Theses and Dissertations

2007

Learning to rank by maximizing the AUC with linear programming for problems with binary output

Kaan Ataman
University of Iowa

Recommended Citation

Ataman, Kaan. "Learning to rank by maximizing the AUC with linear programming for problems with binary output." PhD (Doctor of Philosophy) thesis, University of Iowa, 2007.
http://ir.uiowa.edu/etd/151.

LEARNING TO RANK BY MAXIMIZING THE AUC WITH LINEAR
PROGRAMMING FOR PROBLEMS WITH BINARY OUTPUT

by

Kaan Ataman

An Abstract

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Business Administration in the
Graduate College of The
University of Iowa

July 2007

Thesis Supervisor: Associate Professor W. Nick Street

ABSTRACT

Ranking is a popular machine learning problem that has been studied extensively for more then a decade. Typical machine learning algorithms are generally built to optimize predictive performance (usually measured in accuracy) by minimizing classification error. However, there are many real world problems where correct ordering of instances is of equal or greater importance than correct classification. Learning algorithms that are built to minimize classification error are often not effective when ordering within or among classes. This gap in research created a necessity to alter the objective of such algorithms to focus on correct ranking rather then classification.

Area Under the ROC Curve (AUC), which is equivalent to the Wilcoxon-Mann-Whitney (WMW) statistic, is a widely accepted performance measure for evaluating ranking performance in binary classification problems. In this work we present a linear programming approach (LPR), similar to 1-norm Support Vector Machines (SVM), for ranking instances with binary outputs by maximizing an approximation to the WMW statistic. Our formulation handles non-linear problems by making use of kernel functions. Results on several well-known benchmark datasets show that our approach ranks better than 2-norm SVM and faster than the support vector ranker (SVR).

The number of constraints in the linear programming formulation increases quadratically with the number of data points considered for the training of the algorithm. We tackle this problem by implementing a number of exact and approximate speed-up approaches inspired by well-known methods such as chunking, clustering and subgradient methods. The subgradient method is the most promising because of its solution quality and its fast convergence to the optimal solution.

We adopted the LPR formulation to survival analysis. With this approach it is possible to order subjects by risk for experiencing an event. Such an ordering enables determination of high-risk and low-risk groups among the subjects that can be helpful not only in

medical studies but also in engineering, business and social sciences. Our results show that our algorithm is superior in time-to-event prediction to the most popular survival analysis tool, Cox's proportional hazard regression.

Abstract Approved: _____
Thesis Supervisor

_____
Title and Department

_____
Date

LEARNING TO RANK BY MAXIMIZING THE AUC WITH LINEAR
PROGRAMMING FOR PROBLEMS WITH BINARY OUTPUT

by

Kaan Ataman

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Business Administration in the
Graduate College of The
University of Iowa

July 2007

Thesis Supervisor: Associate Professor W. Nick Street

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

_____

PH.D. THESIS

_____

This is to certify that the Ph.D. thesis of

Kaan Ataman

has been approved by the Examining Committee
for the thesis requirement for the Doctor of
Philosophy degree in Business Administration
at the July 2007 graduation.

Thesis Committee: _____
                  W. Nick Street, Thesis Supervisor

                  _____
                  Samuel Burer

                  _____
                  Faiz Currim

                  _____
                  Padmini Srinivasan

                  _____
                  Hwanjo Yu

To mom and dad

I am not leaving a spiritual legacy of dogmas, unchangeable petrified directives. My spiritual legacy is science and reason.

M. Kemal Ataturk

ACKNOWLEDGMENTS

My PhD adventure began on a warm September afternoon when I visited Nick at his office. I didn't know much about him then, except he was the only person who shared the same research interests as I do. I visited him many more times that year. He must have believed in me. The following fall my PhD journey officially started. I soon realized that I couldn't have asked for a better person to guide me through this journey. He has been a great mentor, whom I learned a lot from. Now, I look back all these years past in Iowa City, I am glad I knocked on Nick's door that September afternoon.

I would like to thank my dissertation committee members, Samuel Burer, Faiz Currim, Padmini Srinivasan and Hwanjo Yu for their invaluable suggestions, critical eye and feedback throughout my PhD studies. To this day I still think Sam's generous evaluation in my comprehensive exam was the main reason I was able to move forward smoothly in my PhD studies.

Most of my time has passed in the ISOR lab (formerly known as CASTOR lab). It has always been a pleasant environment to work and socialize among the PhD students of our department. I would like to thank all the past and present regulars of the room S204; Brian, Chris, Ding, Gautam, Justin, Matt, Thaddeus, Xin Ying and Yi for their valuable friendship.

A special thanks goes to our department secretary Barbara Carr and our College of Business PhD coordinator Renea Jay. They made my life easier day in and day out. They were always one step ahead of me for reminding me of deadlines, filing paper works and handling all the other necessities that goes behind the scenes.

I would also like to thank all my friends in Iowa City for making the life outside the school fun through the years. I will never forget our regular get-togethers at Ugur and Firdevs' place. I feel very lucky to know such great individuals.

Last but not the least; I would like to thank my mom and dad, Bilge and Yener, and

ABSTRACT

Ranking is a popular machine learning problem that has been studied extensively for more then a decade. Typical machine learning algorithms are generally built to optimize predictive performance (usually measured in accuracy) by minimizing classification error. However, there are many real world problems where correct ordering of instances is of equal or greater importance than correct classification. Learning algorithms that are built to minimize classification error are often not effective when ordering within or among classes. This gap in research created a necessity to alter the objective of such algorithms to focus on correct ranking rather then classification.

Area Under the ROC Curve (AUC), which is equivalent to the Wilcoxon-Mann-Whitney (WMW) statistic, is a widely accepted performance measure for evaluating ranking performance in binary classification problems. In this work we present a linear programming approach (LPR), similar to 1-norm Support Vector Machines (SVM), for ranking instances with binary outputs by maximizing an approximation to the WMW statistic. Our formulation handles non-linear problems by making use of kernel functions. Results on several well-known benchmark datasets show that our approach ranks better than 2-norm SVM and faster than the support vector ranker (SVR).

The number of constraints in the linear programming formulation increases quadratically with the number of data points considered for the training of the algorithm. We tackle this problem by implementing a number of exact and approximate speed-up approaches inspired by well-known methods such as chunking, clustering and subgradient methods. The subgradient method is the most promising because of its solution quality and its fast convergence to the optimal solution.

We adopted the LPR formulation to survival analysis. With this approach it is possible to order subjects by risk for experiencing an event. Such an ordering enables determination of high-risk and low-risk groups among the subjects that can be helpful not only in

medical studies but also in engineering, business and social sciences. Our results show that our algorithm is superior in time-to-event prediction to the most popular survival analysis tool, Cox's proportional hazard regression.

TABLE OF CONTENTS

ix

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
INTRODUCTION

Many real world data mining problems require an ordering instead of a simple classification. For example in direct mail marketing, the marketer may want to know which potential customers to target for mailing new catalogs so as to maximize the revenue. If the number of catalogs the marketer is sending is limited then it is not enough to know who is likely to buy a product after receiving the catalog. In this case it would be more useful to distinguish the top n-percent of potential customers who yield the highest likelihood of buying a product. By doing so, the marketer can more intelligently target a subset of her customer base, increasing the expected profit. A slightly different example is collaborative book recommendations. To intelligently recommend a book, an algorithm must distinguish the similarities of book preferences of people who have read and submitted a score for the book. Combining these scores it is possible to return a recommendation list to the user. Ideally this list would be ranked in such a way that the book on the top of the list is expected to appeal the most to the user. It is not hard to see that there is a slight difference between these two ranking problems. In the marketing problem the ranking is based on binary output (purchase, non-purchase) whereas in the book recommendation problem the ranking is based on a collection of partial rankings. In this dissertation we will be concentrating on the ranking problem with binary output.

Over the years, rank-optimizing versions of several learning algorithms have been developed as well as methods that directly optimize some ranking metric, which will be summarized in the next chapter. Among such methods, optimization-based approaches are quite promising, especially large margin methods such as support vector machines. One drawback of such methods have been the computational cost arising from the quadratic form of the optimization problem. Also, rank optimization requires a quadratic or greater number of constraints with respect to the number of data points. Each of these problems

has been addressed in the literature, yielding promising yet mixed results. In this work we tackle these problems in one comprehensive algorithm that utilizes linear programming. Throughout this dissertation we consider the problem of ranking in the context of binary classification where the data has binary outputs.

This dissertation first reviews the literature leading up to our work. The review introduces several key concepts necessary to follow our work and gradually leads the reader from general to specific in terms of the research done in this field. The review also points out the challenges and solutions that appear in the literature as a motivation to our work. Chapter 3 introduces our formulation of a linear program that optimizes ranking of points for problems where cases have only two possible outputs. In the same chapter we compare our approach with some state-of-the-art classification and ranking algorithms and point out the differences and benefits between these algorithms. In Chapter 4, we address the scaling issues of these algorithms. We introduce several exact and approximate heuristics to reduce the complexity of the problem and present our results. At the end of Chapter 4 we present a case study, where we solve a relatively large real world problem that would otherwise be unsolvable with the original formulation. Chapter 5 concentrates on how survival analysis problems can benefit from our algorithm. We demonstrate the performance on a number of real world problems and show that our approach produces favorable results compared to one of the most popular survival analysis approach, Cox's proportional hazard regression. We finish this dissertation with conclusions and possible future directions.

CHAPTER II
LITERATURE REVIEW

This chapter introduces ranking and algorithms that are built to perform ranking in problems with binary outputs. We first motivate the ranking problem. Then we focus on the class of ranking problems with binary outputs. We introduce ROC curves and the area under the ROC curve (AUC) as a performance evaluation metric. We survey the literature for ranking algorithms, more specifically the algorithms that optimize an approximation to the AUC metric or, equivalently, Wilcoxon-Mann-Whitney statistic. We specifically concentrate on rank-optimizing algorithms based on large margin classifiers such as support vector machines. We point out the common problems with such algorithms and possible solutions introduced in the literature. We finish the chapter with an overview of the other rank optimization approaches covered in the literature.

2.1   The Ranking Problem

As a broad subfield of artificial intelligence, machine learning is concerned with the design and development of algorithms and techniques that allow computers to *learn*. The major focus of machine learning research is to extract information from data automatically by computational and statistical methods. Hence, machine learning is closely related to data mining and statistics as well as theoretical computer science. Machine learning algorithms are organized into groups such as: supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning, where the first two are the most popular. The goal of supervised learning is to find a function that maps inputs (independent variables) to outputs (dependent variables), given a set of points with known outputs. Well-known supervised learning algorithms include decision trees [76, 77], artificial neural networks [81, 25], naive Bayes classifier [27], nearest neighbor algorithm [24] and support vector machines (SVMs) [91, 12]. On the other hand, unsupervised learning is a method

of machine learning where a model is fit to unlabeled observations. The most common algorithms include clustering [55] and self organizing maps [60].

Traditional supervised machine learning algorithms are built to minimize classification error. Most of these algorithms also produce a numeric output, such as class membership probabilities or a distance measure usually to a decision surface, etc. Relative values of such numeric outputs are generally ignored by classification algorithms when determining a class label. Many real world applications on the other hand require a ranking or at least a metric for evaluation, such as class membership probabilities. In other words, the real world question asks how positive (or negative) a point actually is. In the light of such research questions, ranking has become a popular machine learning problem that has been addressed extensively in the literature [14, 17, 92, 34, 22, 56].

A wide range of applications can be formulated as ranking problems. Ranking can be as simple as correctly ordering two data points, or it can be finding a total ranking from a collection of partial rankings. Learning to rank in problems with binary outputs can be seen as pairwise ranking of points belonging to one class against points belonging to the other class. This class of problems frequently occurs in the real world. A direct mail marketing example is already given in the introduction section. Another application would be ranking of company stocks in terms of expected return based only on their past up or down movements. Assume that the prediction problem is recommending a stock for buy or sell. One may want to maximize the expected profit or minimize the expected loss. Typically in this type of ranking problem, a subset of data from a category (or multiple categories) is the primary focus of the research problem such as the top $k\%$ of positives (buys) or bottom $k\%$ of negatives (sells). Another type of problem is the one where there is no definite class label but simply some ordered data. A book recommendation example is given in the introduction. Another example would be aggregation of search results on the web. Assume a scenario that a search engine provides search results and asks for feedback. Then a user can reorder a subset of such search results (or this process can be simulated

implicitly) to aid the algorithm towards finding a more accurate ranking.

Classification algorithms are optimized to label unseen instances correctly. However, there are a number of them that can rank fairly well. Caruana at al. showed that stand-alone algorithms such as neural networks, support vector machines, and ensemble methods (such as boosted or bagged decision trees), yield very good ranking performance in problems with binary outputs [15]. The authors also found out that among the individual methods SVM's are good rankers, especially when paired with radial basis function (RBF) kernels. Neural networks with a large number of hidden layers and the k-nearest neighbor algorithm with a large k are also shown to rank well.

In the recent literature many well-known machine learning algorithms have been specifically modified to focus on the problem of ranking rather then minimizing the classification error. For instance, decision trees are known to provide poor probability estimates because their focus is minimizing classification error and size of the tree. With some modifications it is possible to make them produce better probability estimates and subsequently use those estimates to rank better [73, 62]. For neural networks, the back-propagation algorithm [25] can be modified such that the target values of a neural network are set to be the ranks instead of classes [14]. A similar approach can also be applied to the perceptron algorithm [81] to find rank prediction rules that assign each instance a rank which is as close as possible to its original rank [22].

Optimization-based learning algorithms, most notably SVMs [91], are some of the best classification algorithms in existence. SVMs are also known to be good rankers by nature of margin maximization [96]. They can be further modified to rank better by adjusting the constraints to focus on pairwise ordering [10, 79]. Other optimization-based ranking algorithms have been developed to directly optimize metrics related to ranking such as d' [92], which is a measure from signal detection theory [28] indicating how well an algorithm performs on a classification task and is given by

$$d' = \frac{\overline{p} - \overline{n}}{\sigma_n}$$

where $\overline{p}$ is the mean score on all positive points, $\overline{n}$ is the the mean score for the negatives, and σ_n is the standard deviation of negative example scores where the scores that are used are context dependent.

Instance ranking has also been applied to scenarios where user feedback is available on an initial set of rankings [56, 78]. This approach is very popular for ranking search engine results or items (e.g., products) where users can provide feedback. This feedback can be in the form of a user evaluating the initial rankings and reporting mis-ranked items or simply returning a partial ranking where only a subset of items are ranked. A good example to this type of research is Yu's work on SVM selective sampling for ranking [96]. Yu introduces an SVM-based approach where a ranking function is built by using partial feedback provided by the user. At each iteration user feedback is accumulated to a training set and additional constraints are created to enforce the rankings provided by the user. This is repeated until the ranking function is accurate enough, that is, when the user is satisfied by the rankings provided by the algorithm. It is not possible to judge which algorithm performs best since all these investigations target different problems and do not treat common datasets to observe the relative performance.

The literature also includes powerful ensemble methods such as *Rankboost* by Freund et al. [34]. This algorithm is based on the machine learning method called boosting, in particular, Freund and Schapire's *AdaBoost* algorithm [35] and its successor developed by Schapire and Singer [82]. Boosting is an algorithm that produces very accurate rules by combining *weak* rules that are less accurate. Rankboost selectively combines different rankings at each iteration, and checks for incorrectly ranked pairs. If such pairs exist then the algorithm adjusts the weak rank learner to correct those misrankings. Similar to AdaBoost, the RankBoost algorithm focuses on ranking difficult pairs over easier ones making it vulnerable to noisy data.

