad \forall i, \\
& b_j \in \{0, 1\}, \quad \forall j, \\
& 0 \leq x_i \leq 1, \quad \forall i.
\end{aligned} \tag{4.14}$$

The first term in the objective corresponds to support, and the others correspond to coverage and sparsity, similar to the terms in (4.12). Solving (4.14) once for each value of $\bar{s}_X$ does not yield the entire frontier since there may be multiple optima. Analogous to the general case, we solve the formulation iteratively: Start by setting $\bar{s}_X = n$ since the largest possible value of the scaled coverage is n. Let b^* be the first optimum. Add the "infeasibility constraint"

$$\sum_{j : b_j^* = 0} b_j + \sum_{j : b_j^* = 1} (1 - b_j) \geq 1 \tag{4.15}$$

to the formulation, and solve again. If we find another optimum, then we repeat the step above to generate another constraint and re-solve. If the optimal value of $\bar{s} = \sum_{i \in S} x_i$ decreases, then we set the upper bound on $\bar{s}_X$ to a smaller value and iterate again. Note that we can set this new value to be the minimum of $\sum_{i=1}^{n} x_i$ and $\bar{s}_X - 1$ (previous bound minus one); we know that no rule on the remainder of the frontier has scaled coverage greater than $\sum_{i=1}^{n} x_i$, so using this as the bound provides a tighter constraint than using $\bar{s}_X - 1$ whenever $\sum_{i=1}^{n} x_i < \bar{s}_X - 1$.

Thus our rule generation algorithm, called "RuleGen," generates the frontier, one rule at a time, from largest to smallest coverage. The details are shown in Figure 4-1. RuleGen allows optional minimum coverage thresholds $\texttt{mincov}_{-1}$ and $\texttt{mincov}_1$ to be

Set $\texttt{mincov}_{-1}$, $\texttt{mincov}_1$, iter_lim.
For Y in {-1,1}
    Initialize sX $\leftarrow$ n, iter $\leftarrow$ 1, s $\leftarrow$ 0.
    Initialize collection of rule bodies $\mathcal{R}_{\texttt{Y}} = \emptyset$.
    Repeat
        If iter = 1 then
            Solve (4.14) to obtain rule X $\Rightarrow$ Y.
            $\texttt{s} \leftarrow \sum_{\texttt{i} \in \texttt{S}} \texttt{x[i]}$
            iter $\leftarrow$ iter + 1
        $\mathcal{R}_{\texttt{Y}} \leftarrow \mathcal{R}_{\texttt{Y}} \cup \texttt{X}$
        Add new constraint (4.15).
        If iter $\leq$ iter_lim then
            Solve (4.14) to obtain rule X $\Rightarrow$ Y.
            If $\sum_{\texttt{i} \in \texttt{S}} \texttt{x[i]} < \texttt{s}$ then
                $\texttt{sX} \leftarrow \min\left(\sum_{\texttt{i}=1}^{\texttt{n}} \texttt{x[i]}, \texttt{sX} - 1\right)$
                iter $\leftarrow$ 1
            Else iter $\leftarrow$ iter + 1
        Else
            sX $\leftarrow$ sX $-$ 1
            iter $\leftarrow$ 1
    While sX $\geq$ n $\cdot$ $\texttt{mincov}_{\texttt{Y}}$

Figure 4-1: RuleGen algorithm. (Note sX=$\bar{s}_X$ and s=$\bar{s}$.)

imposed on each of the classes of rules. Also, iter_lim limits the number of times we iterate the procedure above for a fixed value of s_X with adding (4.15) between iterates. To find all rules on the frontiers, set $\texttt{mincov}_{-1} = \texttt{mincov}_1 = 0$ and $\texttt{iter\_lim} = \infty$. Figure 4-2 shows a fictitious example to illustrate the steps of the algorithm:

a. Suppose we are constructing the frontier for data with $n = 100$. Initialize sX to n and solve (4.14). Assume the first solution has $\sum_{i \in S} x_i = 67$. Then the algorithm adds the first rule to $\mathcal{R}_{\texttt{Y}}$ and sets s to 67. It adds the infeasibility constraint (4.15) to (4.14) and re-solves. Assume the new rule still has

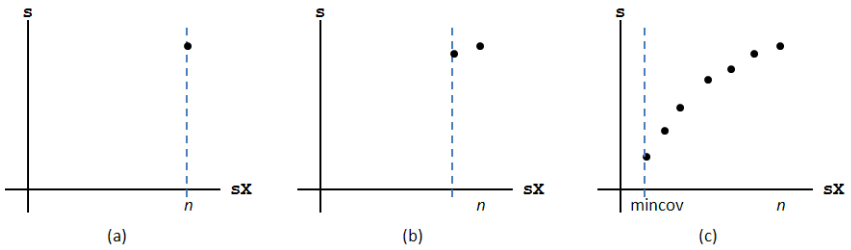


Figure 4-2: Illustrative example to demonstrate the steps in the RuleGen algorithm.

$\sum_{i \in S} x_i = 67$, so the algorithm adds this rule to $\mathcal{R}_{\mathtt{Y}}$, adds another infeasibility constraint to (4.14) and re-solves.

b. Assume the new rule has $\sum_{i \in S} x_i = 65$ and $\sum_{i=1}^{n} x_i = 83$ (corresponding to the support and coverage respectively). Since $\sum_{i \in S} x_i$ decreased, the algorithm sets $\mathtt{sX}$ to $\min\left(\sum_{i=1}^{n} x_i, \mathtt{sX} - 1\right) = \min(83, 99) = 83$ before re-solving to obtain the next rule on the frontier and adding it to $\mathcal{R}_{\mathtt{Y}}$.

c. This process continues until the minimum coverage threshold is reached.

4.3 Building a Classifier

Suppose we have generated L rules, where each rule ℓ is of the form $X_\ell \Rightarrow -1$ or $X_\ell \Rightarrow 1$. Our task is now to rank them to build a decision list for classification. Given a new transaction, the decision list classifies it according to the highest ranked rule ℓ such that X_ℓ is in the transaction, or the highest rule that "applies" to the transaction. In this section, we derive an empirical risk minimization algorithm using MIO that yields an optimal ranking of rules. That is, the ranking returned by our algorithm optimizes the classification accuracy on a training sample.

We always include in the set of rules to be ranked two "null rules:" $\emptyset \Rightarrow -1$, which predicts class -1 for any transaction, and $\emptyset \Rightarrow 1$, which predicts class 1 for any transaction. In the final ranking, the higher of the null rules corresponds effectively to the bottom of the ranked list of rules; all examples that reach this rule are classified by it, thus the class it predicts is the default class. We include both null rules in the set of rules because we do not know which of them would serve as the better default,

Table 4.4: Transaction i is classified as -1 (highest rule that applies predicts -1).

Transaction i	c_i		
{1 0 1 1 0}	-1		
Ranked rules	$p_{i\ell}$	r_ℓ	$u_{i\ell}$
{0 1 0 0 1} ⇒ -1	0	10	0
{0 1 1 0 0} ⇒ 1	0	9	0
{1 0 1 0 0} ⇒ -1	**-1**	**8**	**1**
{0 0 0 0 0} ⇒ 1	1	7	0
⋮	⋮	⋮	⋮
{0 0 1 1 0} ⇒ -1	-1	1	0

that is, which would help the decision list to achieve the highest possible classification accuracy; our algorithm learns which null rule to rank higher.