2.2 Receiver Operating Characteristics (ROC) Curves and Area Under the ROC Curve (AUC)

The most commonly used performance measure to evaluate a classifier's ability to rank instances in binary classification problems is the area under the ROC curve. This section first introduces ROC curves and later defines the area under the ROC curve and how it is related to evaluation of ranking performance.

2.2.1 ROC Curves

ROC curves are widely used for visualization and comparison of performance of binary classifiers. ROC curves were originally used in signal detection theory. They were introduced to the machine learning community by Spackman [86] in 1989, who showed that ROC curves can be used for evaluation and comparison of classification algorithms. ROC analysis is used in radar technology [28], psychology [89], medicine [64], pattern recognition [7], and in machine learning [74]. The main reason that ROC curves became a mainstream performance evaluation tool is the fact that the produced curve is independent of class distribution or underlying misclassification costs [75]. This is a very important property because in most real world problems the true distributions and misclassification costs are unknown, which is problematic for correct performance evaluation of learning algorithms. For example, it is difficult to quantify the cost of errors from incorrectly diagnosing a patient to have cancer versus diagnosing a true cancer patient as healthy.

At this point it is important to understand the basics of ROC curves. We first start by introducing some useful metrics which will be used to construct ROC curves. Given a two-class classifier and an instance there are four possible outcomes. If the instance is positive and if the classifier predicts that it is positive, it called a *true positive*, and if it is predicted negative, it is called a *false negative*. If the instance is negative and the prediction is negative, it is called a *true negative*, and if the prediction is positive, then it is called a *false positive*. Given a classifier and set of data points, a 2×2 matrix, called a *confusion*

True Class

P   N

	True Positives	False Positives
	False Negatives	True Negatives

Predicted Class

P (first row), N (second row)

Figure 2.1: Confusion Matrix

matrix, can be constructed that displays the distribution of outcomes as shown in Figure 2.1. Common performance metrics that can be obtained from a confusion matrix are listed in (2.1),(2.2) and (2.3).

$$fprate = \frac{FP}{allN} \qquad tprate = \frac{TP}{allP} = recall \qquad (2.1)$$

$$precision = \frac{TP}{TP + FP} \qquad accuracy = \frac{TP + TN}{allP + allN} \qquad (2.2)$$

$$F\ measure = \frac{2}{1/precision + 1/recall} \qquad (2.3)$$

ROC curves are two-dimensional graphs that plot true positive rate versus false positive rate. An ROC curve plots the relative trade-off between benefits (true positives) and costs (false positives), by varying the classification threshold, which is usually the probability of membership to a class, distance to a decision surface or simply a score produced by a decision function. Each threshold represents a single classifier and corresponds to a single point on the ROC curve. Some classifiers such as neural networks or naive Bayes naturally

provide these thresholds in the form of probabilities, or SVMs in the form of scores, while other classifiers such as decision trees do not. It is still possible to produce ROC curves for any type of classifier using minor adjustments. Domingos [26] introduced an algorithm that employs bagging [11], a popular ensemble algorithm by Breiman, to generate an ensemble of discrete classifiers, where each classifier produces a vote. The set of votes in this case can be used to generate a score.

In the ROC space shown in Figure 2.2, the upper left corner represents perfect classification while any point on the diagonal line represents random classification. A point in ROC space that lies to the upper left of another point represents a better classifier. A classifier below the diagonal may be said to have useful information, but it is applying the information incorrectly [31]. Classifiers appearing on the left hand-side of an ROC graph, near the x-axis, may be thought of as *conservative*; they make positive classifications only with strong evidence so they make few false positive errors, but they often have low true positive rates as well. Classifiers on the upper right-hand side of an ROC graph may be thought of as *liberal*; they make positive classifications with weak evidence so they classify nearly all positives correctly, but they often have high false positive rates [29].

As pointed out earlier, ROC curves have the nice property of being insensitive to the class distribution and misclassification costs in data. If the proportion of positive points to negatives changes, the ROC curve will not be affected. Large distribution skew is typical in machine learning data, simply because investigating rare data is a common purpose of data mining. However, many algorithms are vulnerable to large skew. In such cases an algorithm may always choose the majority class, and will have substantially small error without learning anything at all. Similarly, when classifying instances, it is typical that the cost of misclassification is different for false-positives and false-negatives. For example, in the spam filtering problem, cost of missing a spam and letting it pass through the filter is not as high as labeling an important email as spam for discarding. However, in general the true misclassification costs are very hard or impossible to obtain and most classifiers assume
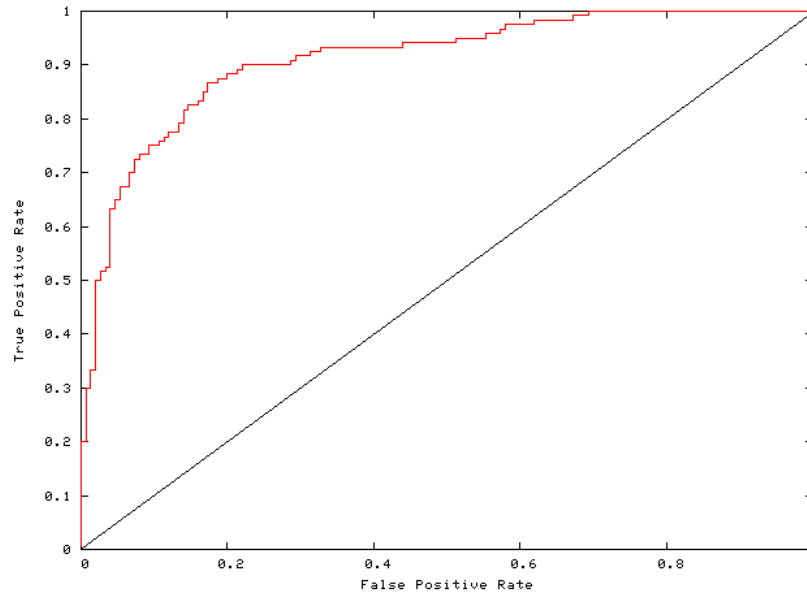
Figure 2.2: A typical ROC curve

that they are equal, hence providing unreliable accuracy values. Therefore, it is typical to use the ROC curve of a classifier, instead of accuracy, to identify the performance especially when the true costs of misclassifications are unknown.

Once ROC curves became mainstream for classifier comparison, in-depth analysis of how to correctly compare classifiers using ROC curves appeared as an important research question. A substantial amount of work in this area was already in place in medical research. Macskassy and Provost provided some of the popular approaches from the medical field to the machine learning community [65]. They performed empirical evaluations on the confidence bands for ROC curves for statistically sound performance comparison. They tested relevant techniques to create confidence intervals such as pooling, vertical averaging and threshold averaging. An in-depth study of the behavior of popular performance metrics such as accuracy, precision and F-measure through the ROC isometrics (contour plots for the metric under investigation) was carried out by Flach [32].

An obvious extension of ROC curves to multi-class problems has been studied by

Fieldsen et al. [30]. The authors extended the two-class ROC analysis to multi-class problem by considering the trade-offs between the misclassification rates from one class into each of the other classes. Rather then considering true and false positive rates, they defined the multi-class ROC surface to be the solution of a multi-objective optimization problem in which these misclassification rates are simultaneously optimized.

An intuitive research question is how to find ways to push the ROC curve further toward the upper left corner of the ROC graph, which in turn would improve the overall performance of classifier. There are several methods for modifying classifiers so that they will get better performance on specific regions of the ROC curve [68]. Mozer et al. proposed four different methods to specifically optimize the performance in the low FP-rate region (lower left) of the ROC curve. The authors used two different weighting scheme for data points to manipulate the the effect of each point when training. They also proposed an optimization based approach and a genetic-algorithm-based approach to obtain better ROC curves.

2.2.2 Area Under the ROC Curve (AUC)

Area under the ROC Curve (AUC) is a single scalar value for classifier comparison [7, 45]. Statistically speaking, the AUC of a classifier is the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. Since AUC is a probability, its value varies between 0 and 1, where 1 represents all positives being ranked higher than all negatives. Larger AUC values indicate better classifier performance across the full range of possible thresholds. Even though it is possible that a classifier with high AUC can be outperformed by a lower AUC classifier at some region of the ROC space, as shown in Figure 2.3, in general the high AUC classifier is better on average when the underlying distribution is not known.

The machine learning community has explored the relationships between AUC and accuracy [18] as well as the weaknesses of accuracy compared to AUC. The common use
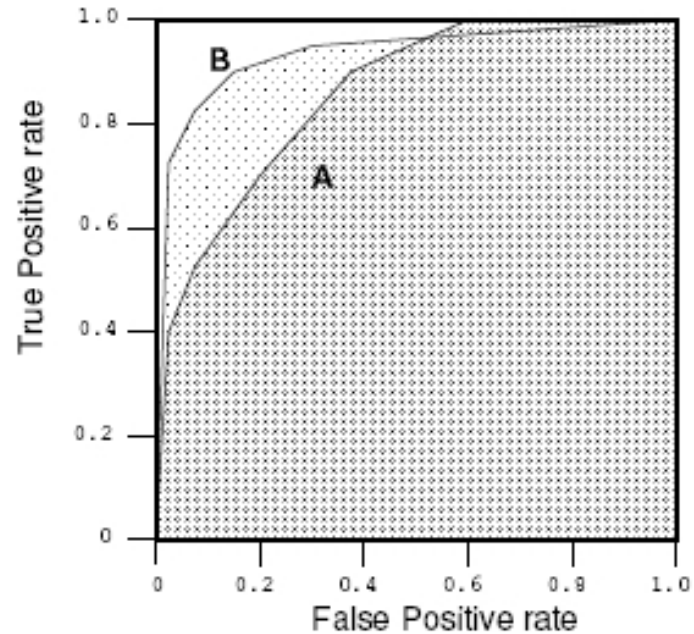
Figure 2.3: Comparison of Area Under the ROC Curve

of accuracy as a performance measure is critically investigated by Provost et al., providing an in depth study using ROC analysis and standard benchmark datasets [74]. The research showed that in order to prove a classifier's dominance over another without knowing the target misclassification cost, the algorithm should perform better for every possible threshold on the ROC curve. Without knowing true misclassification costs, it is not correct to conclude that one classifier is better then the other one.

AUC is found to be statistically consistent and a more discriminating value than accuracy [61]. Ling and Huang's work is significant in that it presents formal proofs to rigourously establish that AUC is a better measure for comparison of learning algorithms. In a follow-up work [51], they compared some of the well-known classifiers such as naive Bayes, decision trees and SVMs by both accuracy and AUC on some popular machine learning problems from the UCI data repository. The authors established a common ground for evaluating the two metrics, accuracy and AUC, in terms of consistency and discriminancy defined as:

Consistency: For two measures f, g on domain Ψ, f, g are (strictly) consistent if there exists no $a, b \in \Psi$ such that $f(a) > f(b)$ and $g(a) < g(b)$.

Discriminancy: For two measures f, g on domain Ψ f is (strictly) more discriminating then g if there exists $a, b \in \Psi$ such that $f(a) > f(b)$ and $g(a) = g(b)$ and there exists no $a, b \in \Psi$ such that $g(a) > g(b)$ and $f(a) = f(b)$.

To illustrate with an example the authors used numerical grades and letter grades that evaluate student performance. A numerical mark gives 100, 99, 98, ..., 1, or 0 to students, while a letter mark gives A, B, C, D, or F to students. Obviously, grades are ordered as A>B>C>D>F and this makes numerical marks consistent with letter marks (and vice versa). In addition, numerical marks are more discriminating than letter marks, since two students who receive 91 and 93 respectively receive different numerical marks but the same letter mark, but it is not possible to have students with different letter marks (such as A and B) but with the same numerical mark.

In a similar study Bradley [7] evaluated six machine learning algorithms (C4.5, multiscale classifier, perceptron, multi-layer perceptron, k-nearest neighbors, and a quadratic discriminant function) on six medical diagnostics data sets. He compared and discussed the use of AUC to the more conventional overall accuracy and found that AUC exhibits a number of desirable properties when compared to overall accuracy, such as; increased sensitivity in Analysis of Variance (ANOVA) tests; a standard error that decreased as both AUC and the number of test samples increased; decision threshold independence; and invariance to a priori class probabilities. The paper concludes with the recommendation that AUC be used in preference to overall accuracy for "single number" evaluation of classification algorithms.

Hand and Till [44] provide a very straightforward way of estimating the AUC given in (2.4). They also draw important conclusions on the relations between AUC, Wilcoxon-Mann-Whitney (WMW)statistic and Gini coefficient. They compute AUC as

$$AUC = 1 - \frac{S_+ - p(p+1)/2}{pn},$$
(2.4)

where p and n represent the number of positive and negative data points respectively, and S_+ is the sum of the ranks of the positive class (after the points are ranked by a ranking function). According to the authors this estimation is also immune to the errors that may be introduced by smoothing procedures. The fractional part of the equation on the right hand side is equivalent to the test statistic used in the WMW two-sample test. The next section provides an in-depth explanation of the relationship between the WMW statistic and AUC.

Before moving to the next section, we look at some examples from [61] that illustrate a comparison of accuracy and AUC in terms of performance evaluation. Assume a dataset with 10 data points, 5 positive and 5 negative examples. Let two classifiers produce probability estimates for those points, and order the data points using these estimates (descending from left to right in the tables provided). Also assume that both classifiers label 5 data points as positive and 5 as negative.

Table 2.1 illustrates a case where two classification algorithms have the same error rate (20%). Although the accuracies are equal it is not hard to imagine picking classifier 1 over 2 since it orders more positives correctly above the negatives. Using equation (2.4) we can calculate the AUC of classifier 1 to be $\frac{24}{25}$ where classifier 2 has an AUC of $\frac{16}{25}$.

Table 2.1: Example 1: Two classifiers with equal accuracy and different AUC

CLASSIFIER 1	+ + + + −	+ − − − −
CLASSIFIER 2	− + + + +	− − − − +

It is not always true that the higher-AUC classifier is better. Table 2.2 shows the case where classifier 4 has a lower error rate (20%) than classifier 3 (40%). On the other hand classifier 3 has a better AUC performance with $\frac{21}{25}$ while classifier 4 has $\frac{16}{25}$. Therefore a good AUC performance does not guarantee a lower error rate for classifiers which is also consistent with the findings in [74].

Table 2.2: Example 3: One classifier
with higher AUC but lower accuracy

CLASSIFIER 3	+ + + − −	+ + − − −
CLASSIFIER 4	− + + + +	− − − − +

The opposite of the last paragraph is also true. In other words a lower error rate does not guarantee better AUC performance. In the example given in Table 2.3 classifier 5 has a lower error rate (40%) then classifier 6 (60%) but both have the same AUC ($\frac{15}{25}$).

Table 2.3: Example 3: Two classifiers
with same AUC and different accuracy

CLASSIFIER 5	+ − − + +	− + + − −
CLASSIFIER 6	+ − + − −	+ + + − −

2.3 Relationship of AUC to the Wilcoxon-Mann-Whitney Statistic

AUC has been introduced as an evaluation criterion for ranking problems where the data has binary outputs. In these problems, data points belong to one of two classes, usually given as positive and negative, and the goal is to learn a ranking such that each positive data point is ranked higher then each negative data point.

Consider the classification task with binary outputs with p positive data points and n negative data points. We assume that a classifier produces a numeric output that can strictly order all the data points using the indicator function, $I(\cdot)$. Let c be a classifier with classification function $f(\cdot)$, where $f(x_i)$ is the output of positive examples and $f(x_j)$ is the output of negative examples. Then the value of the Wilcoxon-Mann-Whitney statistic [45] is given by

$$W = \frac{\sum_{i=0}^{p-1} \sum_{j=0}^{n-1} I(x_i, x_j)}{pn} \tag{2.5}$$

$$I(x_i, x_j) = \begin{cases} 1 & \text{if } f(x_i) > f(x_j) \\ 0 & \text{otherwise} \end{cases},$$

which is also the value of the AUC for c.