We use the following parameters:

$$a_i = \text{true class attribute of transaction } i,$$

$$p_{i\ell} = \begin{cases} 1 & \text{if rule } \ell \text{ predicts class 1 for transaction } i, \\ -1 & \text{if rule } \ell \text{ predicts class } -1 \text{ for transaction } i, \\ 0 & \text{if rule } \ell \text{ does not apply to transaction } i, \end{cases}$$

$$v_{i\ell} = \mathbf{1}_{[X_\ell \text{ is in transaction } i]} = |p_{i\ell}|,$$

and decision variables:

$$c_i = \text{predicted class of transaction } i, \qquad r_\ell = \text{rank of rule } \ell,$$

$$u_{i\ell} = \mathbf{1}_{[\text{rule } \ell \text{ is the rule that predicts the class of transaction } i]}.$$

$$\text{Then} \quad c_i = \sum_{\ell=1}^{L} p_{i\ell} u_{i\ell}, \quad \text{where} \quad u_{i\ell} \in \{0,1\} \quad \forall i, \ell \quad \text{and} \quad \sum_{\ell=1}^{L} u_{i\ell} = 1. \tag{4.16}$$

In words, for a particular transaction i, $u_{i\ell} = 0$ for all except one rule, which is the one among those that apply with the highest rank r_ℓ. Table 4.4 shows an example of these parameters ($p_{i\ell}$) and variables (r_ℓ, $u_{i\ell}$, c_i) for a particular transaction to be classified by a given decision list. The formulation to build the optimal classifier is:

$$\max_{r,r_*,g,u,s,\alpha,\beta} \quad \sum_{i=1}^{n}\sum_{\ell=1}^{L} \tilde{p}_{i\ell} u_{i\ell} + R_{\text{rank}} r_* \tag{4.17}$$

$$\text{s.t.} \quad \sum_{\ell=1}^{L} u_{i\ell} = 1, \quad \forall i, \tag{4.18}$$

$$g_i \geq v_{i\ell} r_\ell, \quad \forall i, \ell, \tag{4.19}$$

$$g_i \leq v_{i\ell} r_\ell + L(1 - u_{i\ell}), \quad \forall i, \ell, \tag{4.20}$$

$$u_{i\ell} \geq 1 - g_i + v_{i\ell} r_\ell, \quad \forall i, \ell, \tag{4.21}$$

$$u_{i\ell} \leq v_{i\ell}, \quad \forall i, \ell, \tag{4.22}$$

$$r_\ell = \sum_{k=1}^{L} k s_{\ell k}, \quad \forall \ell, \tag{4.23}$$

$$\sum_{k=1}^{L} s_{\ell k} = 1, \quad \forall \ell, \tag{4.24}$$

$$\sum_{\ell=1}^{L} s_{\ell k} = 1, \quad \forall k, \tag{4.25}$$

$$r_* \geq r_A, \tag{4.26}$$

$$r_* \geq r_B, \tag{4.27}$$

$$r_* - r_A \leq (L-1)\alpha, \tag{4.28}$$

$$r_A - r_* \leq (L-1)\alpha, \tag{4.29}$$

$$r_* - r_B \leq (L-1)\beta, \tag{4.30}$$

$$r_B - r_* \leq (L-1)\beta, \tag{4.31}$$

$$\alpha + \beta = 1, \tag{4.32}$$

$$u_{i\ell} \leq 1 - \frac{r_* - r_\ell}{L-1}, \quad \forall i, \ell, \tag{4.33}$$

$$\alpha, u_{i\ell}, s_{\ell k} \in \{0, 1\}, \quad \forall i, \ell, k,$$

$$0 \leq \beta \leq 1,$$

$$r_\ell \in \{1, 2, \ldots, L\}, \quad \forall \ell.$$

As in previous chapters, we use (4.17) to refer to the entire MIO formulation and not just the objective. The first term in the objective corresponds to accuracy: Since

$a_i c_i$ is 1 if transaction i is correctly classified and -1 otherwise, the number of correct classifications is

$$\sum_{i=1}^{n} \left(\frac{a_i c_i + 1}{2} \right) = \frac{1}{2} \left(n + \sum_{i=1}^{n} a_i c_i \right).$$

Thus, using (4.16) and letting $\tilde{p}_{i\ell} = a_i p_{i\ell}$, we want to maximize

$$\sum_{i=1}^{n} a_i c_i = \sum_{i=1}^{n} \sum_{\ell=1}^{L} \tilde{p}_{i\ell} u_{i\ell}.$$

Constraint (4.18) enforces that for each i, only one of the $u_{i\ell}$ variables equals one while the rest are zero. To capture the definition of $u_{i\ell}$, we use an auxiliary variable g_i, which represents the rank of highest the applicable rule for transaction i. Through (4.19) and (4.20), there is only one ℓ such that $u_{i\ell} = 1$ is feasible, namely the ℓ corresponding to the highest value of $v_{i\ell} r_\ell$. Constraints (4.21) and (4.22) are not necessary but help improve the linear relaxation and thus are intended to speed up computation. We assign the integral ranks r_ℓ using (4.23) through (4.25), which imply $s_{\ell k} = 1$ if rule ℓ is assigned to rank k. The matching between ranks and rules is one-to-one.

We add regularization in order to favor a shorter overall list of rules. That is, our regularizer pulls the rank of the higher null rule as high as possible. If r_A is the rank of $\emptyset \Rightarrow -1$ and r_B is the rank of $\emptyset \Rightarrow 1$, then we add r_* to the objective function, where r_* is the maximum of r_A and r_B. The regularization coefficient of r_* in the objective is R_{rank}. We capture r_* using (4.26) through (4.32): Either $\alpha = 1$ and $\beta = 0$ or $\beta = 1$ and $\alpha = 0$. If $\alpha = 1$, then $r_* = r_A$. If $\beta = 1$, then $r_* = r_B$. Since we are maximizing, r_* equals the higher of r_A and r_B. Note that if α is binary, then β need not be binary because the constraint $\alpha + \beta = 1$ forces integral values for β. If the rank r_ℓ of rule ℓ is below r_*, then $u_{i\ell} = 0$ for all i, so (4.33) is also valid, and we include it to help speed up computation.

The Ordered Rules for Classification (ORC) algorithm consists of generating rules using RuleGen, computing the $p_{i\ell}$ and $v_{i\ell}$, and then solving (4.17). The rule generation step could also be replaced by a different method, such as Apriori [Agrawal and Srikant, 1994]. We use RuleGen in the experiments.

4.4 Computational Results

We used various publicly available datasets from the UCI Machine Learning Repository [Asuncion and Newman, 2007], so our results can be easily compared with those of other works. For each dataset, we divided the data evenly into three folds and used each fold in turn as a test set, training each time with the other two folds. The training and test accuracy were averaged over these three folds. We compared the ORC algorithm with four other classification methods—logistic regression [see Hastie et al., 2001, Dreiseitl and Ohno-Machado, 2002], Support Vector Machines (SVM) [Vapnik, 1995, Burges, 1998], Classification and Regression Trees (CART) [Breiman et al., 1984], and AdaBoost [Freund and Schapire, 1995]—all run using R 2.8.1. We used the radial basis kernel and regularization parameter $C = 1$ for SVM, and decision trees as base classifiers for AdaBoost. The ORC algorithm was implemented using ILOG AMPL 11.210 with the Gurobi solver.[1]

Here we explain how we chose the parameter settings for the ORC experiments. In generating rules with Formulation (4.14), we wanted to ensure that $R_{\text{gen\_x}}$ was small enough that the solver would never choose to decrease the scaled support $\sum_{i \in S} x_i$ just to decrease the scaled coverage $\sum_{i=1}^{n} x_i$. That is, $R_{\text{gen\_x}}$ should be such that we would not sacrifice maximizing s for lower s_X; this required only that this parameter be a small positive constant, so we chose $R_{\text{gen\_x}} = \frac{0.1}{n}$. Similarly, we did not want to sacrifice maximizing s or lowering s_X for greater sparsity, so we chose $R_{\text{gen\_b}} = \frac{0.1}{nd}$. In order to not sacrifice classification accuracy for a shorter decision list in ranking the rules with Formulation (4.17), we chose $R_{\text{rank}} = \frac{1}{L}$. We also used a minimum coverage threshold of 0.05, and iterated five times ($\texttt{mincov}_{-1} = \texttt{mincov}_1 = 0.05$, $\texttt{iter\_lim} = 5$); these choices were based on preliminary experiments on the datasets to determine parameters that would yield in a reasonable number of rules.

Table 4.5 shows the dataset sizes as well as average number of rules generated by RuleGen and runtimes in seconds for our algorithms; the runtimes for other methods

[1]For SPECT, Haberman, Votes, and CarEval, we used Gurobi 4.5.2 on a computer with an Intel quad core Xeon E5687 3.60GHz processor and 48GB of RAM. For the other datasets, we used Gurobi 3.0.0 on a computer with two Intel quad core Xeon E5440 2.83GHz processors and 32GB of RAM.

were negligible. We generally terminated the solver before (4.17) solved to provable optimality; see the appendix at the end of this chapter for details on the experiments. Table 4.6 shows the average training and test classification accuracy (± one standard deviation) for each dataset. Bold indicates the highest average in the row. These results show that in terms of accuracy, the ORC algorithm is on par with top methods such as SVM or boosted decision trees. In fact, in an extensive empirical comparison of machine learning algorithms, Caruana and Niculescu-Mizil [2006] found that boosted decision trees performed best overall.

Table 4.5: Dataset sizes and for each dataset: average number of rules generated by RuleGen (for both classes), Time_1 = average time to generate all rules using RuleGen (seconds), and Time_2 = average time to rank rules using ORC algorithm (seconds).

Dataset	n	$\lvert S\rvert$	d	**#Rules**	**Time$_1$**	**Time$_2$**
SPECT	267	212	22	145	72	8862
Haberman	306	225	10	15	15	6
MONK1	432	216	17	55	101	247
MONK2	432	142	17	45	124	5314
MONK3	432	228	17	58	100	731
Votes	435	267	16	266	108	21506
B.Cancer	683	239	27	198	616	12959
Mammo	830	403	25	58	671	3753
TicTac	958	626	27	53	1241	4031
CarEval	1728	518	21	58	706	7335

*(4.17) solved to provable optimality for MONK1 and MONK3

4.5 Interpretability

Interpretability is a subjective matter, but we aim to demonstrate that the ORC classifier performs well in terms of being easy to understand. We give examples using a few of the datasets from Section 4.4. For each dataset, we take the rules from training on Folds 1 and 2 of the data. Classifiers generated by CART are interpretable because of their decision tree structure. The other methods for classification are not as easily interpreted. For example, the logistic regression model is

$$p = \frac{1}{1 + e^{-\beta_0 + \beta^T t}},$$

Table 4.6: Classification accuracy (averaged over three folds).

Dataset		LogReg	SVM	CART	AdaBoost	ORC
SPECT	train	0.8783 ± 0.0399	0.8633 ± 0.0366	0.8390 ± 0.0227	0.8764 ± 0.0149	$\mathbf{0.8970 \pm 0.0471}$
	test	0.7978 ± 0.0297	$\mathbf{0.8464 \pm 0.0619}$	0.7828 ± 0.0425	0.8202 ± 0.0297	0.7753 ± 0.0389
Haberman	train	0.7712 ± 0.0247	$\mathbf{0.7876 \pm 0.0221}$	0.7680 ± 0.0221	0.7761 ± 0.0123	0.7680 ± 0.0242
	test	$\mathbf{0.7582 \pm 0.0442}$	0.7386 ± 0.0204	0.7418 ± 0.0453	0.7418 ± 0.0226	$\mathbf{0.7582 \pm 0.0442}$
MONK1	train	0.7523 ± 0.0053	0.9907 ± 0.0080	0.9282 ± 0.0040	**1**	**1**
	test	0.7454 ± 0.0106	0.9537 ± 0.0424	0.8843 ± 0.0212	**1**	**1**
MONK2	train	0.6470 ± 0.0256	0.6736 ± 0.0035	0.7500 ± 0.0284	0.7523 ± 0.0106	$\mathbf{0.8299 \pm 0.0217}$
	test	0.6019 ± 0.0526	0.6713 ± 0.0145	0.6690 ± 0.0729	0.6505 ± 0.0395	$\mathbf{0.7338 \pm 0.0356}$
MONK3	train	**1**	0.9861 ± 0.0104	**1**	**1**	**1**
	test	**1**	0.9722 ± 0.0069	**1**	**1**	**1**
Votes	train	$\mathbf{0.9816 \pm 0.0190}$	0.9747 ± 0.0020	0.9598 ± 0.0105	0.9724 ± 0.0103	0.9747 ± 0.0072
	test	$\mathbf{0.9586 \pm 0.0276}$	0.9563 ± 0.0080	0.9540 ± 0.0159	0.9563 ± 0.0040	0.9563 ± 0.0080
B.Cancer	train	0.9788 ± 0.0121	$\mathbf{0.9846 \pm 0.0022}$	0.9561 ± 0.0110	0.9692 ± 0.0134	0.9766 ± 0.0108
	test	0.9502 ± 0.0417	$\mathbf{0.9619 \pm 0.0142}$	0.9488 ± 0.0091	$\mathbf{0.9619 \pm 0.0268}$	0.9532 ± 0.0091
Mammo	train	0.8482 ± 0.0136	$\mathbf{0.8687 \pm 0.0088}$	0.8422 ± 0.0076	0.8560 ± 0.0089	0.8536 ± 0.0165
	test	0.8374 ± 0.0249	0.8217 ± 0.0245	0.8301 ± 0.0217	$\mathbf{0.8422 \pm 0.0265}$	0.8337 ± 0.0202
TicTac	train	0.9833 ± 0.0080	0.9494 ± 0.0133	0.9348 ± 0.0047	0.9937 ± 0.0027	**1**
	test	0.9823 ± 0.0148	0.9165 ± 0.0262	0.8873 ± 0.0061	0.9750 ± 0.0062	**1**
CarEval	train	0.9580 ± 0.0027	0.9821 ± 0.0018	0.9659 ± 0.0035	$\mathbf{0.9962 \pm 5e\text{-}04}$	0.9598 ± 0.0093
	test	0.9485 ± 0.0027	0.9728 ± 0.0066	0.9618 ± 0.0046	$\mathbf{0.9907 \pm 0.0044}$	0.9508 ± 0.0036

where p is the probability that the class of transaction t is 1. The SVM model is a hyperplane that maximizes the margin between the hyperplane and the closest point to it from both classes; by using kernels, we can raise the dimension of the model and achieve high accuracy, but not interpretability. Though there is work devoted to interpreting SVMs, the result is usually a smaller set of nonlinear features, still within a linear combination [Sonnenburg et al., 2005]. AdaBoost combines weak classifiers—decision trees in our experiments—by minimizing an exponential loss function; thus, even though the base classifiers may be interpretable, the final model is not.

4.5.1 Haberman's Survival

In this dataset, each "transaction" represents a patient who underwent surgery for breast cancer. We split the two original features representing age, and number of positive axillary lymph nodes detected, each into five bins, so that our final dataset has ten binary features. RuleGen generated six rules predicting that the patient died within five years (<5) and ten rules predicting that the patient survived at least five years (5+). The classifier constructed by the ORC algorithm has four rules ranked above the highest null rule, that is, it has five rules. This is shown in Table 4.7. This simple classifier essentially implies the following:

Table 4.7: ORC classifier for Haberman's Survival data, predict 5+ (survived 5+ yrs) or <5 (died in <5 yrs).

Rule	**Patient Age**					**Number of Nodes**					Y	$\bar{s}$	$\bar{s}_X$
	30s	40s	50s	60s	>69	0	1-9	10-19	20-29	>29			
1						X					5+	80	93
2							X				5+	53	83
3		X									<5	18	55
4			X								<5	20	71
5											5+	147	204

1. If the patient has at most nine positive axillary nodes, then predict "5+."
2. If she has more than nine nodes and is in her 40s or 50s, then predict "<5."
3. Otherwise, predict "5+."

For comparison, CART generates the classification tree shown in Figure 4-3. Denoting by $P(x, 5+)$ the probability of patient x being in class "5+," the classifier implies:

1. If patient x has zero nodes and is not in her 40s, then $P(x, 5+) = 0.91$.
2. If patient x has zero nodes and is in her 40s, then $P(x, 5+) = 0.71$.
3. If patient x has more than zero nodes and is in her 30s, then $P(x, 5+) = 0.91$.
4. If patient x is not in her 30s, and has one to nine or more than nineteen nodes, then $P(x, 5+) = 0.61$.
5. If patient x is not in her 30s and has ten to nineteen nodes, then $P(x, 5+) = 0.33$.

We typically use a threshold probability of 0.5 for classification, in which case the tree can be summarized as: if the patient is not in her 30s and has ten to nineteen nodes, then predict "< 5;" otherwise, predict "5+." This is similar to the ORC classifier.

4.5.2 MONK's Problem 1

There are 17 binary features in this dataset, derived from the following six integer-valued features.

a1 = 1, 2, 3	**a2** = 1, 2, 3	**a3** = 1, 2	**a4** = 1, 2, 3	**a5** = 1, 2, 3, 4	**a6** = 1, 2

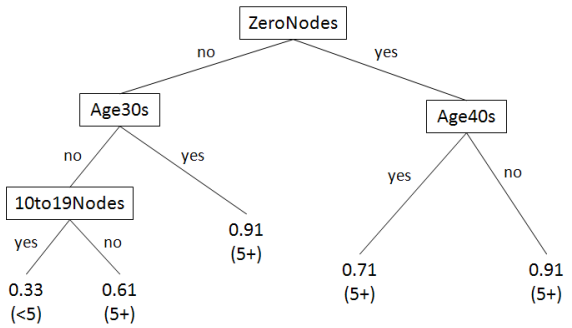


Figure 4-3: CART classifier for Haberman's Survival data (predicted class in parentheses).