The proof is based on the observation that the AUC is the probability $P(X^+ > X^-)$ where X^+ is the random variable corresponding to the distribution of outputs for positive points and X^- is the one corresponding to the negatives [43]. The WMW statistic is the expression of this probability in the discrete case [45].

2.4 Optimization of the AUC by WMW Statistic

At this point it seems intuitive that an algorithm maximizing the WMW statistic will also maximize the AUC and hence the ranking performance. An algorithm trained to maximize the WMW statistic will increase the number of correctly-ordered positive-negative pairs in a given dataset, as well as correctly ordering positives and negatives among themselves. Also, any improvement of this metric will result in a larger area under the ROC curve. The WMW statistic itself is not a continuous function, making it a combinatorial problem which is difficult to solve directly. Also, if the positive and negatives are separable the problem is ill-posed since there exist infinitely many optimal solutions. Therefore the formulation needs to be stabilized by regularization, such as error minimization. Several papers suggested an approximation approach to the WMW statistic. Both Yan et al. [95] and Herschtal and Raskutti [48] used a continuous function as an approximation to the WMW statistic. By doing so they were able to use gradient methods to solve the optimization problem.

Herschtal and Raskutti's formulation (2.6) approximates the WMW statistic by a differentiable sigmoid function to calculate a *rank statistic*, $R(\beta)$,

$$R(\beta) = \frac{1}{pn} \sum_{i,j}^{p,n} s(\beta(x_i - x_j)) \quad \forall x_i \in X^+, x_j \in X^-, \tag{2.6}$$

where *p* is the number of positives and *n* is the number of negatives, β is the vector of coefficients of the predictor variables and $s(x)$ is a sigmoid function given as,

$$s(x) = \frac{1}{1 + e^{-x}}.$$

For large $||\beta||$ the sigmoid rank statistic is a good approximation to the AUC.

Yan et al.'s formulation approximates the indicator function $I(x_i, x_j)$ in (2.5) by

$$R(x_i, x_j) = \begin{cases} (f(x_i) - f(x_j) - \gamma)^2 & \text{if } f(x_i) - f(x_j) \le \gamma \\ 0 & \text{otherwise,} \end{cases} \tag{2.7}$$

where γ is a margin with $0 \le \gamma \le 1$ to approximate $I(-x_i, -x_j)$ and f is a numeric output of the algorithm. With this formulation it is possible to use a gradient-based method, such as a limited memory BFGS method, to train a classifier by minimizing the objective function given as

$$\sum_{i=0}^{p-1} \sum_{j=0}^{n-1} R(x_i, x_j),$$

where *i* and *j* are the same indices used in (2.5) with *p* positive and *n* negative data points.

2.5 AUC Optimization by SVMs

2.5.1 Support Vector Machines

SVMs are among the best classification algorithms and they also rank very well. This is mostly due to the property of margin maximization in the optimization problem. To better understand a rank optimizing SVM, we need to examine the SVM formulation.

A standard SVM learner solves the quadratic optimization problem given in (2.8) to obtain an optimal hyperplane ($wx + b = 0$) that minimizes the classification error [91]:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y_i((wx_i) + b) \ge 1, \forall i = 1, ..., l, \end{aligned} \tag{2.8}$$

where w is the vector of attribute weights and y_i is the class label of the data point x_i. The constrained optimization problem given in (2.8) can be dealt with by introducing Lagrange multipliers $\alpha_i \geq 0$ to obtain

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i(y_i((x_iw) + b) - 1). \tag{2.9}$$

The Lagrangian L is minimized with respect to the primal variables w and b and maximized with respect to the dual variables α_i. Intuitively, if a constraint given in (2.8) is violated, then $y_i((wx_i) + b) < 1$, in which case L can be increased by increasing the corresponding α_i. At the same time, w and b will have to change such that L decreases. To prevent $-\alpha_i(y_i((wx_i) + b) - 1)$ from becoming arbitrarily large, the change in w and b will ensure that, provided that the problem is separable, the constraint will eventually be satisfied.

For non-separable problems, which are more typical in the real world, a soft margin hyperplane can be used by introducing slack variables:

$$\min_{w,\xi} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{l}\xi_i$$

$$\text{s.t.} \quad \begin{aligned} y_i((w.x_i) + b) &\geq 1 - \xi_i, \forall i = 1, ..., l \\ \xi_i &\geq 0, \end{aligned} \tag{2.10}$$

where $C > 0$ is the trade-off parameter in the objective function and ξ_i are the slack variables for each constraint [19]. Usually the dual form of (2.10) is solved to obtain an optimal hyperplane. The dual is given as:

$$\min_{\alpha} \quad -\sum_{i=1}^{l}\alpha_i + \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j x_i x_j$$

$$\text{s.t.} \quad \begin{aligned} C \geq \alpha_i &\geq 0, \quad i = 1, ..., l \\ \sum_{i=1}^{l}\alpha_i y_i &= 0. \end{aligned} \tag{2.11}$$

Note that in (2.11) data points appear as dot products, $x_i \cdot x_j$. Although we know that in the current feature space the problem is linearly non-separable, it is very likely that in a much higher dimension the problem may become linearly separable by a hyperplane. A mapping function, $\Phi : \Re^n \to \Re^m$ where $m > n$, can be used for such a transformation to a higher

dimensional space. However, the transformation operation Φ is computationally expensive. Instead a kernel function, $k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$, can be used for mapping into the new feature space. This yields a non-linear decision boundary in the original space. By the use of the kernel function $k(x_i, x_j)$, it is possible to compute the separating hyperplane without explicitly carrying out the mapping calculation into the high-dimensional feature space. This is also known as the *kernel trick* [83].

The quadratic optimization function given in equation (2.11) produces an optimal set of α's, and a set of weights for an optimal hyperplane can be recovered by

$$w = \sum_{i=1}^{l} \alpha_i y_i x_i.$$

Also, using the optimal set of α's, a decision function can be obtained to classify a new data point x_t:

$$f(x) = sgn\left(\sum_{i=0}^{l} y_i \alpha_i k(x_t, x_i) + b\right). \tag{2.12}$$

Here, we would also like to define 1-norm SVM in the primal form given as (2.13) since some of our later formulations will refer to 1-norm SVMs. The formulation is equivalent to a 2-norm SVM except the 1-norm of w is minimized instead of the 2-norm.

$$
\begin{aligned}
\min_{w,\xi} \quad & ||w||_1 + C \sum_{i=1}^{l} \xi_i \\
\text{s.t.} \quad & y_i((w.x_i) + b) \geq 1 - \xi_i, \forall i = 1, ..., l \\
& \xi_i \geq 0,
\end{aligned}
\tag{2.13}
$$

Literature indicates there is no difference in terms of algorithm performance between a 1-norm and a 2-norm SVM formulations [98, 9]. We adopted 1-norm to be able to formulate our problem as an linear program. Linear programs can be solved by simplex method [23]. Although simplex method is known to be exponential in algorithmic complexity, generally in practice it solves much faster then QP algorithms such as interior point methods.

2.5.2   Rank-optimizing SVMs

While SVMs do a good job among classifiers of ranking instances, recent research has focused on further improving ranking performance of SVMs. An effort by Rakotoma-monjy to investigate AUC maximizing SVMs led to the formulation given in (2.14) [79]. The dual form of his formulation included a form of kernel structure that optimize the AUC performance and is given as:

$$
\min_{\alpha} \quad -\sum_{i,j=1}^{p,n} \alpha_{i,j} + \frac{1}{2} \sum_{i,j=1}^{p,n} \sum_{u,v=1}^{p,n} \alpha_{i,j}\alpha_{u,v} k_{ij,uv}
$$

$$
\text{s.t.} \quad C \geq \alpha_{i,j} \geq 0,
$$

(2.14)

where $x_i, x_u \in X^+$ and $x_j, x_v \in X^-$. $k_{ij,uv}$ is defined as $k[(x_i - x_u) - k(x_i - x_v) - k(x_j - x_u) + k(x_j - x_v)]$, where $k(\cdot)$ is a kernel function that maps the original feature space to a higher-dimensional feature space. Typical kernel functions that can be used here include radial basis function (RBF) kernels and polynomial kernels, which are explained in the next chapter. Solving this quadratic optimization formulation yields the following decision function:

$$
f(x) = \sum_{i,j=1}^{p,n} \alpha_{i,j}(k(x_i - x_j, x) + b).
$$

(2.15)

The optimal α's represent a weight for each positive-negative pair of points, therefore the number of α's is equal to the number of positives times the number of negatives. The number of variables grows quadratically with the increasing number of training points.

Rakotomamonjy's work led to more investigations in the area. Brefeld et al. worked on the same formulation for both 2-norm and 1-norm SVMs which led to an improved ranking performance [10]. The time complexity of AUC maximizing SVMs continued to be a problem with both number of variables and constraints being quadratic with respect to the number of data points. We will investigate this further in the following chapters.

Joachims et al. also introduced a very similar formulation to the two given above in the context of web search with implicit feedback. The authors present their SVM ranker

formulation as a quadratic programming problem given as:

$$\min_{w,\xi_{i,j}} \quad \frac{1}{2} w \cdot w + C \sum_{i,j=1}^{p,n} \xi_{i,j}$$

$$\text{s.t.} \quad w \cdot \Phi(d_i, q) \geq w \cdot \Phi(d_j, q) + 1 - \xi_{i,j}$$

$$\xi_{i,j} \geq 0,$$

(2.16)

where d_i, d_j represents documents from a set of documents D, q represents a user query and $\xi_{i,j}$ represents the error of ranking a more relevant document below a less relevant one. The mapping function Φ is similar to $k(\cdot)$ from the previous algorithms. The logic behind this formulation is that each of the constraints represents the similarity between two documents evaluated in the context of a single query. Using this formulation a preset number of documents can be evaluated and ranked based on a query. In such a setup the queries (typically constructed by the user) are the key for constructing an implicit ranking of documents.

2.6 Other AUC Optimization Methods

As can be seen from the recent line of work, AUC optimization has become very popular in the machine learning literature. Several other ways to maximize the AUC are summarized in this section.

Bostrom investigated the use of incremental reduced error pruning for maximizing AUC instead of accuracy. This is a technique that has been extensively used for efficient separate-and-conquer rule learning [37, 16, 33]. While a commonly employed pruning criterion, based on precision, has been shown to maximize AUC [38], the author showed that a commonly-used exclusion criterion, based on accuracy, may include rules that result in concave ROC curves, as well as exclude rules that result in convex ROC curves. The empirical investigation over a span of 34 datasets showed an improved AUC compared to any other accuracy-based criteria. Another method introduced by Sebag et al. tackled AUC optimization for learning linear hypotheses in medical data mining, using evolutionary computation

techniques [84, 41]. The stochastic nature of the setup provides scalability for very large datasets. This approach also provides the advantage of exploitation of a subset of solutions generated in the evolutionary process (sensitivity analysis) without any extra computation cost.

Ordinal regression is another tool that can be utilized for rank optimization problems in binary classification. It is similar to regression, where data points are mapped to scalar values which help to classify (or rank) pairs of points. A large margin algorithm based on ordinal regression modeled by Herbrich et al. [47] ranks points by intervals on the real line and then finds a scoring function that maps each point to its ordinal value. The resulting task is also known as *preference learning* [46]. Learning of preference reduces to a standard classification problem if pairs of objects are considered. For each ordinal regression problem there is a corresponding preference learning problem on pairs of objects. In fact, previous algorithms discussed under section 2.5.2 are all special cases of the ordinal regression method.

In this chapter we thoroughly surveyed the literature starting from problem of ranking. We then moved to ranking in problems with binary outputs. We defined the metric, AUC, that evaluates the ranking performance for problems with binary outputs. The equivalency of AUC and the WMW statistic was also introduced. We briefly summarized AUC/WMW-optimizing algorithms, including rank-optimized support vector machines. We present AUC-maximizing SVM formulations and point out the major drawback of such algorithms which is the time and space complexity arising from the quadratic expansion of the required constraints for optimization. In the next chapter we introduce our version of a ranker and evaluate its performance against similar algorithms.

CHAPTER III
THE LINEAR PROGRAMMING FORMULATION

3.1 Motivation

The area under the ROC curve (AUC) is a commonly-used measure for evaluating the quality of the ranking performance of an algorithm for data with binary output. It is a good general-purpose evaluation method because it is invariant to cost and class distribution. Thus, it makes sense to directly optimize some approximation of the AUC (which is equivalent to the Wilcoxon-Mann-Whitney (WMW) statistic [93]) by, for example, attempting to correctly rank every positive/negative pair of points. Our formulation given in the next section constructs a linear-programming-based 1-norm SVM and uses a rank-optimizing kernel function that reduces the size of the resulting problem previously introduced in the literature [10, 79] while avoiding their drawbacks.

In this chapter, we first introduce a linear-programming-based ranking algorithm. Then we look at its similarities and differences to SVM. Later we compare its AUC performance against regular 2-norm SVM and a recently published variation of an SVM formulation optimized for AUC performance using popular benchmark datasets. We also evaluate the classification performance of these algorithms.

3.2 The Algorithm: Linear Programming Ranker (LPR)

Let (x, y) be an instance in the training set X, where x is the data vector and $y \in \{-1, 1\}$ is the class label for that instance. We refer to the set of positive points in the data set as X^+ and negatives as X^-, such that $X^+ \cup X^- = X$. To maximize the WMW statistic, we would like to have all positive points ranked higher than all negative points, such as:

$$f(x_i) > f(x_j) \quad \forall x_i \in X^+, x_j \in X^-,$$

where f is some scoring function which will be defined in the next paragraph. A perfect separation of classes is impossible for most real world datasets. Therefore a linear program

is constructed to minimize some error term corresponding to the number of incorrect orderings. We construct a linear program with soft constraints penalizing negatives that are ranked above positives. This requires p (number of positive points) times n (number of negative points) constraints in total.

The problem of ordering instances is hard, especially when the number of items to order goes up. There are $k!$ possible ways to order k instances, which generates an extremely large search space with a large k. Obviously, we would prefer not to wander in the search space until we obtain a good solution (or better yet find the global optimum if we are lucky). We use the set of generated constraints to guide us by systematically reducing the search space. However, for many real world problems, use of hard constraints (constraints that must be satisfied) may make the optimization problem impossible to solve since all constraints cannot be satisfied at the same time. Soft constraints (constraints that can be violated with a cost of slack) can be used to avoid this problem. Our formulation avoids combinatorial complexity by minimizing the error, z, which is the difference in scores of incorrectly ordered pairs.

We select a scoring function, $f(x)$, such that it assigns a real-valued score to each data point while making the optimization problem continuous. This leads to the following set of constraints:

$$f(x_i) - f(x_j) \geq \varepsilon - z_{ij} \quad \forall x_i \in X^+, x_j \in X^-, \tag{3.1}$$

where ε is a small quantity usually set to 1 and added to the right hand side to avoid strict inequalities in the constraints.

The scoring function we use here is similar to an SVM decision function (Equation 2.12) without the intercept, b. The intercept is not necessary since it would add a constant amount for all the points in the scoring function and would have no influence on either the relative scores or the overall ranking. The scoring function is defined as

$$f(x) = \sum_{l \in X} y_l \alpha_l k(x, x_l) \tag{3.2}$$

where α_l represents a weight for each point in the training set. The function $k(\cdot)$ is a kernel function. With this formulation we can obtain a score for any x using the training points x_k's and the trained set of weights, α's.