This dataset is constructed so that each transaction is in class 1 if either a1=a2 or a5=1, and is in class -1 otherwise. RuleGen generated a total of 19 rules that predict class -1 and 33 rules that predict class 1. The ORC classifier is shown in Table 4.8. It

Table 4.8: ORC classifier for MONK's Problem 1 data, predict class 1 or -1.

Rule			s	s_X
{a1=3, a2=3}	$\rightarrow$	1	33	33
{a1=2, a2=2}	$\rightarrow$	1	30	30
{a5=1}	$\rightarrow$	1	65	65
{a1=1, a2=1}	$\rightarrow$	1	31	31
$\emptyset$	$\rightarrow$	-1	152	288

implies that if a row satisfies one of the four possible conditions to be in class 1, then predict class 1; otherwise predict -1. Thus, it achieves perfect accuracy. The CART classifier is shown in Figure 4-4 and implies:

If	**Predict**	
a5 = 1	1	(a5 = 1)
a5 $\neq$ 1, a4 = 3	-1	
a5 $\neq$ 1, a4 $\neq$ 3, a1 = 2, a2 $\neq$ 2	-1	
a5 $\neq$ 1, a4 $\neq$ 3, a1 = 2, a2 = 2	1	(a5 $\neq$ 1, a4 $\neq$ 3, a1 = a2 = 2)
a5 $\neq$ 1, a4 $\neq$ 3, a1 $\neq$ 2, a2 = 2	-1	
a5 $\neq$ 1, a4 $\neq$ 3, a1 $\neq$ 2, a2 = 1, a1 $\neq$ 1	-1	
a5 $\neq$ 1, a4 $\neq$ 3, a2 = 1, a1 = 1	1	(a5 $\neq$ 1, a4 $\neq$ 3, a1 = a2 = 1)
a5 $\neq$ 1, a4 $\neq$ 3, a2 $\neq$ 2, a2 $\neq$ 1, a1 = 1	-1	
a5 $\neq$ 1, a4 $\neq$ 3, a1 $\neq$ 2, a2 $\neq$ 2, a2 $\neq$ 1, a1 $\neq$ 1	1	(a5 $\neq$ 1, a4 $\neq$ 3, a1 = a2 = 3)

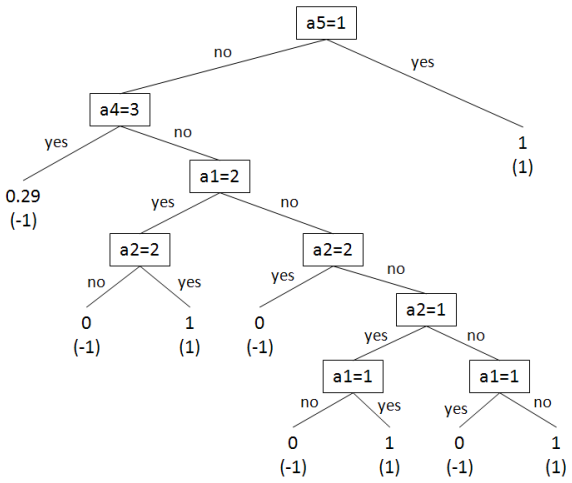


Figure 4-4: CART classifier for MONK's Problem 1 data (predicted class in parentheses).

In fact, the rules for predicting class 1, highlighted above in parentheses, are almost correct. However, the CART classifier specifies extra conditions on feature a4, which lowers the accuracy.

4.5.3 Congressional Votes

Each transaction in this dataset represents a member of the U.S. House of Representatives in 1984. There were sixteen key votes, shown in Table 4.9.[2] In addition, the table shows how many Republicans voted yes or no (Ry and Rn, respectively) and how many Democrats voted yes or no (Dy and Dn, respectively). Note that a representative could also vote neither yes nor no. Each of the sixteen binary features of the dataset corresponds to one of the sixteen key votes, and is one if the representative voted yes. In total, RuleGen generated 110 rules for Republicans and 116 rules for Democrats.

The ORC classifier is shown in Table 4.10; an "X" indicates a vote for yes. There is almost no overlap between the votes contained in the rules for Democrats and the rules for Republicans. The only overlap is with key vote 6, which was to allow student religious groups to meet in public secondary schools during non-class hours

[2]For details, see the CQ Almanac at http://library.cqpress.com/cqalmanac/.

Table 4.9: Key votes for Congressional Votes data.

V	Key Vote	Ry	Rn	Dy	Dn	V	Key Vote	Ry	Rn	Dy	Dn
1	handicapped-infants	31	131	151	100	9	mx-missile	17	141	182	56
2	water-project-cost	75	73	119	119	10	immigration	91	73	125	138
3	budget-resolution	21	139	229	29	11	synfuels-cutback	21	135	127	126
4	physician-fee-freeze	157	2	15	240	12	education-spending	133	20	36	213
5	el-salvador-aid	156	8	56	200	13	superfund-right-to-sue	135	22	73	178
6	religious-in-schools	147	17	123	134	14	crime	154	3	89	163
7	anti-satellite-test-ban	39	122	199	59	15	duty-free-exports	14	142	160	91
8	aid-to-nicaraguan	24	132	217	45	16	export-act-south-africa	96	50	173	12

Table 4.10: ORC classifier for Congressional Votes data, predict D (Democrat) or R (Republican).

Rule	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Y	s	s_X
1						X			X		X						D	20	20
2		X	X			X					X						D	28	28
3				X	X									X			R	106	115
4				X	X	X											R	103	112
5				X								X					R	95	101
6				X										X		X	R	67	69
7																	D	168	290

if other groups did so; the Democrats were split on this vote (that is, the number voting yes was approximately the same as the number voting no), while the majority of Republicans voted yes. The other key votes that characterized the Democrats were all one of the following two kinds:

- split for Democrats, majority no or split for Republicans (this was true for key votes 2 and 11),
- majority yes for Democrats, majority no for Republicans (this was true for key votes 3 and 9).

Most of the key votes other than V6 that characterized the Republicans followed one pattern: majority yes for Republicans and majority no for Democrats; this was true for key votes 4, 5, 12, and 14. The exception was key vote 16, which was majority yes for both Republicans and Democrats. The key vote (4) that appears in all rules for Republicans was to include provisions imposing a one-year physician fee freeze for Medicare services and to remove provisions that increased spending.

The CART classifier is shown in Figure 4-5. Using a threshold probability of 0.5, it implies: if a representative either did not vote yes for key vote 4, or voted yes for all three of key votes 4, 11, and 3, then the class is Democrat; otherwise it is Republican.

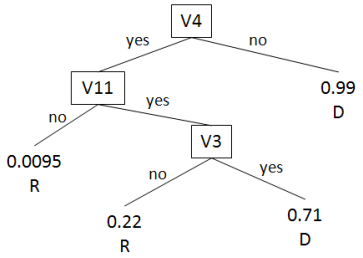


Figure 4-5: CART classifier for Congressional Votes data (predicted class in parentheses).

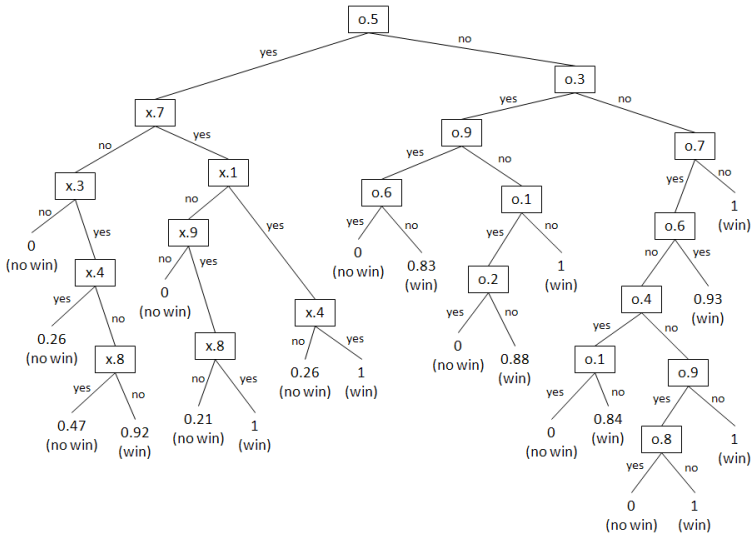
4.5.4 Tic-Tac-Toe

Our final example is the Tic-Tac-Toe dataset. Each data point represents a board configuration at the end of a tic-tac-toe game where player x played first, and the classification problem is to identify whether player x won. This is an easy task for a human, who just has to see if there are three x's in a line. There are nine features in the original data, each representing a square on a tic-tac-toe board. The possible values for each feature are: x, o, or b (player x, player o, or blank). We use 27 binary variables to capture the board configurations. In total, RuleGen generated 11 rules that predict that player x does not win and 43 rules that predict that player x wins.