In our objective function we would like to minimize the errors, z, along with the magnitudes of the coefficients, α. We minimize α to prevent them from getting arbitrarily large as well as to maximize the separation margin between the data points. The proposed objective function and linear program can be obtained as follows by substituting the scoring function (3.2) in (3.1) and rearranging the left side of the inequality:

$$\min_{\alpha,z} \quad \sum_{l \in X} \alpha_l + C \sum_{i \in X^+, j \in X^-} w_{i,j} z_{i,j}$$

$$\text{s.t.} \quad \sum_{l \in X} y_l \alpha_l [k(x_i, x_l) - k(x_j, x_l)] \geq 1 - z_{i,j} \quad \forall x_i \in X^+, x_j \in X^- \tag{3.3}$$

$$\alpha, z \geq 0$$

where C is the tradeoff parameter between the two parts of the objective, $w_{i,j}$ is the weight on each pairwise error (usually set to a constant value in our experiments), and $z_{i,j}$ represents an error term for each positive/negative pair. In this formulation we set C as a single scalar value that increases or decreases the importance of errors relative to the sum of the α's. The parameter w lets the user to explicitly manipulate the weight on each error term. In our experiments with the benchmark datasets the w's are set to 1 making all the pairwise errors equal. However, there may be certain cases where a domain expert knows the relative importance of errors. In such cases those individual error weights can be beneficial to improve the performance.

In our experiments we use RBF kernels of the form

$$k(x, x_l) = e^{-\gamma \|x - x_l\|^2}, \tag{3.4}$$

where γ is the RBF parameter that controls the smoothness of the function. For comparison versus 2-norm SVM we also used polynomial kernel in the form

$$k(x, x_l) = (x x_l + 1)^p, \tag{3.5}$$

where p is the power term that adjusts the complexity. RBF will be our choice of kernel

function throughout the dissertation. It yields better performance compared to polynomial kernels, which we show later in this chapter, as well as the literature supporting the choice of RBF kernels for AUC maximization [15].

The output of the linear program (3.3) gives the optimal set of weights, α^*. The instances with non-zero weights, which we call $ranking\ vectors$, represent the unique points that influence the ranking. By using a kernel function, we are mapping the input space to a feature space defined by the kernel function. This makes it possible to rank the data with reduced error, in the mapped space. Once the optimal weights α^* are found for each point in the training set we can use the scoring function

$$f(x_{test}) = \sum_{l \in X} y_l \alpha_l^* k(x_{test}, x_l) \tag{3.6}$$

to obtain scores for each test point. Finally we can rank the test points by the scores obtained.

At this point it is important to understand the differences between our approach and similar formulations. Radlinski and Joachims [78] presented a formulation for rank optimization where each constraint represents a pairwise ordering of points. While their kernel structure is similar to ours, each of their constraints represents the similarity between two documents evaluated in the context of a single query. In such a setup the queries are the key for constructing an implicit ranking of documents. Our formulation, on the other hand, sums the similarities across a training set of labeled pairs of training points.

The kernel structure we use in the constraints to enforce the ranking of positives over negatives is $[k(x_i, x_l) - k(x_j, x_l)]$ as opposed to Rakotomamonji's or Brefeld et al.'s $[k(x_i, x_u) - k(x_i, x_v) - k(x_j, x_u) + k(x_j, x_v)]$, where i, u are indices from the set of positive points and j, v are from the set of negatives. Our formulation reduces the complexity of the problem by reducing the number of variables from (pn) to $(p + n)$, while the number of constraints in each of the algorithms remain as (pn). The intuition behind this is, instead of comparing the difference of positive-negative pairs to each other in the feature space, we simply compare each positive and negative pair of points to a fixed point in the

dataset and measure the difference. An RBF kernel evaluating two points produces a score between 0 and 1. The more the similarity between two points the closer the score to 1. Our approach compares each data point, x_l, to a positive/negative pair, x_i and x_j, and the difference $k(x_i, x_l) - k(x_j, x_l)$ produces a positive value if the data point(x_l) is more similar to the positive point(x_i) in the pair or negative value if the point is more similar to the negative(x_j). The partial scores from each evaluated pair are summed to generate an overall score for each data point which can then be used for ranking. This process captures the idea presented in Rakotomamonji's and Brefeld et al.'s formulations while reducing the complexity of the problem.

When an RBF kernel is used, our approach is similar to distance-weighted k-nearest neighbor (kNN) algorithm, where k is equal to the number of ranking vectors. Consider the scoring function of distance-weighted kNN where only the ranking vectors are used as training points, $f(x_{test}) = \sum_{r \in X} y_r \times D(x_{test}, x_r)^{-1}$, where x_r's are ranking vectors. If the inverse of RBF is used as the distance function such that $D(x_i, x_j) = e^{\gamma \|x_i - x_j\|^2}$, the above scoring function differs from (3.6) only by the coefficients α_r, which are obtained by the optimization procedure in our approach.

3.3 Ranking Vectors vs. Support Vectors

Although similar in formulation, there is still a difference in nature of ranking and support vectors. To visualize the differences between ranking vectors and support vectors, we created a simple artificial dataset that represents an XOR problem. XOR is one of the basic forms of linearly non-separable problems. We can utilize a kernel function to make this problem separable in a higher-dimensional space. We used RBF as the kernel of our choice for both LPR and SVM. After training both algorithms we obtain the following Figure 3.1, which plots both ranking and support vectors in the original feature space, as well as the decision boundary of the SVM .
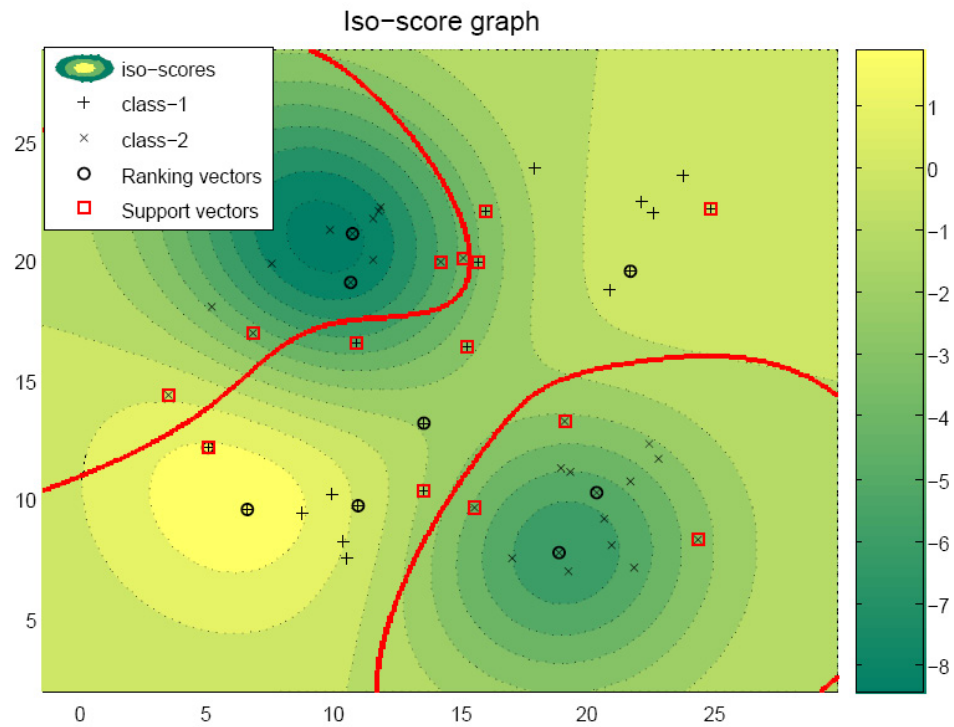
Figure 3.1: Ranking vectors vs. support vectors

As we can see from Figure 3.1 ranking and support vectors appear in different locations in the feature space. Ranking of points can be evaluated based on iso-score lines from the figure. The feature space is divided into regions of scores represented by iso-score lines. Therefore the algorithm tries to order points such that points from the positive class appear in the high-score regions and points from the negative class in the low-score regions. As expected, the support vectors appear close to the support vector decision boundary. Ranking vectors on the other hand appear around either regions that are representative of a class or at locations where it can bend the iso-score lines just a bit such that locally more pairs are ranked correctly. The number of ranking vectors are proportional with the difficulty of the problem such that for harder problems we observe more ranking vectors. Also, as the RBF parameter γ increases, which increases the complexity of the model, the number of ranking vectors increase similar to support vector machines.

3.4 Comparison of AUC performance vs. 2-norm SVM

In our experiments we used 13 data sets from the UCI repository [6]. Multi-class data sets are converted to two-class problems by using one class vs. others scheme. Details of these conversions are given in Table 3.1. As a first step we compared the performance of LPR with a regular 2-norm SVM using RBF and polynomial kernels [4].

Table 3.1: Overview of the datasets and modification details

Datasets	# of pts	# attrib.	% rare class	# of class - Comments
BOSTON	506	14	9	(MEDV<35)=1,REST=0
ECOLI	336	8	15	"PP"=1,REST=-1
GLASS	214	10	14	"7"=1,REST=-1
HEART	270	14	44	2
SONAR	208	61	47	2
SPECTF	351	45	28	2
CANCER(WPBC)	194	34	24	2
IONOSPHERE	351	35	36	2
HABERMAN	306	4	26	2
LIVER(BUPA)	345	7	42	2
CANCER(WBC)	699	10	34	2
CANCER(WDBC)	569	32	37	2
SEGMENT	210	20	14	"BRICK"=1,REST=-1

We implemented LPR in Matlab [54] and used CPLEX [53] for optimization in the Matlab environment. For performance comparisons, we used SVM implementation by Sequential Minimal Optimization (SMO) algorithm [72] available in WEKA [94]. We constructed a grid search to find the best RBF and polynomial kernel parameter settings for both LPR and SVM using $\gamma = \{0.01, 0.1, 1\}$, $p = \{1, 2, 3\}$ and $C = \{1, 10, 100\}$. For

all the datasets, we averaged 5×10-fold cross-validation results. In each cross-validation the data is split into 80% for training, 10% for validating (parameter tuning) and 10% for testing. For each fold, the algorithm trains with training data using all possible parameter combinations. Each trained model is evaluated on the validation set to find the optimal parameter for that fold. Finally, we use the model with the optimal parameters on the test set. This procedure is repeated 10 times in each cross-validation run. We make sure that each point is tested only once in the cross-validation procedure.

Here, we would like to note that for a number of runs in our experiments in Chapters 3 and 4, CPLEX ran out of memory solving the WBC and WDBC datasets. We include the results from the ones that were solvable in performance comparisons.

Table 3.2: AUC results using RBF kernel from 5×10-fold cross-validation

Dataset	LPR	SVM
BOSTON	0.9654(0.0100)	0.9484(0.0082)
ECOLI	0.9598(0.0080)	0.9494(0.0022)
GLASS	0.9708(0.0064)	0.9511(0.0064)
HEART	0.9072(0.0044)	0.8566(0.0039)
SONAR	0.9244(0.0055)	0.9027(0.0089)
SPECTF	0.9156(0.0060)	0.8926(0.0042)
WPBC	0.7314(0.0203)	0.7160(0.0159)
IONOSPHERE	0.9735(0.0068)	0.9529(0.0018)
HABERMAN	0.6564(0.0212)	0.6974(0.0064)
LIVER	0.7198(0.0190)	0.7239(0.0058)
SEGMENT	0.9878(0.0010)	0.9842(0.0026)
WDBC	0.9920(0.0012)	0.9868(0.0012)
(W,L,T)	(10,1,1)	

Table 3.2 shows that our ranking algorithm performed better in general then 2-norm SVM in AUC performance and when RBF kernel is used. The last row of the table shows statistical comparisons in the form of (win, loss, tie) where the statistical significance was set at the 0.05 level. As a side note, for all our significance tests throughout this dissertation we use paired t-test at the 0.05 level. LPR, optimized for ranking, was significantly better in AUC performance for ten datasets.

Table 3.3: Accuracy results using RBF kernel from 5×10-fold cross validation

Dataset	LPR	SVM
BOSTON	0.9510(0.0090)	0.9440(0.0071)
ECOLI	0.9458(0.0054)	0.9476(0.0033)
GLASS	0.9701(0.0070)	0.9645(0.0053)
HEART	0.7919(0.0198)	0.8022(0.0062)
SONAR	0.6890(0.0334)	0.8706(0.0098)
SPECTF	0.9038(0.0060)	0.8436(0.0122)
WPBC	0.7705(0.0139)	0.6908(0.0262)
IONOSPHERE	0.9359(0.0077)	0.9380(0.0069)
HABERMAN	0.7219(0.0107)	0.7403(0.0104)
LIVER	0.6916(0.0156)	0.7105(0.0062)
SEGMENT	0.9743(0.0064)	0.9743(0.0043)
(W,L,T)	(2,1,8)	

To investigate the effects of AUC optimization on classification performance we also compiled accuracy results from both algorithms which are provided in Table 3.3. We believe that the nature of AUC optimization that ranks positive points above negatives could have a favorable effect on accuracy since the process also enforces separation of two classes. To obtain accuracy for our algorithm we find the threshold (score) that gives

the best accuracy value on the validation set. We used the same threshold on the test set to classify the test points.

When accuracies were compared, LPR was in general as good as 2-norm SVM, tying in 8 datasets. Although the classification performance is similar, SVMs have the advantage of being significantly faster than LPR. Still it is interesting to observe a rank optimized algorithm performing as well as a state-of-the-art classification algorithm.

Table 3.4: AUC results using polynomial kernel from 5×10-fold cross validation

Datasets	LPR	SVM
BOSTON	0.9571(0.0180)	0.9420(0.0128)
ECOLI	0.9485(0.0065)	0.9409(0.0033)
GLASS	0.9662(0.0118)	0.9506(0.0090)
HEART	0.8564(0.0113)	0.8164(0.0012)
SONAR	0.8851(0.0172)	0.8739(0.0109)
SPECTF	0.8921(0.0123)	0.8829(0.0079)
WPBC	0.7534(0.0171)	0.7189(0.0227)
IONOSPHERE	0.8926(0.0210)	0.8758(0.0104)
HABERMAN	0.6823(0.0132)	0.6935(0.0069)
LIVER	0.7231(0.0222)	0.7286(0.0047)
SEGEMENT	0.9770(0.0119)	0.9873(0.0025)
(W,L,T)	(4,0,7)	

We have experimented with using polynomial kernels with LPR (p = 1,2,3). We compared the results to 2-norm SVM with polynomial kernel using the same datasets. The comparison results using polynomial kernel are given in Table 3.4. We observed that LPR was significantly better in 4 out of 11 datasets and tied with SVM on the others. We conclude that any of the two kernel functions we used shows a better AUC performance against

SVM, however the improvement achieved using polynomial kernel is not as impressive as RBF.

We also compared the accuracy performance results when polynomial kernels are used. The results presented in table 3.5 were similar to the RBF performance. LPR performs as well as SVM in classification performance tying in 10 datasets and and significantly winning only in one. Again the drawback of LPR is that it is significantly slower then SVM.

Table 3.5: Accuracy results using polynomial kernel from 5×10-fold cross validation

Datasets	LPR	SVM
BOSTON	0.9483(0.0068)	0.9431(0.0080)
ECOLI	0.9285(0.0076)	0.9254(0.0060)
GLASS	0.9645(0.0021)	0.9626(0.0089)
HEART	0.7852(0.0148)	0.8022(0.0173)
SONAR	0.7952(0.0223)	0.8153(0.0133)
SPECTF	0.8741(0.0184)	0.8449(0.0081)
WPBC	0.7620(0.0172)	0.7421(0.0209)
IONOSPHERE	0.8860(0.0107)	0.8860(0.0077)
HABERMAN	0.7311(0.0188)	0.7320(0.0069)
LIVER	0.7032(0.0178)	0.7246(0.0132)
SEGMENT	0.9838(0.0043)	0.9743(0.0072)
(W,L,T)	(1,0,10)	

3.5   LPR vs. Support Vector Ranker (SVR)

As a next step we compared the performance of LPR against Brefeld's rank optimized support vector machine (SVR), which is a recently published state-of-the-art AUC optimizing algorithm. We implemented the SVR as a quadratic programming (QP) problem, which is formulated as an AUC maximizing 1-norm SVM as given in [10]. The comparison setup is implemented in Matlab and the optimization problem is solved using CPLEX. The major problem with SVR is that its space complexity is $\mathcal{O}(n^4)$. Because the size of the constraint matrix can get insolvably large based on the size of the data, it was not possible to use 10-fold cross-validation that uses 80% of the data to train. Therefore we used a sampling methodology to reduce the training data size such that we sampled as many points as possible to train the SVR in reasonable time and used the remaining data points for parameter tuning and testing. We used exactly the same training, tuning and testing data points for LPR for fair comparison. We repeated this procedure 25 times and averaged the results.

Table 3.6 shows the results of AUC performance comparison between LPR and SVR as well as algorithm solving times in seconds. After significance testing we observe that for 11 out of 12 datasets the results are not significantly different. This shows us that both algorithms are equally good in maximizing the AUC. The strength of our formulation is the speed, meaning that we can obtain the same performance much faster then SVR. We were also able to compare the two algorithm in terms of classification accuracy. The results given in Table 3.7 shows that LPR performs similar to SVR tying 9 times out of the 12 datasets. We observe larger standard deviations here compared to performance tests vs. SVM. The reason is that here we average 25 runs with random sampling as opposed to averaging 5 cross-validation runs which are already the average of results from 10 folds.

In conclusion, in this chapter we demonstrated that LPR produces significantly better AUC results then 2-norm SVM using benchmark datasets from UCI machine learning data repository. It also performs fairly well in terms of classification performance against a

Table 3.6: AUC performance comparison: LP Ranker (LPR) vs. Support Vector Ranker (SVR) results averaged over 25 runs with random sampling

Datasets	LPR	SVR	t_{LPR}(s)	t_{SVR}(s)
BOSTON	0.9456(0.0340)	0.9404(0.0337)	0.22	9.47
ECOLI	0.9595(0.0193)	0.9626(0.0195)	0.34	33.5
GLASS	0.9480(0.0614)	0.9435(0.0680)	0.20	23.9
HEART	0.8743(0.0319)	0.8751(0.0341)	1.23	196
SONAR	0.9246(0.0475)	0.9581(0.0271)	2.3	201
SPECTF	0.8697(0.0244)	0.8731(0.0293)	1.4	109
WPBC	0.7369(0.1032)	0.7609(0.1189)	1.6	79
IONOSPHERE	0.9657(0.0204)	0.9672(0.0187)	1.28	160
HABERMAN	0.6678(0.0628)	0.6759(0.0668)	1.52	98.3
LIVER	0.7330(0.0338)	0.7377(0.0341)	2.17	188
WBC	0.9947(0.0028)	0.9946(0.0026)	0.42	155
WDBC	0.9924(0.0039)	0.9930(0.0040)	0.42	167
(W,L,T)	(0,1,11)			

state-of-the-art classification algorithm. When we compare our algorithm against an AUC-optimizing SVR we observe that LPR performs equally well as SVR while dramatically reducing algorithm solving times.

Table 3.7: Accuracy performance comparison: LP Ranker (LPR) vs. Support Vector Ranker (SVR) results averaged over 25 runs with random sampling

Datasets	LPR	SVR
BOSTON	0.9373(0.0181)	0.9384(0.0156)
ECOLI	0.9408(0.0177)	0.9429(0.0189)
GLASS	0.9816(0.0230)	0.9708(0.0205)
HEART	0.7883(0.0422)	0.7889(0.0486)
SONAR	0.6753(0.1330)	0.5976(0.0852)
SPECTF	0.8160(0.0312)	0.8251(0.0316)
WPBC	0.7807(0.0650)	0.7748(0.0534)
IONOSPHERE	0.9112(0.0283)	0.7840(0.1251)
HABERMAN	0.7243(0.0340)	0.7306(0.0313)
LIVER	0.6969(0.0453)	0.6965(0.0356)
WBC	0.9616(0.0123)	0.9600(0.0160)
WDBC	0.9589(0.0133)	0.9636(0.0184)
(W,L,T)	(3,0,9)	

CHAPTER IV
SPEEDING UP THE OPTIMIZATION

4.1 Motivation and Background

Optimization-based learning algorithms are usually not robust to increasing number of data points. The solution times can increase dramatically with the number of data points making it infeasible to apply such algorithms to large or even moderately-sized problems. This issue has been addressed in the linear programming (LP) and SVM literature over the years, providing a wide array of speed-up approaches for optimization problems in the context of classification [8, 66, 36]. However, the nature of the ranking problem introduces a different challenge making traditional constraint reduction methods, typically used in classification problems, inappropriate.

The formulation of a mathematical program for ranking introduced in this dissertation includes a constraint for each positive-negative pair, resulting in a quadratic number of constraints, and limiting the size of solvable problems. Several methods have been proposed in the literature to address the problem of solving a large number of constraints in reasonable time, the most significant being the chunking algorithm [8]. Chunking divides the data into manageable bins and optimizes the bins separately so that the whole problem can be solved in shorter time. After solving for each chunk, the points that are the most influential are added to the next bin as the algorithm iterates through the whole dataset. Unfortunately chunking, in its original form, is not a good fit for the ranking problem. Unlike classification problems, the influential points that determine the ranking in each chunk are not necessarily relevant to other chunks. The location of influential points is highly data dependent, and unlike support vectors they do not necessarily appear close to a decision surface. Because of this reason those points can be very different even for a slightly different subproblem, making chunking-like approaches ineffective.

Constraint reduction in ranking problems where constraints represent pairwise orderings has not been addressed consistently in the literature. Hueristics to reduce constraints have included random selection of a subset of constraints, a nearest neighbor-like data reduction approach and clustering of data points [48, 10, 79, 97]. Herschtal et al. suggested the redundancy of some of the positive-negative pairs. They proposed to randomly choose a subgroup of pairs, which does not specifically remove the redundant pairs but reduces the number of constraints and therefore reduces the space complexity to $\mathcal{O}(n)$. The authors did not compare their results to any known algorithm making it difficult to judge the relative performance. Brefeld et al.'s formulation has space complexity of $\mathcal{O}(n^4)$ (pn variables and pn constraints). The authors proposed a speed-up approach that reduced this to $\mathcal{O}(n^2)$ by approximating this problem, representing all the pairs by $p+n$ cluster centers. They utilized Goswami et al.'s fast k-means clustering [42] to reduce the data. Rakotomamonjy proposed a nearest-neighbor approach, where only the k neighboring negatives of each positive point are ranked lower then the positive. Most of the methods used are ad-hoc and employed data that is not publicly available.

A common approach which is known to help in solving large scale optimization problems faster but with reduced accuracy, is the subgradient approach. Subgradient methods are used to solve convex optimization problems where the objective function is non-differentiable. They work similar to gradient methods with a few modifications and have guaranteed convergence for certain step size strategies [69]. Originally introduced by Shor [85], the approach has been shown to work for maximum-margin learning problems [80]. Despite being a powerful approach to approximate and solve large problems within reasonable time, the subgradient method has the drawback of requiring several control parameters (such as starting point, step size, convergence criteria) that need to be fine-tuned for good performance.

To summarize, quadratic expansion of the number of constraints with the increasing number of data points remains a challenge. In our formulation the number of constraints

needed to obtain the optimal solution is equal to the number of positive points times the number of negatives. This number grows very quickly as the number of data points increases making the LP solution time unreasonably long. To tackle this issue we have implemented a number of exact and approximate methods [1] to reduce the number of constraints. Exact methods use heuristics to solve the original problem by iteratively building the set of required constraints needed for the exact solution. Approximate methods on the other hand solve an approximation to the original problem either by a data reduction approach or by an early termination of the algorithm by allowing a large tolerance.

4.2   Exact Methods

We have tried several speed up heuristics, related to chunking [8], to reduce the number of constraints by removing potentially redundant ones. Chunking divides the data into manageable bins and optimizes them separately so that the whole problem can be solved in reasonable time. After solving for each chunk, points that are most influential (points with nonzero α's) are added to the next bin as the algorithm iterates through the whole dataset. Unfortunately the original chunking algorithm did not work well for our formulations. The reason is that in our case non-zero α's represent ranking vectors. They are far away from the decision boundary and in general are not necessarily relevant to other chunks as in regular chunking. Therefore we made some adjustments to the algorithm to better fit the structure of our problem. A general overview of the modified chunking algorithm is shown in Figure 4.1. Note that the schemes we implement here can still obtain the optimal LPR solution since the LP will always include the set of *necessary* constraints to correctly rank the data points.

We experimented with two variations of chunking. In the first scheme, after each chunk we evaluated the weights on the remaining data points, and for the next chunk we

Create all possible constraints, C

Divide C into manageable chunks, C_i

Initialize empty constraint set S

while size of S not converged

    for each chunk of constraints, $C_i \subset C$,

        Solve (3.3) with constraints $S \cup C_i$, obtaining α^*

        Evaluate α^* on $C \setminus C_i$ for pairwise violations

        Add violated constraints to S

    end for

end while

Figure 4.1: Chunking Algorithm Outline

add to the set of previously violated constraints only those constraints that represent pairwise violations. In this case we observed a majority of the violated constraints accumulating during the first few iterations instead of showing a gradual increase through iterations as we expected. As a second approach, to reduce the effect of a sudden increase of constraints, for each chunk we added only the top 1000 most violated constraints in the LP. This seemed to work well in most of the datasets. In this case the number of passes, hence the number of iterations until convergence is increased, slowing the LP solution time. An overview of the speed up comparisons are given in Table 4.1. The second column shows the total number of possible constraints for the given dataset. Columns 3 to 5 show the results for the first approach and columns 6 to 8 show the results for the second approach. The column labeled "pass" in columns 3 and 6 represents the number of full passes by the algorithm on the dataset until the number of retained constraints converged. With the implemented approaches we were able to reduce the number of constraints needed to solve the optimization problem in most of the datasets.

Table 4.2 shows the times in seconds for running each algorithm on the benchmark

Table 4.1: Comparison of speed-up heuristics

Dataset	#cons.	1st Speed up Approach			2nd Speed up Approach		
		#pass	#cons.	%reduc.	#pass	#cons.	%reduc.
BOSTON	18172	2	4920	72.92	5	4501	75.23
ECOLI	12336	3	4170	66.20	4	4312	65.05
GLASS	4509	3	3036	32.67	3	3039	32.60
HEART	14824	3	6730	54.60	5	4692	68.35
SONAR	8888	5	3874	56.41	5	3939	55.68
SPECTF	20240	3	4775	76.41	5	3186	84.26
CANCER(WPBC)	5628	3	3981	29.26	5	3585	36.30
IONOSPHERE	23142	2	5850	74.72	4	5060	78.13
HABERMAN	15022	2	12580	16.26	7	6409	57.34
LIVER(BUPA)	23711	2	22500	5.11	8	8250	65.21
CANCER(WBC)	86616	2	7370	91.49	2	3288	96.20

datasets. As we explained earlier, in general we did not achieve any improvements on the solution times versus LPR even though we managed to reduce the number of constraints generated. Although we solve a smaller size problem at each iteration, the number of passes for convergence is the basic factor that dictates the overall solution times. Slow convergence was the reason for these slower timings. The significance of these results is that we were able to solve larger problems with especially the second speed-up approach since the number of constraints solved is relatively small to the original problem size. In other words, we were able to solve problems that were otherwise not solvable with LPR because of memory limitations. We do not include those results here, because we cannot run any performance comparison using the original formulation.

Table 4.2: Comparison of algorithm run times LPR vs 1st and 2nd speed up approaches, in seconds

Dataset	t_{LPR}(sec)	t_1(sec)	t_2(sec)
BOSTON	35	62	52
ECOLI	23	16	24
GLASS	1.3	4.4	4.6
HEART	30	43	49
SONAR	11	26	42
SPECTF	66	83	39
CANCER(WPBC)	8.6	19	25
IONOSPHERE	54	69	47
HABERMAN	156	496	221
LIVER(BUPA)	195	1450	1070
CANCER(WBC)	172	167	60

4.3 Approximate Methods

4.3.1 Hierarchical Local Clustering

The quadratic expansion of the size of the problem becomes an issue as the dataset sizes reach around 1000 points, especially with balanced class distributions. In this section we propose a clustering approach to reduce the number of data points which in turn will reduce the number of constraints.

Clustering, typically used to find a structure in a collection of unlabeled data, is an unsupervised learning algorithm. Clustering can also been used as a data reduction method, by representing all the points in a cluster by a representative center point. In this way, clustering can be used to reduce the number of points in a large optimization problem. Such methods have been shown to perform well in reducing the size of optimization-based

classification and ranking algorithms [10, 97].

In a classification problem, only one class transition is needed, so clustering points that are all on one side of the boundary will have little effect on the learning outcome as long as the class of the clusters is correctly identified. To optimize ranking, however, our scoring function needs to differentiate between all neighboring points of different classes. Therefore, in order to obtain the best possible ranking vectors, regions containing class transitions should not be clustered. This enables data points to be clustered in the regions where the data is strictly from one class, forming cluster centers in characteristic regions. At this point this approach is clearly different from then Brefeld's reduction heuristic [10]. Brefeld used a k-means approach where each cluster may have members with both class labels. Yu et al.'s heuristic [97] is similar around the boundary region in the sense that it preserves boundary points from being clustered. However we do this all across the data since our formulation does not have a single decision boundary. A critical decision when constructing a clustering algorithm is the selection of an effective stopping criterion. Excessive clustering would leave the optimization method unable to correctly score neighboring points, while a conservative approach may not reduce the data to desired sizes. Here we will use class transitions as a stopping criterion for the clustering approach we propose.

Our algorithm [3] clusters same-class regions into a single cluster center as long as no point of the opposite class exists in those regions, and assigns a weight to that center proportional to the number of points clustered. The algorithm is set up to build similar to an agglomerative clustering approach, that is, the two nearest cluster neighbors are merged if they are from the same class. If they are from the opposite class, those clusters are flagged and cannot be used again. The algorithm stops when all neighboring clusters are from opposite classes. Once we have the reduced number of data points, we create the regular constraints to rank positives higher than negatives. Each constraint will have a significance weight, based on the weights obtained from the clustering algorithm. Therefore a violated constraint that is enforcing a large chunk of positives over a large chunk of negatives will

get a much higher penalty in the objective function. The outline of the clustering heuristic is given in Figure 4.2 and a graphical illustration of the clustering iterations is given in Figure 4.3 on a simple two-dimensional problem.

initialize distance matrix using all data points

set each point x_i as a cluster center and initialize $w_i = 1$

while all centers are not flagged

    find closest centers x_i , x_j

    if x_i and x_j are from the same class

        merge to the geometric center: $x_c = \frac{w_i x_i + w_j x_j}{w_i + w_j}$

        set $w_c = w_i + w_j$

        remove x_i and x_j from data

        modify distance matrix

    else

        flag x_i and x_j

    end if

end while

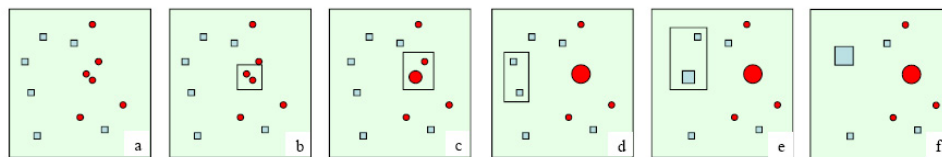Figure 4.2: Agglomerative local clustering pseudocode

Figure 4.3: Progress of clustering iterations: Nearest same-class points are clustered iteratively (3a-3e) until every nearest neighbor is from the opposite class (3f).