Figure 4-6 shows the CART classifier. The ORC classifier, shown in Figure 4-7, is much simpler and decides the class of a board the same way a typical human would: if the board has three x's in a line, which can occur in eight different configurations, then player x wins; otherwise, player x does not win. It achieves perfect accuracy, whereas the accuracy of CART is about 0.94.

4.6 Summary

In this work, we have developed algorithms for producing interpretable, yet accurate, classifiers. The classifiers we construct are decision lists, which use association rules as building blocks. Both of the challenges addressed in this work, namely the task of



Notation	
x.1	'x' in box 1
o.1	'o' in box 1
b.1	box 1 is blank
x.2	'x' in box 2
o.2	'o' in box 2
b.2	box 2 is blank
x.3	'x' in box 3
⋮	⋮

Figure 4-6: CART Classifier for Tic-Tac-Toe data (predicted class in parentheses).

1	win			x
	54			x
	54			x

2	win	x		
	61		x	
	61			x

3	win	x	x	x
	42			
	42			

4	win			
	54			
	54	x	x	x

5	win	x		
	57	x		
	57	x		

6	win		x	
	61		x	
	61		x	

7	win			x
	54		x	
	54	x		

8	win			
	55	x	x	x
	55			

9	no win			
	215			
	638			

Figure 4-7: ORC classifier for Tic-Tac-Toe data (predicted class, $\bar{s}$, and $\bar{s}_X$ on left).

mining interesting rules, and the task of ordering them, have always been hampered by "combinatorial explosion." Even with a modest number of items in the dataset, there may be an enormous number of possible rules, and even with a modest number of rules, there are an enormous number of ways to order them. On the other hand, MIO methods are naturally suited to handle such problems; they not only encode the combinatorial structure of rule mining and rule ordering problems, but also are able to capture the new forms of regularization introduced in this work, that is, favoring more compact rules and shorter lists.

Our computational experiments show that ORC competes well in terms of accuracy against the top classification algorithms on a variety of datasets. In this work, we

used only one setting of the parameters for all of the experiments to show that even an "untuned" version of our algorithm performs well; however, by varying these parameters, it may be possible to achieve still better predictive performance. Since this research is among the first to use MIO methods for machine learning, and in particular to create decision lists using optimization-based (non-heuristic) approaches, it opens the door for further research on how to use optimization-based approaches for rule mining, forming interpretable classifiers, and handling new forms of regularization.

Appendix: Details from Experiments

In Section 4.4, we summarized training and test classification accuracies that were averaged over three folds for each dataset. In Tables 4.11 through 4.20 of this appendix, we show the accuracy for each of the individual folds. Train12 and Test3 refer to training on Folds 1 and 2 and testing on Fold 3. Train13 and Test2 refer to training on Folds 1 and 3 and testing on Fold 2. Train23 and Test1 refer to training on Folds 2 and 3 and testing on Fold 1. $\mathbf{L_{-1}}$ and $\mathbf{L_1}$ are the numbers of rules generated by RuleGen for class -1 and class 1 respectively. For these tables, Time_1 is the total time for generating all $\mathbf{L_{-1}} + \mathbf{L_1}$ rules; Time_2 is the time when the final solution was found, either before solving to optimality or before being terminated after a specified amount of time. Runtimes for the other methods were too small to be a significant factor in assessment. Here we list specific details for individual datasets:

1. **SPECT Heart.** We terminated each run of (4.17) after three hours.
2. **Haberman's Survival.** The original data set has 3 features, and we expanded the first and third features into 10 binary features. We terminated each run of (4.17) after 10 minutes.
3. **MONK's Problem 1.** For (4.17), it took 83, 602, and 5645 seconds to solve to optimality for Train12, Train13, and Train23 respectively, even though it took substantially less time to find the final solution.
4. **MONK's Problem 2.** We terminated each run of (4.17) after three hours.
5. **MONK's Problem 3.** (4.17) solved to optimality when the last solution was found.
6. **Congressional Voting Records.** We terminated each run of (4.17) after seven hours.
7. **Breast Cancer Wisconsin (Original).** The dataset has 699 rows, each representing a patient. There are $n = 683$ remaining transactions after removing rows with missing values. There are nine original attributes, each taking integer values between 1 and 10. We used categorical variables to capture whether each

Table 4.11: Classification accuracy on SPECT Heart dataset.

	LogReg	SVM	CART	Adaboost	ORC	L_{-1}	L_1	$Time_1$	$Time_2$
Train12	0.8426966	0.8258427	0.8258427	0.8651685	0.8426966	10	127	73	9611
Test3	0.8314607	0.9101124	0.8314607	0.8314607	0.7977528	–	–	–	–
Train13	0.9213483	0.8988764	0.8651685	0.8932584	0.9213483	6	144	62	10686
Test2	0.7752809	0.7865169	0.7640450	0.7865169	0.7977528	–	–	–	–
Train23	0.8707865	0.8651685	0.8258427	0.8707865	0.9269663	10	139	80	6289
Test1	0.7865169	0.8426966	0.7528090	0.8426966	0.7303371	–	–	–	–

Table 4.12: Classification accuracy on Haberman's Survival dataset.

	LogReg	SVM	CART	Adaboost	ORC	L_{-1}	L_1	$Time_1$	$Time_2$
Train12	0.7450980	0.7647059	0.7450980	0.7647059	0.7401961	6	10	19	9
Test3	0.8039216	0.7549020	0.7941176	0.7549020	0.8039216	–	–	–	–
Train13	0.7745098	0.7892157	0.7696078	0.7745098	0.7794118	4	11	11	5
Test2	0.7549020	0.7450980	0.7156863	0.7549020	0.7549020	–	–	–	–
Train23	0.7941176	0.8088235	0.7892157	0.7892157	0.7843137	5	10	14	5
Test1	0.7156863	0.7156863	0.7156863	0.7156863	0.7156863	–	–	–	–

attribute is between 1–4, 5–7, and 8–10, so in total there are $d = 27$ items. We terminated each run of (4.17) after four hours.

8. **Mammographic Mass.** The dataset has 961 rows, each representing a patient. There are $n = 830$ remaining transactions after removing rows with missing values. After transforming the attributes using binary variables, there are $d = 25$ items. The patient ages were categorized into seven bins: 29 and under, 30–39, 40–49, 50–59, 60–69, 70–79, and 80 and over. We terminated each run of (4.17) after three hours.

9. **Tic-Tac-Toe.** The original data set has 9 features, which we expanded to $d = 27$ binary variables to capture the board configurations. For (4.17), Train12 solved to optimality in 1271 seconds; Train13 and Train23 had optimality gaps of about 0.02% and 0.01% respectively when the final solutions were found, and we terminated solving after three hours.

10. **Car Evaluation.** The original data set has 9 features, which we expanded to 21 binary features. We terminated each run of (4.17) after seven hours.

We performed one set of experiments that involved tuning parameters. Table 4.21 shows the average classification accuracy for three methods. The first is SVM with $C = 1$; the second is a tuned version of SVM, where we varied the C parameter and chose the one with the best average test performance in hindsight; and the third

Table 4.13: Classification accuracy on MONK's Problem 1 dataset.