Once the clustering is complete, the objective function is modified to add weights to the error terms. The weight on error $z_{i,j}$ is the product of computed point weights w_i

and w_j, representing the fact that the new constraint replaces $w_i w_j$ old ones, one for each positive/negative pair in the two clusters. We designate the set of cluster centers as Ω with positive cluster centers Ω^+ and negative cluster centers Ω^- such that $\Omega^+ \cup \Omega^- = \Omega$. The new objective is given as:

$$\min_{\alpha,z} \quad \sum_{l\in\Omega} \alpha_l + C \sum_{s\in\Omega^+,t\in\Omega^-} w_s w_t z_{s,t}. \tag{4.1}$$

4.3.2 Solving the LPR by Subgradient Method

The subgradient method is used to solve convex optimization problems where the objective function is non-differentiable and in particular can be expressed in the following form:

$$\min_{x} \quad f(x) = \sum_{i=1}^{m} f_i(x)$$
$$\text{s.t.} \quad x \in X, \tag{4.2}$$

where $f_i : \Re^n \to \Re$ are non-differentiable convex functions and X is a convex subset of $\Re^n$. It is not hard to see that, with a few adjustments, our problem can be written in this form.

From (3.3) we can re-write the set of pairwise errors as

$$z_{i,j} = \max\{0, 1 - \sum_{l\in X} y_l \alpha_l [k(x_i, x_l) - k(x_j, x_l)]\}, \quad \forall x_i \in X^+, x_j \in X^-. \tag{4.3}$$

For conciseness we refer to $\sum_{l\in X} y_l \alpha_l [k(x_i, x_l) - k(x_j, x_l)]$ as the LHS's (left-hand-sides) of the constraint matrix. We can rewrite the objective function in (3.3) as

$$f(\alpha) = \sum_{l\in X} \alpha_l + C \sum_{i\in X^+,j\in X^-} \max\{0, 1 - LHS\}. \tag{4.4}$$

In this form the formulation can be approximated by a subgradient method. The approach we use here is very similar to a descent method with a few important modifications. We will use the typical gradient descent update rule and the set of α's will be updated at each

iteration as

$$\alpha_{k+1} = P_\alpha(\alpha_k - s_k g_k), \tag{4.5}$$

where k is the current iteration, s_k is a positive step size at k, g_k is a subgradient ($\partial f(\alpha_k)$) of the function at α_k at k, and P_α denotes projection of α onto the feasible region. The update, $\alpha_k - s_k g_k$, may cause some of the α's to become negative which makes them infeasible for the original problem. Therefore at each iteration we project the current solution back into the feasible region, if necessary, by setting any negative α's to 0. As with any subgradient method, the objective value is not guaranteed to be monotonically decreasing, in contrast to regular gradient decent methods. Therefore it is common to keep track of the best solution, in our case the best set of α's, throughout the iteration process.

In the literature there are many results on the convergence of the subgradient methods. For constant or diminishing step size, the algorithm is guaranteed to converge to within a ϵ-range of the optimal value in a finite number of iterations [69]. Step size rules we considered for our problem are constant step size, $s_k = c$, where c is a constant independent of k, and diminishing step size, $s_k = a/\sqrt{k}$, where $a > 0$ and constant. We also tested the dynamic step size rule from Nedic and Bertsekas [69] given as

$$s_k = \lambda \frac{f(\alpha_k) - f_{target}}{\|g_k\|^2}, \tag{4.6}$$

where λ is a convergence parameter given as $0 < \lambda \leq 2$ and f_{target} is a targeted objective function value. For our choice of λ we used the dynamic rule suggested by Lorena and Senne [63], where λ is halved after a given number of iterations if the objective does not improve. The algorithm terminates when a very small preset level of λ is reached. After some experimentation we decided to use the dynamic step size approach since it was the fastest in terms of convergence and required less parameter tuning when applied to different problems.

The subgradient method with dynamic step size requires a number of parameters to be set. The initial point, α_0, is arbitrarily set, typically to a vector of zeros. However, it

is possible to start from a more favorable point (warm start) if one has more information about the problem. The step size parameter λ_0 is the starting value for λ which is a value between 0 and 2, and is typically set to 2. The parameter λ is updated, typically by halving its current value (although that reduction can be dynamic as well), when the current best objective function value does not improve for h iterations. λ_{end} is the stopping criterion of the subgradient algorithm. The algorithm stops once the current λ falls below this threshold, typically set to 0.005. Another parameter is the difference between current and target objective function, given as the numerator in (4.6). This value designates how much improvement is desired toward the optimal value. For our experiments we set this difference to be 10% of the current objective function value, such as $f(\alpha_k) - f_{target} = 0.1f(\alpha_k)$. We have also experimented with $0.05f(\alpha_k)$ and $0.2f(\alpha_k)$ but did not observe significant change in performance. This self-adjusting nature of the dynamic step size makes it favorable for the application to different datasets with minimal supervision (parameter tuning). The pseudocode for the proposed subgradient method with dynamic step size is given in Figure 4.4.

One requirement of our ranking formulation, different from its classification counterparts, was that the whole constraint matrix needed to be built before solving. This was a major problem since the number of constraints increases quadratically with the number of data points. One important advantage of the subgradient method is that unlike the LP formulation, it does not require the whole constraint matrix to be built, avoiding huge memory requirements for solving the problem. Another advantage of the subgradient method is that each subgradient is cheaper to compute (linear complexity with respect to the number of constraints) at each iteration as opposed to the LP pivoting (quadratic complexity with respect to the number of constraints). Although the time for each iteration still increases quadratically with the increasing number of data points, the amount of time it takes to solve the problem is reduced dramatically, especially if subgradient converges fast, as we show in the next section.

initialize λ_0, λ_{end}, α_0, h

set $f_{best} = \infty$, $k = 1$, $k_{best} = 0$, $h_{count} = 0$

while $\lambda > \lambda_{end}$

    if $f(\alpha_k) < f_{best}$

        $f_{best} = f(\alpha_k)$

        $k_{best} = k$

        $h_{count} = 0$

    else if $h_{count} < h$

        $h_{count} = h_{count} + 1$

    else

        $\lambda = \lambda/2$

        $h_{count} = 0$

    end if

    $g_k = \partial f(\alpha_k)$, $s_k = \lambda \frac{f(\alpha_k) - f_{target}}{\|g_k\|^2}$

    $\alpha_{k+1} = P_\alpha(\alpha_k - s_k g_k)$

    $k = k + 1$

end while

Figure 4.4: Pseudocode for subgradient approximation with dynamic stepsize

4.4 Results and Discussions of Approximate Methods

For the experiments in this chapter we used the datasets introduced in Chapter 3 from UCI data repository. We constructed the same grid search to find the best RBF parameter settings with $C = \{1, 10, 100\}$ and $\gamma = \{0.01, 0.1, 1\}$. For all the datasets, we averaged 5×10-fold cross validation results. We ran our experiments on a 3.2 Ghz Pentium-IV with 2GB of memory.

4.4.1 Clustering Approximation Results

According to the results given in Table 4.3 the reduction in the size of data is on average about 57% causing an average reduction of about 76% in the number of constraints. We can see a substantial speed-up in the LP solving times after adopting the clustering heuristic (Table 4.4). The times shown for the clustering algorithm include the calculation and updates of the distance matrix that are used to identify nearest points, the time to cluster the data set and finally to solve the LP with the reduced data. With this approach we were able to solve larger datasets in reasonable time. Results show that the time to solve the approximate problem is faster by up to seven times in some cases.

Table 4.3: Reduction in number of points and constraints using agglomerative local clustering

Dataset	#points	reduc.#pts	%reduc.	#con.	reduc.#con.	%reduc.
boston	406	83	79.56	14313	1416	90.11
ecoli	270	57	78.89	9576	702	92.67
glass	172	24	86.05	3552	108	96.96
heart	216	114	47.22	11520	3224	72.01
sonar	167	117	29.94	6942	3422	50.71
spectf	282	159	43.62	15912	4760	70.09
wpbc	156	100	35.90	4403	2139	51.42
ionosphere	281	125	55.52	18180	3204	82.38
haberman	245	137	44.08	11700	4510	61.45
liver	276	178	35.51	18560	7872	57.59
segment	168	38	77.38	3456	240	93.06
wdbc	456	122	73.25	48620	3696	92.40
Average			**57.24**			**75.90**

We observe that the biggest performance gain was achieved for problems where large chunks of same class data points were isolated from the other class. These are the cases where it is possible to represent large number of points by only a few cluster centers maintaining the high quality of the ranking generalization. As the classes mix up in the original feature space the clustering becomes less effective in terms of data reduction.

Table 4.4: Comparison of algorithm run times LPR vs LPR(clus), in seconds

Dataset	t_{LPR}(sec)	$t_{LPR(clus)}$(sec)	%reduc.
boston	11.89	3.67	69.13
ecoli	4.99	1.29	74.15
glass	0.95	0.47	50.53
heart	7.97	1.88	76.41
sonar	4.33	1.91	55.89
spectf	12.99	3.49	73.13
wpbc	1.82	0.98	46.15
ionosphere	10.24	2.28	77.73
haberman	31.16	6.37	79.56
liver	107.43	22.34	79.21
segment	0.89	0.51	42.70
wdbc	47.77	5.83	87.92

We compared the AUC performance of our algorithm with clustering (LPR(clus)) vs. SVM (Table 4.5). Although there is a slight performance loss due to data reduction from our results in the previous chapter, we still obtain better or comparable results to SVM with this approach.

Here we would like to note that we obtain faster LPR timings for the same datasets

in Table 4.4 compared to Table 4.2 from exact heuristics. We overhauled many of our procedures in our Matlab code to run our implemented algorithms more efficiently with less memory requirements. This happened while we were implementing the approximate methods but long after the exact methods.

4.4.2 Subgradient Approximation Results

As a result of its heuristic nature, the subgradient algorithm relies on good parameter selection for good performance. One very important parameter that affects the performance is the step size. In our tests we used a number of different step size approaches. As mentioned earlier all the step sizes we used are guaranteed to converge to some proximity of the optimal solution. However, convergence may take quite some time depending on the choice of step size. We tested convergence times with three different step size approaches: constant c, diminishing $1/\sqrt{k}$, where k is the current iteration, and the dynamic step size given in (4.6). The convergence plot on a sample problem is given in Figure 4.5. The comparison suggests that although $1/\sqrt{k}$ converges very fast in the first few iterations, it slows down and falls behind the dynamic step size in the later iterations. On this sample problem the subgradient approach using dynamic step size converges and terminates much faster than the other two. Fast convergence and ability to self-adapt to different problems made the dynamic step size was our choice for the rest of the experimentation.

We carried out a comprehensive performance comparison between LPR, LPR(clus), LPR(sub) and 2-norm SVM in terms of AUC performance over 12 datasets from the UCI repository. The results are given in Table 4.5. We carried out significance tests at 0.05 level using 5×10 fold cross-validation results. The total win-loss-tie results for all 12 datasets are given in Table 4.6. We observe that LPR and both constraint reduction heuristics perform very well against SVM in terms of ranking performance. LPR is slightly better than two heuristics. However it is the slowest since it solves the complete problem. The two heuristics are comparable in performance with no significant performance difference

in 9 datasets, although LPR(sub) is significantly better on the remaining 3 datasets. One incident we would like to note here is that in one case LPR(sub) performed significantly better then LPR. In other words the subgradient approximation performed better then the original LP formulation. This is not unusual since the results are averaged over the unseen test set, and it is possible that an early termination of subgradient method may have better generalization, simply because of less over-fitting for that specific problem.
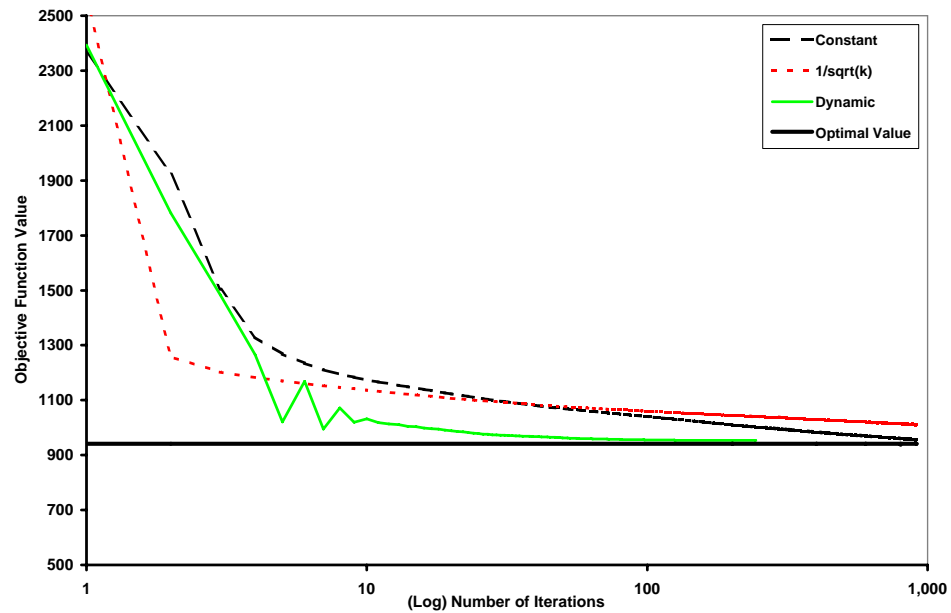
Figure 4.5: Convergence rate with various step size approaches

One important question to pose at this point is how the solving times compare for the introduced heuristics. For this purpose we compared solving times for LPR, LPR(clus) and LPR(sub) with an increasing number of data points on a sample dataset. Experimental results are given in Figure 4.6. As the figure shows LPR cannot solve the problem once the number of data points is above 550, which corresponds to roughly 75,000 constraints, because of memory limitations. LPR(clus) is significantly faster then LPR and can solve this problem entirely. However, this heuristic will suffer from memory limitations once the number of data points is increased further. LPR(sub) on the other hand seems to be the

Table 4.5: Comparison of the AUC performance. LPR: LP Ranker, LPR(clus): LP Ranker with clustering, LPR(sub): LP Ranker with subgradient method

Dataset	LP Ranker	LPR(clus.)	LPR(sub)	SVM
boston	0.9654(0.0100)	0.9632(0.0056)	0.9634(0.0039)	0.9484(0.0082)
ecoli	0.9598(0.0080)	0.9601(0.0019)	0.9599(0.0029)	0.9494(0.0022)
glass	0.9708(0.0064)	0.9426(0.0213)	0.9666(0.0051)	0.9511(0.0064)
heart	0.9072(0.0044)	0.9026(0.0025)	0.8961(0.0053)	0.8566(0.0039)
sonar	0.9244(0.0055)	0.9119(0.0155)	0.9485(0.0105)	0.9027(0.0089)
spectf	0.9156(0.0060)	0.9001(0.0090)	0.9089(0.0053)	0.8926(0.0042)
wpbc	0.7314(0.0203)	0.7234(0.0297)	0.7414(0.0225)	0.7160(0.0159)
ionosphere	0.9735(0.0068)	0.9669(0.0044)	0.9791(0.0036)	0.9529(0.0018)
haberman	0.6564(0.0212)	0.6434(0.0136)	0.6759(0.0106)	0.6974(0.0064)
liver	0.7198(0.0190)	0.7323(0.0177)	0.7391(0.0044)	0.7239(0.0058)
segment	0.9878(0.0010)	0.9804(0.0059)	0.9837(0.0040)	0.9842(0.0026)
wdbc	0.9920(0.0012)	0.9914(0.0028)	0.9884(0.0008)	0.9868(0.0012)

fastest of the three for this example. One scenario in which the LPR(clus) may be favorable to LPR(sub) is when the dataset is distributed in somewhat distinct clusters in the feature space, which is not unusual in some real world problems. This way the algorithm may take full advantage of its strength. However, overall LPR(sub) is the clear winner among the two heuristics in terms of problem solving times.

4.5 A Case Study: CoIL Challenge 2000 - Caravan Insurance Data

Direct mailings to a companys potential customers, also knows as junk mail, can be a very effective way to market a product or service. However, much of this junk mail is really of no interest to the majority of the people that receive it. Most of it is discarded, not only

Table 4.6: Comparison of the statistical signifi-
cance results

Test @ 0.05	Win	Tie	Loss
LPR vs. SVM	10	1	1
LPR(clus) vs. SVM	5	6	1
LPR(sub) vs. SVM	9	2	1
LPR vs. LPR(clus)	1	11	0
LPR vs. LPR(sub)	2	9	1
LPR(sub) vs. LPR(clus)	3	9	0

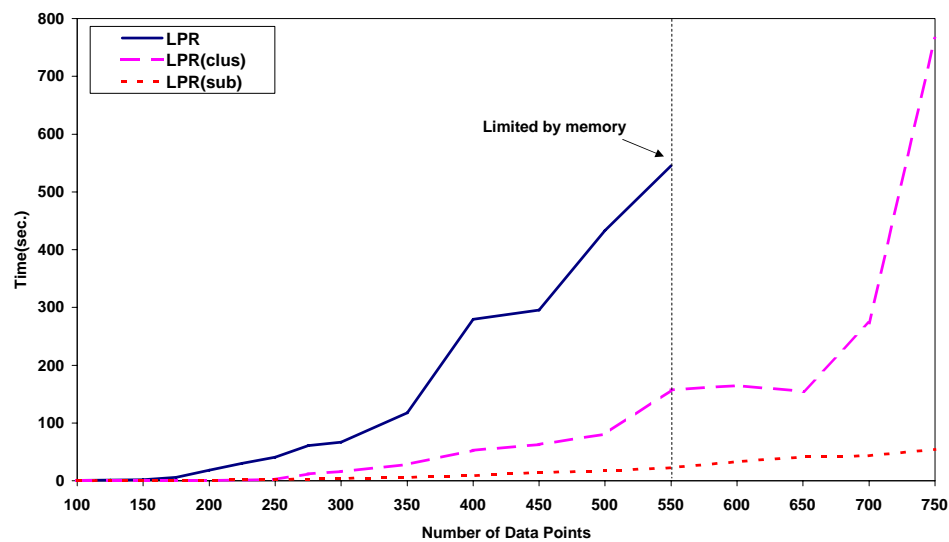Figure 4.6: Problem solving times vs. number of data points for all the rank optimization
approaches

wasting the money that the company spent preparing it, but also filling up landfill waste

sites or needing to be recycled. If the company had a better understanding of who their

potential customers were, they would know more accurately who to send it to, so some of

this waste and expense could be reduced.

The CoIL Challenge 2000 [90], a data mining competition organized by the the Computational Intelligence and Learning Cluster, sponsored by the EU, presented a real world problem from the domain of direct marketing. The problem was from an insurance company that wished to gain a better understanding of who their potential customers are, so that they would know more accurately who to send marketing information to.

The data was taken from a solicitation of 9,822 European households to buy insurance for a recreational vehicle (RV). The data was provided for the CoIL 2000 forecasting competition. For our analysis, we used a training set with 5,822 households and a testing set with 4,000 households. The training data are used to train and calibrate the model. Of the 5,822 prospects in the training data set, 348 purchased RV insurance, resulting in a hit rate of 5.97%. From the marketers perspective, this is the hit rate that would be obtained if solicitations were sent out randomly to consumers in the firms database. The problem can be posed such that we can calculate the top k% of the customers who are most likely to buy the insurance. We evaluate the performance of the model on the test set. In addition to the observed RV insurance policy choices, each data entry contains 85 additional variables, containing information on both socio-demographic characteristics such as size of the household, income level, education level, etc and ownership of various types of insurance policies such as car insurance, life insurance, etc.

The CoIL competition consisted of two parts: prediction and description. In the prediction task, the underlying problem is to find the subset of customers with a probability of having a caravan insurance policy above some boundary probability. The known policyholders can then be removed and the rest receive a mailing. The boundary depends on the costs and benefits such as of the costs of mailing and benefit of selling insurance policies. To approximate and simplify this problem, the challenge was to find a set of 800 customers out of 4000 in the test set that contains the most caravan policy owners. The purpose of the description task was to give a clear insight to why customers have a caravan insurance policy and how these customers are different from other customers.

There were 238 policy holders in the test set of 4000 customers. The winning model of the prediction task was by Charles Elkan. Using a naive Bayes approach he captured 121 of the 238 policy owners in the 800 customers selected out of 4000. A random selection would yield 42 policy owners. Runner ups were Petri Kontkanen from the University of Helsinki, Finland (115 policy owners), Andrew Greenyer from The Database Group, United Kingdom and Arnold Koudijs, Cap Gemini, The Netherlands (both 112). The winner of the description task was Kim and Street using artificial neural networks guided by an evolutionary attribute selection approach (ELSA/ANN) [58].

We approached this as a ranking problem such that we want to get as many policy owners as possible in the top 20% (800 customers) of the test set. We set up LPR such that it uses 2/3 of the 5822 points in the training set to train and the rest of the points for parameter tuning. Using the best parameters obtained we evaluated the performance of LPR on the test set of 4000 household. The number of constraints created by the LPR is close to 1 million. Obviously, it is not possible to solve this problem as is by LPR because of the memory limitations. Therefore we utilized a sub-gradient approach to approximate the LPR formulation.

Table 4.7: Hit rates by % split on CoIL data

Top k%	# Correct	Hit rate: LPR	Hit rate: ELSA	Recall
5	43	21.5	19.6	18.1
10	70	17.5	17.5	29.4
20	116	14.5	14.4	48.7
50	186	9.3	9.6	78.2

Table 4.7 shows that LPR captures 116 of the policy owners out of 238, corresponding to a 14.5% hit rate in the top 20% of the data. This number is significantly better than

the 5.97% hit rate over the whole data set. This means that if the marketers used our algorithm they would send solicitation to 1/5 of the customers capturing almost half (48.7%) of the would-be owners. 116 hits would have ranked second to Charles Elkan's model in the COIL challenge. Table 4.7 also shows the comparison of the hit rates at different levels for LPR and ELSA/ANN [58]. The results show that both algorithms perform equally well. Training of the ELSA/ANN approach is very complex and time consuming. It uses an evolutionary attribute selection approach as a wrapper to reduce the dimensionality of the data, where it evaluates many models before solving the original problem with a neural network algorithm. Our approach however is a straightforward optimization problem. The lift curve obtained from LPR model is given in Figure 4.7.

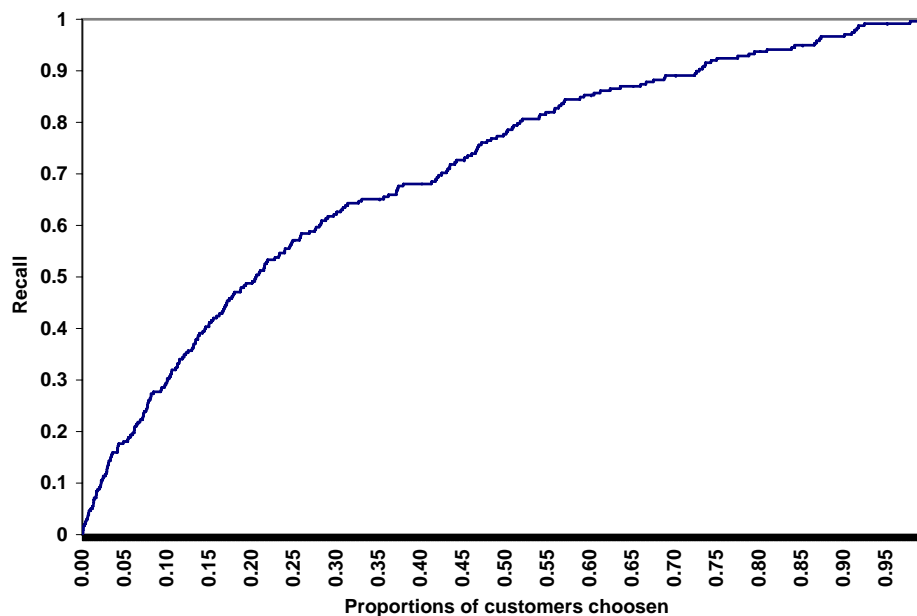Figure 4.7: Lift curve of the model by subgradient approximation of LPR

One of the setbacks of subgradient approach is the number of parameters that need to be optimized. Along with these parameters we also needed to optimize for RBF kernel parameters. For this case study, instead of using a grid search approach we simply tried a few combinations that yielded good performance on the tuning set. A rigorous parameter search

for this problem is set as a future work. The best results were obtained with RBF parameters of $C = 10, \gamma = 0.01$ and subgradient parameters of $\lambda_0 = 1, \lambda_{end} = 0.005, f_{target} = 0.05f$ and $h = 10$. Also instead of dividing λ by 2 at each iteration we divided it by 1.5 to allow more iterations. With these settings and roughly 1 million constraints each iteration took about 15 minutes on a dual core Pentium 3.2Ghz with 2GB of ram. It took around 170 iterations to solve the problem with these parameter settings. The convergence plot in Figure 4.8 and convergence of AUC in Figure 4.9 shows that after 20 iterations we were able to obtain relatively good solutions. Both figures show fluctuations in the convergence curves and the reason for this is the step size taken at each iteration. A large step size may lead to a worse solution in the next iteration, however a small step size can increase the convergence times dramatically. For this reason we used dynamic step size here, where the step sizes gradually reduced based on the objective function value. As we can see from the figures, the fluctuations are reduced as the number of iterations increases.
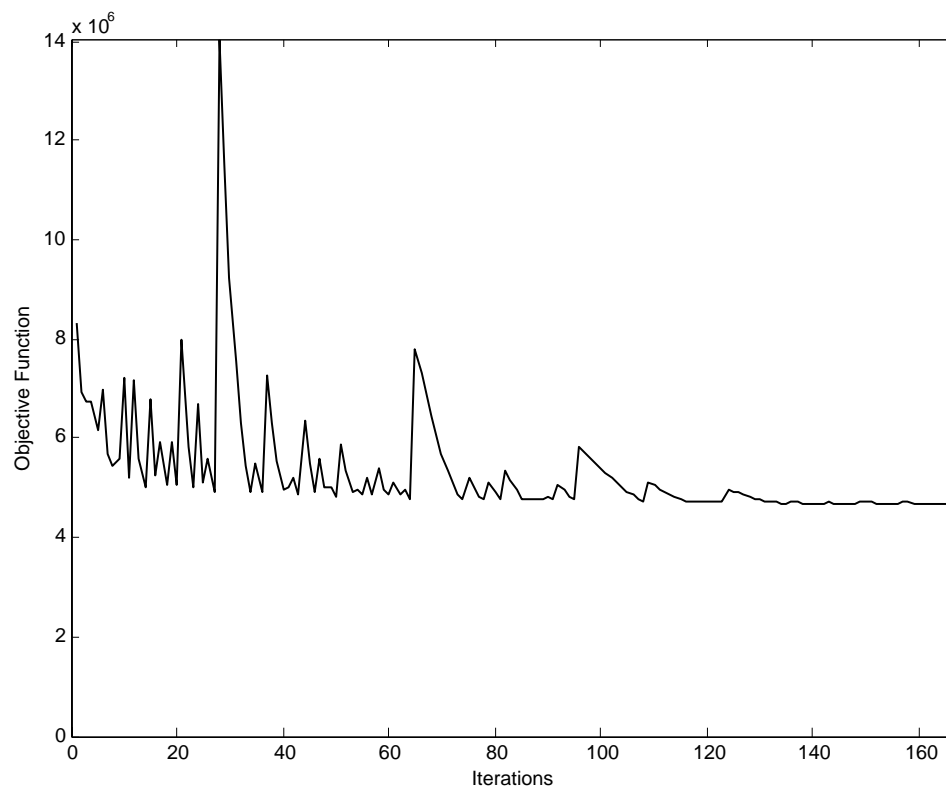

Figure 4.8: Convergence of objective function using the subgradient method
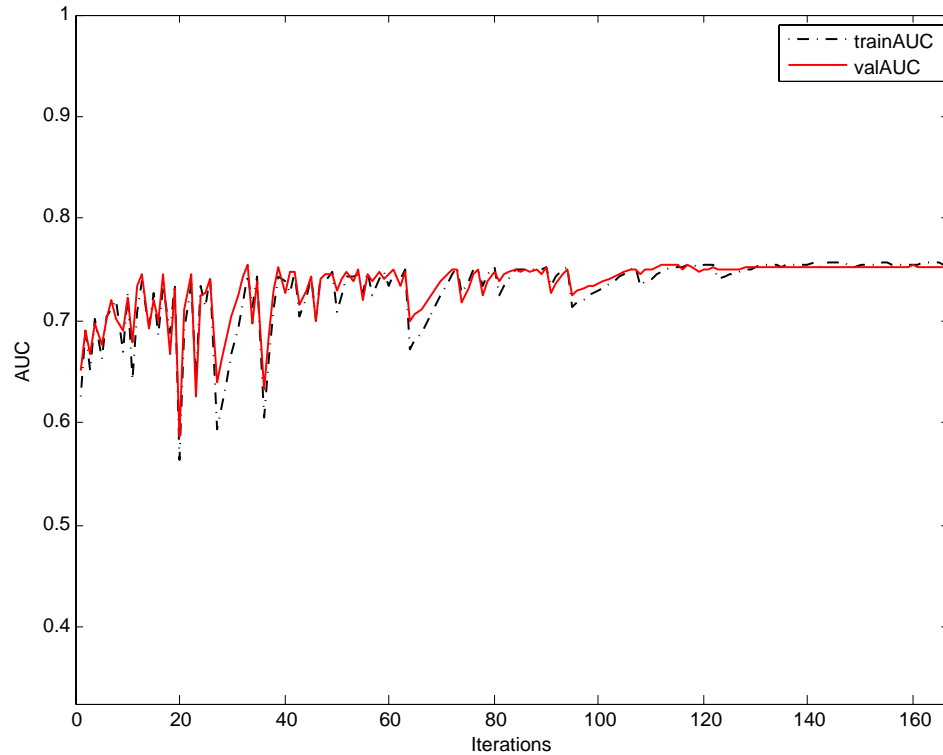
Figure 4.9: Convergence of training and validation AUC using the subgradient method

4.6 Chapter Recap

Optimization-based learning algorithms are usually not robust to increasing number of data points. The solution times can increase dramatically with the number of data points making it infeasible to apply such algorithms to large or even moderately-sized problems.

In this chapter we introduced a number of exact and approximate heuristics to speed-up the LP we solve. We used variations to the chunking approach to construct two exact heuristics. Although we were able to reduce the size of the problem, we were not able to improve LP solving times by using these heuristics.

Next, we introduced two approximation heuristics. The first heuristic is a hierarchical local clustering approach that reduces the size of the problem by systematically reducing the number of data points. This approach clusters same-class regions and maintains class

transition regions, which is essential for good performance in rank optimization. The clustering approach reduced the number of constraints by 76% on average, while improving the solution times by up to seven times with respect to LPR. The second heuristic is based on the subgradient method which approximates the LP formulation. We have tested three different step size approaches and observed that dynamic step size was the best choice. The subgradient approximation was significantly faster then both LPR and LPR(clus) especially for larger datasets.

The results presented show that our rank optimization algorithm with the proposed heuristics performs better in general against regular 2-norm SVMs in terms of AUC performance. Both approximate heuristics provide significant speed-up in solution times compared to LPR. Of the two heuristics the subgradient approach was the clear winner. It was not only faster, but also did not have any limiting memory requirements as the LP(clus) did with increasing problem size.