	LogReg	**SVM**	**CART**	**Adaboost**	**ORC**	$\mathbf{L_{-1}}$	$\mathbf{L_1}$	$\mathbf{Time_1}$	$\mathbf{Time_2}$
Train12	0.7534722	1	0.9236111	1	1	19	33	96	58
Test3	0.7430556	1	0.9027778	1	1	–	–	–	–
Train13	0.7465278	0.9861111	0.9305556	1	1	21	35	102	338
Test2	0.7569444	0.9444444	0.8611111	1	1	–	–	–	–
Train23	0.7569444	0.9861111	0.9305556	1	1	19	39	105	344
Test1	0.7361111	0.9166667	0.8888889	1	1	–	–	–	–

Table 4.14: Classification accuracy on MONK's Problem 2 dataset.

	LogReg	**SVM**	**CART**	**Adaboost**	**ORC**	$\mathbf{L_{-1}}$	$\mathbf{L_1}$	$\mathbf{Time_1}$	$\mathbf{Time_2}$
Train12	0.6562500	0.6736111	0.7569444	0.7430556	0.8472222	28	21	135	2786
Test3	0.6388889	0.6666667	0.6666667	0.6388889	0.7638889	–	–	–	–
Train13	0.6666667	0.6770833	0.7187500	0.7638889	0.8055556	30	16	125	8440
Test2	0.6250000	0.6597222	0.5972222	0.6944444	0.6944444	–	–	–	–
Train23	0.6180556	0.6701389	0.7743056	0.7500000	0.8368056	27	14	112	4717
Test1	0.5416667	0.6875000	0.7430556	0.6180556	0.7430556	–	–	–	–

Table 4.15: Classification accuracy on MONK's Problem 3 dataset.

	LogReg	**SVM**	**CART**	**Adaboost**	**ORC**	$\mathbf{L_{-1}}$	$\mathbf{L_1}$	$\mathbf{Time_1}$	$\mathbf{Time_2}$
Train12	1	0.9756944	1	1	1	41	18	97	961
Test3	1	0.9652778	1	1	1	–	–	–	–
Train13	1	0.9965278	1	1	1	35	27	102	1008
Test2	1	0.9791667	1	1	1	–	–	–	–
Train23	1	0.9861111	1	1	1	31	23	102	224
Test1	1	0.9722222	1	1	1	–	–	–	–

Table 4.16: Classification accuracy on Congressional Voting Records dataset.

	LogReg	**SVM**	**CART**	**Adaboost**	**ORC**	$\mathbf{L_{-1}}$	$\mathbf{L_1}$	$\mathbf{Time_1}$	$\mathbf{Time_2}$
Train12	1	0.9758620	0.9689655	0.9827586	0.9827586	110	116	103	22899
Test3	0.9310345	0.9517241	0.9448276	0.9586207	0.9517241	–	–	–	–
Train13	0.9620690	0.9724138	0.9620690	0.9620690	0.9689655	137	146	109	20536
Test2	0.9862069	0.9517241	0.9448276	0.9586207	0.9517241	–	–	–	–
Train23	0.9827586	0.9758620	0.9482759	0.9724138	0.9724138	141	148	113	21082
Test1	0.9586207	0.9655172	0.9724138	0.9517241	0.9655172	–	–	–	–

Table 4.17: Classification accuracy on Breast Cancer dataset.

	LogReg	**SVM**	**CART**	**Adaboost**	**ORC**	$\mathbf{L_{-1}}$	$\mathbf{L_1}$	$\mathbf{Time_1}$	$\mathbf{Time_2}$
Train12	0.9868421	0.9868421	0.9561404	0.9758772	0.9714912	123	66	551	12802
Test3	0.9515419	0.9515419	0.9515419	0.9559471	0.9603524	–	–	–	–
Train13	0.9648352	0.9824176	0.9450550	0.9538462	0.9692308	136	53	637	11703
Test2	0.9912280	0.9780702	0.9561404	0.9912280	0.9561404	–	–	–	–
Train23	0.9846154	0.9846154	0.9670330	0.9780220	0.9890110	135	82	661	14373
Test1	0.9078947	0.9561404	0.9385965	0.9385965	0.9429825	–	–	–	–

Table 4.18: Classification accuracy on Mammographic Mass dataset.

	LogReg	**SVM**	**CART**	**Adaboost**	**ORC**	$\mathbf{L_{-1}}$	$\mathbf{L_1}$	$\mathbf{Time_1}$	$\mathbf{Time_2}$
Train12	0.8465704	0.8628159	0.8447653	0.8519856	0.8501805	33	26	637	1340
Test3	0.8514493	0.8297101	0.8297101	0.8514493	0.8297101	–	–	–	–
Train13	0.8354430	0.8643761	0.8336347	0.8499096	0.8390597	29	28	706	2498
Test2	0.8519856	0.8411552	0.8519856	0.8628159	0.8555957	–	–	–	–
Train23	0.8625678	0.8788427	0.8481013	0.8661844	0.8716094	29	30	669	7422
Test1	0.8086643	0.7942238	0.8086643	0.8122744	0.8158845	–	–	–	–

Table 4.19: Classification accuracy on Tic-Tac-Toe dataset.

	LogReg	**SVM**	**CART**	**Adaboost**	**ORC**	$\mathbf{L_{-1}}$	$\mathbf{L_1}$	$\mathbf{Time_1}$	$\mathbf{Time_2}$
Train12	0.9921630	0.9545455	0.9388715	0.9968652	1	11	43	1278	1232
Test3	0.9656250	0.9031250	0.8812500	0.9687500	1	–	–	–	–
Train13	0.9765258	0.9593114	0.9295775	0.9921753	1	14	36	1202	3292
Test2	0.9937304	0.9467085	0.8934170	0.9811912	1	–	–	–	–
Train23	0.9812207	0.9342723	0.9358372	0.9921753	1	12	44	1244	7570
Test1	0.9874608	0.8996865	0.8871473	0.9749216	1	–	–	–	–

Table 4.20: Classification accuracy on Car Evaluation dataset.

	LogReg	**SVM**	**CART**	**Adaboost**	**ORC**	$\mathbf{L_{-1}}$	$\mathbf{L_1}$	$\mathbf{Time_1}$	$\mathbf{Time_2}$
Train12	0.9574653	0.9826389	0.9696180	0.9965278	0.9539931	45	13	598	8241
Test3	0.9513889	0.9756944	0.9670139	0.9913194	0.9496528	–	–	–	–
Train13	0.9609375	0.9800347	0.9652778	0.9965278	0.9548611	45	13	610	4952
Test2	0.9461806	0.9774306	0.9583333	0.9947917	0.9479167	–	–	–	–
Train23	0.9557292	0.9835070	0.9626736	0.9956597	0.9704861	48	10	911	8813
Test1	0.9479167	0.9652778	0.9600694	0.9861111	0.9548611	–	–	–	–

is ORC. The results for SVM with $C = 1$ and ORC are repeated from Table 4.6. Table 4.21 shows that the overall performance of the untuned ORC algorithm is still on par with that of the tuned SVM algorithm.

Table 4.21: Results of tuned SVM (highest in row highlighted in bold).

Dataset		**SVM** (default $C = 1$)	**SVM** (hindsight)	C	**ORC**
SPECT	train	0.8633 ± 0.0366	0.8745 ± 0.0319	1.2	$\mathbf{0.8970 \pm 0.0471}$
	test	0.8464 ± 0.0619	$\mathbf{0.8502 \pm 0.0566}$		0.7753 ± 0.0389
Haberman	train	$\mathbf{0.7876 \pm 0.0221}$	0.7761 ± 0.0279	0.4	0.7680 ± 0.0242
	test	0.7386 ± 0.0204	$\mathbf{0.7582 \pm 0.0442}$		$\mathbf{0.7582 \pm 0.0442}$
MONK1	train	0.9907 ± 0.0080	**1**	1.4	**1**
	test	0.9537 ± 0.0424	**1**		**1**
MONK2	train	0.6736 ± 0.0035	0.6736 ± 0.0035	1	$\mathbf{0.8299 \pm 0.0217}$
	test	0.6713 ± 0.0145	0.6713 ± 0.0145		$\mathbf{0.7338 \pm 0.0356}$
MONK3	train	0.9861 ± 0.0104	**1**	1.6	**1**
	test	0.9722 ± 0.0069	**1**		**1**
Votes	train	0.9747 ± 0.0020	$\mathbf{0.9793 \pm 0.0034}$	1.4	0.9747 ± 0.0072
	test	0.9563 ± 0.0080	$\mathbf{0.9586 \pm 0.0069}$		0.9563 ± 0.0080
B.Cancer	train	0.9846 ± 0.0022	$\mathbf{0.9868 \pm 0.0044}$	1.2	0.9766 ± 0.0108
	test	0.9619 ± 0.0142	$\mathbf{0.9634 \pm 0.0203}$		0.9532 ± 0.0091
Mammo	train	$\mathbf{0.8687 \pm 0.0088}$	0.8608 ± 0.0095	0.6	0.8536 ± 0.0165
	test	0.8217 ± 0.0245	0.8313 ± 0.0197		$\mathbf{0.8337 \pm 0.0202}$
TicTac	train	0.9494 ± 0.0133	0.9901 ± 0.0009	6	**1**
	test	0.9165 ± 0.0262	0.9844 ± 0.0143		**1**
CarEval	train	0.9821 ± 0.0018	**1**	7.2	0.9598 ± 0.0093
	test	0.9728 ± 0.0066	$\mathbf{0.9988 \pm 0.0010}$		0.9508 ± 0.0036

Chapter 5

Conclusion

In a recent article by the McKinsey Global Institute, reporters estimated that by 2009, nearly all economic sectors in the United States had on average over 200 terabytes (200,000 gigabytes) of data per company with at least 1000 employees.[1] The growing availability of "big data" allows companies to conduct ever more precise studies about their clients, products, services, transactions, inventory, equipment, etc., thus giving them the opportunity and capability to make increasingly informed decisions. In an age in which analytics is of primary interest for many businesses and industries, both MIO and machine learning are highly useful in their own right. The broad aspiration of this thesis was to explore the intersection of these two analytic paradigms.

MIO involves the mathematical modeling and optimization of the exact objective of interest in order to make the best possible decision, and its applications span a tremendous range of areas such as healthcare, energy, transportation, and sports. Machine learning concerns the design of efficient algorithms that automatically learn complex patterns from data, giving rise to computing applications such as spam detection, object recognition in images, and web search. The companies of today's business world amass tremendous amounts of data, prompting the need to be able to make optimal choices based on data-driven methods. These circumstances motivated us to consider the combination of MIO and machine learning.

[1] http://www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation

MIO is a natural framework for a number of machine learning problems that have loss functions with discrete components. In this thesis, we designed MIO-based algorithms for supervised ranking, association rule mining, and associative classification. Whereas most conventional machine learning methods are heuristic in nature and not designed to find truly optimal solutions, MIO methods provide a means to optimize the exact objectives of interest. In computational experiments, our MIO approach either matched or improved upon state-of-the-art machine learning methods. We hope that this work inspires and guides further research in the development of mathematical programming methodologies for problems in machine learning.

Bibliography

Shivani Agarwal, Deepak Dugar, and Shiladitya Sengupta. Ranking chemical structures for drug discovery: A new machine learning approach. *Journal of Chemical Information and Modeling*, 50(5):716–731, April 2010.

Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 487–499, 1994.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

Martin Anthony. Decision lists. Technical report, CDAM Research Report LSE-CDAM-2005-23, 2005.

Arthur Asuncion and David J. Newman. UCI machine learning repository, 2007. URL `http://www.ics.uci.edu/ mlearn/MLRepository.html`.

Kaan Ataman, W. Nick Street, and Yi Zhang. Learning to rank by maximizing AUC with linear programming. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 123–129, 2006.

Eddie Bajek. (Almost) How the Elias Sports Bureau rankings work, 2008. Unpublished blog entry at http://tigers-thoughts.blogspot.com/2008/07/almost-how-elias-sports-bureau-rankings.html.

Roberto J. Bayardo and Rakesh Agrawal. Mining the most interesting rules. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 145–154, 1999.

E.M.L. Beale and J.A. Tomlin. Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variable. In J.R. Lawrence, editor, *Proceedings of the fifth international conference on operational research*, pages 447–454. Tavistock, London & Wiley, New York, 1970.

James Bennett and Stan Lanning. The Netflix Prize. In *Proceedings of KDD Cup and Workshop*, 2007.

Dimitris Bertsimas and Romy Shioda. Classification and regression via integer optimization. *Operations Research*, 55(2):252–271, 2007.

Dimitris Bertsimas and Robert Weismantel. *Optimization Over Integers*. Dynamic Ideas, 2005.

Robert E. Bixby, Mary Fenelon, and Zonghao Gu. MIP: Theory and practice – closing the gap. In *System Modelling and Optimization: Methods, Theory, and Applications*. Kluwer Academic Publishers, 2000.

Endre Boros, Peter L. Hammer, Toshihide Ibaraki, Alexander Kogan, Eddy Mayoraz, and Ilya Muchnik. An implementation of logical analysis of data. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):292–306, 2000.

Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, July 1997.

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

J. Paul Brooks. Support vector machines with the ramp loss and the hard margin loss. *Operations Research*, 2010.

Oliver Büchter and Rüdiger Wirth. Discovery of association rules over ordinal data: A new and faster algorithm and its application to basket analysis. 1394:36–47, 1998.

Christopher Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, 2005.

Christopher J. Burges, Robert Ragno, and Quoc V. Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 395–402, 2006.

Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 129–136, New York, NY, USA, 2007. ACM.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 161–168, 2006.

Rich Caruana, Shumeet Baluja, and Tom Mitchell. Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation. In *Advances in Neural Information Processing Systems (NIPS)*, volume 8, pages 959–965, 1996.

Volkan Cevher and Andreas Krause. Greedy dictionary selection for sparse representation. In *NIPS 2009 Workshop on Discrete Optimization in Machine Learning (DISCML)*, 2009.

Seth J. Chandler. Analyzing US News and World Report rankings using symbolic regression, 2006. Unpublished blog entry at http://taxprof.typepad.com/taxprof_blog/files/analyzing_u.S. News & World Report Rankings Using Symbolic Regression.pdf.

Stéphan Clemençon and Nicolas Vayatis. Ranking the best instances. *Journal of Machine Learning Research*, 8:2671–2699, 2007.

Stéphan Clemençon and Nicolas Vayatis. Empirical performance maximization for linear rank statistics. *Advances in Neural Information Processing Systems (NIPS)*, 21, 2008.

Stéphan Clemençon, Gábor Lugosi, and Nicolas Vayatis. Ranking and empirical minimization of u-statistics. *Annals of Statistics*, 36(2):844–874, 2008.

Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Journal of Association for Computational Linguistics*, pages 175–182, 2003.

William S. Cooper, Aitao Chen, and Fredric C. Gey. Full text retrieval based on probabilistic equations with coefficients fitted by logistic regression. In *Proceedings of the 2nd Text Retrieval Conference (TREC-2)*, pages 57–66, 1994.

David Cossock and Tong Zhang. Subset ranking using regression. In *Conference on Learning Theory (COLT)*, pages 605–619, 2006.

George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

Lori E. Dodd and Margaret S. Pepe. Partial AUC estimation and regression. *UW Biostatistics Working Paper Series*, 2003. URL `http://www.bepress.com/uwbiostat/paper181`.

Stephan Dreiseitl and Lucila Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics*, 35:352–359, 2002.

Seyda Ertekin and Cynthia Rudin. On equivalence relationships between classification and ranking algorithms. *Journal of Machine Learning Research*, 12:2905–2929, 2011.

Cèsar Ferri, Peter Flach, and José Hernández-Orallo. Learning decision trees using the area under the ROC curve. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, pages 139–146. Morgan Kaufmann, 2002.

Michael J. Fine, Thomas E. Auble, Donald M. Yealy, Barbara H. Hanusa, Lisa A. Weissfeld, Daniel E. Singer, Christopher M. Coley, Thomas J. Marrie, and Wishwa N. Kapoor. A prediction rule to identify low-risk patients with community-acquired pneumonia. *The New England Journal of Medicine*, pages 243–250, 1997.

Yoav Freund and Robert Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Computational Learning Theory*, 904: 23–37, 1995.

Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4: 933–969, 2003a.

Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4: 933–969, 2003b.

Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954, 2008.

Johannes Fürnkranz, Eyke Hüllermeier, and Stijn Vanderlooy. Binary decomposition methods for multipartite ranking. *Machine Learning and Knowledge Discovery in Databases*, 5781:359–374, 2009.

Cyril Furtlehner and Marc Schoenauer. Multi-objective 3-sat with survey-propagation. In *NIPS 2010 Workshop on Discrete Optimization in Machine Learning (DISCML)*, 2010.

Liqiang Geng and Howard J. Hamilton. Interestingness measures for data mining: a survey. *ACM Computing Surveys*, 38, September 2006.

Bart Goethals. Survey on frequent pattern mining. Technical report, Helsinki Institute for Information Technology, 2003.

Paul E. Green, Abba M. Krieger, and Yoram (Jerry) Wind. Thirty years of conjoint analysis: Reflections and prospects. *Interfaces*, 31(3):S56–S73, 2001.

Phil Gross, Ansaf Salleb-Aouissi, Haimonti Dutta, and Albert Boulanger. Ranking electrical feeders of the New York power grid. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA)*, pages 725–730, 2009.

Peter L. Hammer, Alexander Kogan, and Miguel A. Lejeune. Reverse-engineering banks' financial strength ratings using logical analysis of data. Working paper available at http://www.optimization-online.org/DB_FILE/2007/02/1581.pdf, 2007.

Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15: 55–86, 2007.

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, New York, 2001.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers*, 2000.

Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. *SIGKDD Explorations*, 2: 58–64, June 2000.

Eric I. Hsu and Sheila A. McIlraith. Deriving height-based bounds by relaxing the maximum satisfiability problem. In *NIPS 2010 Workshop on Discrete Optimization in Machine Learning (DISCML)*, 2010.

Kalervo Järvelin and Jaana Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 41–48, 2000.

Dennis L. Jennings, Teresa M. Amabile, and Lee Ross. Informal covariation assessments: Data-based versus theory-based judgements. In Daniel Kahneman, Paul Slovic, and Amos Tversky, editors, *Judgment Under Uncertainty: Heuristics and Biases,*, pages 211–230. Cambridge Press, Cambridge, MA, 1982.

Heng Ji, Cynthia Rudin, and Ralph Grishman. Re-ranking algorithms for name tagging. In *Proceedings of the HLT-NAACL Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, pages 49–56, 2006.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2002.

Ellis L. Johnson, George L. Nemhauser, and Martin W.P. Savelsbergh. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing*, 2000.

Maurice G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

Adam R. Klivans and Rocco A. Servedio. Toward attribute efficient learning of decision lists and parities. *Journal of Machine Learning Research*, 7:587–602, 2006.

Wojciech Kotłowski, Krzysztof Dembczyński, and Eyke Hüllermeier. Bipartite ranking through minimization of univariate loss. In *ICML*, pages 1113–1120, 2011.

John Lafferty and Chengxiang Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 111–119, 2001.

Quoc V. Le and Alex Smola. Direct optimization of ranking measures. arXiv:0704.3359v1 [cs.IR], 2007.

Quoc V. Le, Alex Smola, Olivier Chapelle, and Choon Hui Teo. Optimization of ranking measures. *Journal of Machine Learning Research*, pages 1–48, 2010.

Ping Li, Christopher J. Burges, and Qiang Wu. McRank: Learning to rank using multiple classification and gradient boosting. In *Advances in Neural Information Processing Systems (NIPS)*, pages 845–852, 2007.

Wenmin Li, Jiawei Han, and Jian Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. *IEEE International Conference on Data Mining*, pages 369–376, 2001.

Hui Lin and Jeff Bilmes. An application of the submodular principal partition to training data subset selection. In *NIPS 2010 Workshop on Discrete Optimization in Machine Learning (DISCML)*, 2010.

Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 80–96, 1998.

Philip M. Long and Rocco A. Servedio. Attribute-efficient learning of decision lists and linear threshold functions under unconcentrated distributions. In *Advances in Neural Information Processing Systems (NIPS)*, volume 19, pages 921–928, 2007.

Luigi Malagò, Matteo Matteucci, and Giovanni Pistone. Stochastic relaxation as a unifying approach in 0/1 programming. In *NIPS 2009 Workshop on Discrete Optimization in Machine Learning (DISCML)*, 2009.

Mario Marchand and Marina Sokolova. Learning with decision lists of data-dependent features. *Journal of Machine Learning Research*, 2005.

Irina Matveeva, Andy Laucius, Christopher Burges, Leon Wong, and Timo Burkard. High accuracy retrieval with multiple nested ranker. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 437–444, 2006.

Tyler McCormick, Cynthia Rudin, and David Madigan. A hierarchical model for association rule mining of sequential events: An approach to automated medical symptom prediction. *MIT DSpace, Operations Research Center*, 2011.

Ken McGarry. A survey of interestingness measures for knowledge discovery. *The Knowledge Engineering Review*, 20:39–61, 2005.

Nicolai Meinshausen. Node harvest. *The Annals of Applied Statistics*, 4(4):2049–2072, 2010.

Charles E. Metz. Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, 8 (4):283–298, 1978.

George A. Miller. The magical number seven, plus or minus two: Some limits to our capacity for processing information. *The Psychological Review*, 63(2):81–97, 1956.

Gretchen Morgenson and Louise Story. Rating agency data aided wall street in deals. New York Times, Business section, April 24, 2010. Article at http://www.nytimes.com/2010/04/24/business/24rating.html.

Hai Nguyen, Katrin Franke, and Slobodan Petrović. Optimizing a class of feature selection measures. In *NIPS 2009 Workshop on Discrete Optimization in Machine Learning (DISCML)*, 2009.

Nils J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

Claudia Perlich, Foster Provost, and Jeffrey S. Simonoff. Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research*, 4: 211–255, 2003.

Joseph Putter. The treatment of ties in some nonparametric tests. *The Annals of Mathematical Statistics*, 26(3):368–386, 1955.

Shyamsundar Rajaram and Shivani Agarwal. Generalization bounds for k-partite ranking. In *NIPS 2005 Workshop on Learning to Rank*, 2005.

Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

Ulrich Rückert. *A Statistical Approach to Rule Learning*. PhD thesis, Technischen Universität München, 2008.

Cynthia Rudin. The P-Norm Push: A simple convex ranking algorithm that concentrates at the top of the list. *Journal of Machine Learning Research*, 10:2233–2271, 2009.

Cynthia Rudin, Rebecca Passonneau, Axinia Radeva, Haimonti Dutta, Steve Ierome, and Delfina Isaac. A process for predicting manhole events in Manhattan. *Machine Learning*, 80:1–31, 2010.

Cynthia Rudin, Benjamin Letham, Ansaf Salleb-Aouissi, Eugen Kogan, and David Madigan. Sequential event prediction with association rules. In *Proceedings of the 24th Annual Conference on Learning Theory (COLT)*, 2011.

I. Richard Savage. Nonparametric statistics. *Journal of the American Statistical Association*, 52(279):331–344, 1957.

Libin Shen and Aravind K. Joshi. An SVM based voting algorithm with application to parse reranking. In *Proc. HLT-NAACL 2003 workshop on Analysis of Geographic References*, pages 9–16, 2003.

György J. Simon, Vipin Kumar, and Peter W. Li. A simple statistical model and association rule filtering for classification. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 823–831, 2011.

Marina Sokolova, Mario Marchand, Nathalie Japkowicz, and John Shawe-Taylor. The decision list machine. In *Advances in Neural Information Processing Systems (NIPS)*, volume 15, pages 921–928, 2003.

Sören Sonnenburg, Gunnar Rätsch, and Christin Schäfer. Learning interpretable svms for biological sequence classification. 3500(995):389–407, 2005.

Ao-Jan Su, Y. Charlie Hu, Aleksandar Kuzmanovic, and Cheng-Kok Koh. How to improve your Google ranking: Myths and reality. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 1, pages 50–57, 2010.

Ajit C. Tamhane and Dorothy D. Dunlop. *Statistics and data analysis*. Prentice Hall, 2000.

Pang-Ning Tan and Vipin Kumar. Interestingness measures for association patterns: a perspective. Technical report, Department of Computer Science, University of Minnesota, 2000.

Fadi Thabtah. A review of associative classification mining. *The Knowledge Engineering Review*, 22:37–65, March 2007.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, Yasemin Altun, and Yoram Singer. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

Koen Vanhoof and Benoît Depaire. Structure of association rule classifiers: a review. In *Proceedings of the International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 9–12, 2010.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

Dennis D. Wackerly, William Mendenhall III, and Richard L. Scheaffer. *Mathematical statistics with applications*. Duxbury, 2002.

Jun Xu. A boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

Jun Xu, Tie Yan Liu, Min Lu, Hang Li, and Wei Ying Ma. Directly optimizing evaluation measures in learning to rank. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2008.

Guizhen Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 344–353, 2004.

Xiaoxin Yin and Jiawei Han. CPAR: Classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 331–335, 2003.