Although not addressed in this thesis, we believe it is possible to combine the two heuristic in order to solve even larger problems. Also, for larger problems we speculate that it would be more time-efficient to use ensemble methods. Each individual ranker in the ensemble maybe trained with a subset of training points such that every training point is used in at least one of the rankers in the ensemble. Then, orderings from each ranker can be aggregated to obtain a total ranking of all test points. We leave this as a future research direction.

As a real world application to the proposed heuristics we used the CoIL Challenge 2000, caravan insurance dataset. The problem was from an insurance company that wishes to gain a better understanding of who their potential customers are, so that they would know more accurately who to send marketing information to. We were able to solve the problem, which produced close to 1 million constraints without any memory limitations. The quality of the results were on par with the best algorithms from the CoIL challenge. We used a subgradient approximation to LPR and we were able to obtain good solutions as early as

20 iterations. Although the solution times heavily depend on the selected parameters, we were able to show that LPR is scalable when paired with the subgradient method proposed here.

CHAPTER V
SURVIVAL ANALYSIS BY RANK OPTIMIZATION

5.1   Motivation and Background

Survival analysis [59], also known as time-to-event analysis, is a branch of statistics that deals with death in biological organisms and failure in mechanical systems. In engineering it is also known as reliability theory or reliability analysis. Death or failure is called an *event* in the survival analysis literature, and so models of death or failure are generically termed time-to-event models. In the big picture it is not hard to see that the application of survival analysis is not limited to health care or engineering problems. The event can be virtually anything, such as time to divorce or dropping out of high school, time to a customer churning from a service, time to filing an insurance claim, etc.

The analysis of survival data is complicated by issues of censoring, where events can only be observed within a certain time window, and by truncation, where subjects enter the study only if they survive a sufficient length of time or subjects are included in the study only if the event has occurred by a given time. There are four different types of censoring: right truncation, left truncation, right censoring and left censoring. Right censoring is the most common of all. Right censoring means that the observation is incomplete because the subject did not experience the event during the time frame of the study. Typically, a study does not span enough time to observe the event for all the subjects in the study.

Another important concept in survival analysis is the *hazard rate*. Hazard rate is defined as the probability that an individual will experience an event at time t while that individual is at risk for the event. Thus, the hazard rate is really just the unobserved rate at which events occur. Although hazard rate is an unobserved variable, it controls both the occurrence and the timing of the events. A related representation is the *survival rate*, which is the cumulative probability of the event *not* occurring up to time t. A survival curve, therefore, always starts with unit probability of survival at time $t = 0$ (see Figure 5.1).

In survival analysis every observation has two outputs: status and time-to-event. Status is a binary output, $\{0,1\}$, where 1 means the event is observed and 0 means the observation is censored, or in other words the event has not occurred within the time frame of the study (event-free survival). Every observation also has a time output, which either represents the time to an event or an observed event-free time. Because of this structure, survival analysis does not map easily to either a regression or classification problem. Clearly regular regression models are inappropriate since we only know event times for some of the examples. To apply classification, we could treat the cases with observed events as positives and separate them from the non-events, but the fact that the events occurred at different times limits the utility of the resulting model. A better approach would be to separate the class "event by time t^*" but this fails to take advantage of some information (e.g., cases censored before t^* are left out completely).

A number of techniques have been developed to model and predict censored data points. These techniques include non-parametric approaches such as Life-Table analysis and Kaplan-Meier curves, parametric models such as Cox's proportional hazard regression, and models based on machine learning and optimization techniques.

One of the oldest methods for analyzing survival data is to compute a life-table [5] (also called a mortality table or actuarial table). A life-table is simply a frequency distribution table where the distribution of survival times is divided into a predefined number of intervals. For each of those intervals the number and proportion of subjects that survived the interval, the number and proportion of the cases that failed to survive the interval and the number and proportion of subjects that are censored in the interval is calculated. Once these numbers and proportions are known it is possible to calculate the survival function and hazard rate.

Another popular non-parametric method to analyze survival data is using Kaplan-Meier curves [57]. Kaplan-Meier curves are the maximum-likelihood estimates of the true population survival function, showing the probability of survival through time (Figure 5.1)

and can be computed as:

$$S(t) = \prod_{j=1}^{t} [\frac{n-j}{n-j+1}]^{\delta(j)},$$

where $S(t)$ is the estimated survival function, n is the total number of cases and $\delta(j)$ is 1 if the j'th case is censored and 0 otherwise. This estimate of the survival function is also called the product-limit estimator. The advantage of the Kaplan-Meier method over the life table method for analyzing survival and failure time data is that the resulting estimates do not depend on the grouping of the data (into a certain number of time intervals). Actually, the product-limit method and the life table method are identical if the intervals of the life table contain at most one observation.

Kaplan-Meier curves allow the survival rates of different populations in a study to be compared visually. For example, imagine in a medical study two groups of patients are given separate drugs to prevent a disease from recurring. A Kaplan-Meier curve displaying the estimated probability of disease-free time for each patient group can be plotted for comparison. Then using comparison techniques such as Gehan's generalized Wilcoxon test [39], it is possible to evaluate the statistical significance of the difference.

Survival analysis offers several parametric regression models for estimating the relationship of variables to survival times. These models are extensively used among researchers because they can offer insight into various important parameters such as hazard rate. Typical parametric models include fitting a survival function using Weibul, exponential, log-logistic, log-normal or gamma distributions.

The Cox proportional hazard model is the most general and extensively used of the regression models because it is not based on any assumptions concerning the nature or shape of the underlying survival distribution [20, 21]. A proportional hazards model has the property that individuals of concern have hazard functions that are proportional to one another. Although this is seemingly a strong statement, the method is still one of the most popular and successful approaches to analyzing survival data. A hazard function can be
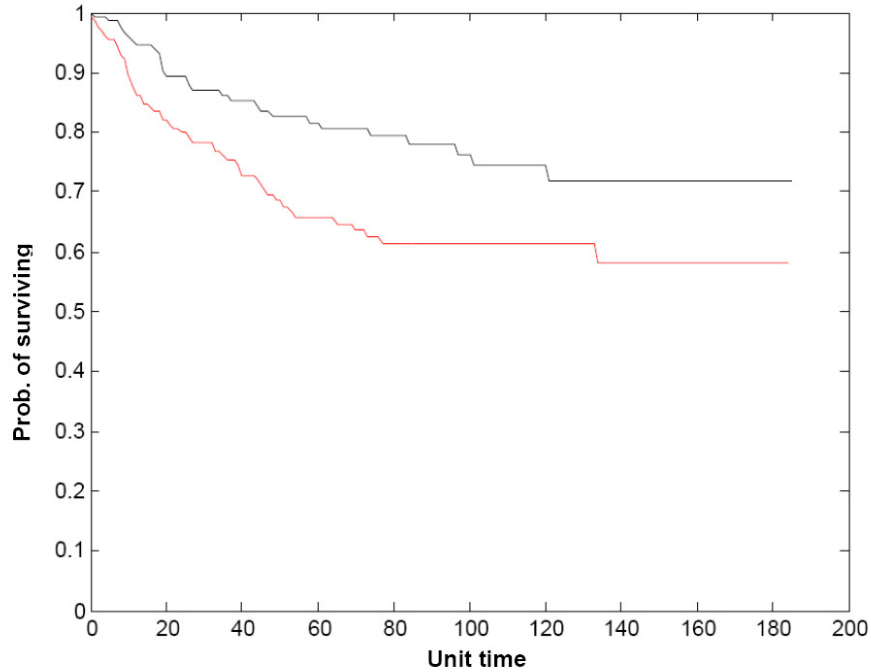
Figure 5.1: Sample Kaplan-Meier curves for comparison of two subject groups

written as $h(t|\mathbf{x}) = h_0(t)g(\mathbf{x})$, where $g(\mathbf{x})$ is a function of the inputs and $h_0(t)$ is a baseline hazard function. No particular shape is assumed for the baseline hazard; it is in general estimated nonparametrically. Cox's regression models the hazard function as the dependent variable and estimates the hazard multiplier using a linear combination of the independent variables yielding the common formulation

$$h(t|\mathbf{x}) = h_0(t)exp(\sum_{i=1}^{n}\beta_i x_i),$$

where the β's are regression coefficients.

Survival analysis has also gained attention in machine learning research providing a wide array of algorithms. The literature shows that neural networks are the most common approach to survival analysis. Ohno-Machado's study [70] showed that there is no significant difference in terms of performance between Cox's regression models and artificial neural networks (ANNs), and regression models provide more insight to the problem by providing information on which attributes were most influential for prognosis. On the

other hand we observe other studies where ANNs worked quite well for survival analysis [87, 71], however these studies were on a single dataset from a certain domain and did not have comparisons to common methods. A study by Zupan et al. suggested a weighting scheme on the subjects that enables machine learning algorithms to perform better with the survival data where the prediction of probability of an event (and not its probability dependency on time) is of interest [99]. Their work showed that the weighting scheme helped the machine learning algorithm to perform as well as or better than well-established statistical techniques. Optimization-based algorithms have also been developed for survival analysis. A linear programming approach applied to Wisconsin breast cancer study by Mangasarian et al. showed increased accuracy of both diagnosis and prognosis [67]. The novelty of their approach is that they were able to formulate an optimization problem that can handle censored data. Although this algorithm worked well, it is limited to building linear models.

In this chapter we approach survival analysis with a modified version [2] of a linear programming approach we introduced earlier. With this approach it is possible to order subjects by the risk for experiencing an event. Such an ordering enables determination of high-risk and low-risk groups among the subjects that can be helpful not only in medical studies but also in engineering, business and social sciences.

5.2 Survival Prediction using LPR

In this section we explain the modifications to LPR necessary to make it applicable to survival analysis with censored data. The primary observation here is that we no longer have positive and negative points, so the set of constraints will be constructed differently. When comparing the survival characteristics of two sets of cases (such as shown in Figure 5.1), we again use the Wilcoxon statistic, so the intuition is that we introduce constraints between "good" (long survival time) points and "bad" (short survival time) points.

To explain the constraint selection process we consider an example of a cancer prognosis problem (see for example the WPBC data in next section). We denote a single patient

as x_i, the outcome for this patient as p_i ($p_i = \{0,1\}$ where $0 = $ non-recur and $1 = $ recur) and time as t_i (where t_i is observed disease-free time if $p_i = 0$ or recurrence time if $p_i = 1$). In this data we cannot have constraints enforcing every recurring ($p_i = 1$) case to be ranked higher then every non-recurring ($p_i = 0$) case. This is because some patients have left the study (censored) and we do not know if the cancer recurred sometime in the future. All we know is their cancer-free time. Therefore we cannot have pairwise comparison for all the patients. We begin with constraints for pairs such that an earlier recurring case will rank higher (worse) if both cases recurred (5.1). Further, an earlier recurring case will rank higher than a longer cancer-free case (5.2). Therefore, we would ideally enforce the constraints

$$f(x_i) > f(x_j) \quad \text{if} \quad (p_i = 1, p_j = 1) \quad \text{and} \quad (t_i < t_j) \tag{5.1}$$

$$f(x_i) > f(x_j) \quad \text{if} \quad (p_i = 1, p_j = 0) \quad \text{and} \quad (t_i < t_j), \tag{5.2}$$

where $f(x)$ is the ranking function. As mentioned, we do not enforce a constraint between a censored point and a recurring point with a later time. We also avoid enforcing constraints between non-recurring cases. Intuitively, one might think that longer event-free times should be forced to rank lower than shorter ones, however, we find experimentally that explicitly including these constraints (possibly with a different weight) does not improve the results.

The objective function and the form of the constraints do not change from the classification formulation. However the sets of positive points and negative points, X^+ and X^-, are replaced by the set of uncensored points, X^u (the points with observed events), and the set of censored points, X^c. The new set of constraints is given as

$$\sum_{l \in X} y_l \alpha_l [k(x_i, x_l) - k(x_j, x_l)] \geq 1 - z_{i,j}, \tag{5.3}$$

$$\forall x_i, x_j \in X^u, \qquad t_i \leq t_j$$

$$\forall x_i \in X^u, x_j \in X^c, \quad t_i \leq t_j$$

5.3 Data and Experimentation

In this section we introduce the datasets that we use in this chapter. The datasets are from four studies in which the main target is to analyze time-to-event in the presence of censored data. Three of the datasets are from the field of health care and one from marketing.

5.3.1 Wisconsin Breast Cancer Prognostic Data

Wisconsin Prognostic Breast Cancer (WPBC) is a famous benchmark dataset that is also located in UCI Machine Learning Repository [6] for public access. This data has been used in past publications [88, 67]. The dataset used here, which is a larger version of the one found in the UCI Repository, contains 325 records. These are consecutive patients seen at the University of Wisconsin Hospitals and Clinics after 1984, and include only those cases exhibiting invasive breast cancer and no evidence of distant metastases at the time of diagnosis. There are 32 features of which 30 are computed from a digitized image of a fine needle aspirate of a breast mass. These attributes describe characteristics of the cell nuclei. The last 2 attributes are tumor size and number of positive lymph nodes. The endpoint (event) is cancer recurrence. We investigate the potential of effectively distinguishing high and low-risk patients. Such information is very helpful for doctors and patients deciding on a post-operative treatment regimen.

5.3.2 Seer Data

The National Cancer Act of 1971 mandated the collection, analysis, and dissemination of all data useful in the prevention, diagnosis, and treatment of cancer. The act resulted in the establishment of the National Cancer Program under which the Surveillance, Epidemiology, and End Results (SEER) program was developed. A continuing project of the National Cancer Institute, the SEER Program collects cancer data from designated population-based cancer registries in various areas. The dataset used in this work was

obtained from this study [13] and contains five attributes (histological grade, tumor size, tumor extent, number of positive and examined lymph nodes) for 44,135 breast cancer patients between the years 1977 to 1982. The endpoint in this data is cancer-related death. Similar to the WPBC data, we investigate the potential of effectively identifying high and low-risk cancer patients.

5.3.3 Burn Patients Data

Infection of a burn wound is a common complication resulting in extended hospital stays and death in severe cases. In a study to evaluate a protocol change in disinfectant practices in a large Midwestern university medical center, 154 patient charts and records were reviewed [52]. The study used medical records of patients treated during an 18-month study and provided burn wound infections and other medical information. In this problem we would like to identify high-risk patients so that they get the utmost care to prevent an infection. A successful evaluation of such a group can not only save lives but also reduce the cost of treatment.

5.3.4 Customer Churn Data

We were provided a dataset by a telecommunications company to investigate the predictability of customer churn. The data contains information on the customers' demographics, type of service, and payment details and has more than 150 attributes. It also contains information on length of service and whether the customer is still with the company or not. The complete dataset contains more then 400,000 records. We reduced the dimentions of the problem for robust evaluation by picking 5 attributes, both by conventional attribute selection techniques and by using expert domain knowledge. The most important problem to address from a business perspective is to identify customers with a high risk of churning. It is generally true that retaining customers is less costly for a company than obtaining new ones. Therefore if such a high-risk subset of customers can be identified, those customers

can be targeted with a stronger campaign to keep them. This will also result in reduced cost because a smaller group of customers can be targeted.

5.3.5   Experimental Procedure

The experiments are designed to compare our ranking algorithm against Cox's proportional hazard regression for predicting time to event. The algorithms are tested using ten-fold cross-validation, with the following procedure used to combine the folds. After a model is trained for one fold, it ranks the test points in order from high to low risk. Each point is then assigned a score from 0 to 100 based on the percentage of training points that it ranks above. These scores give us a common measure for combining the test points from different folds, creating a total ordering on the points in terms of testing prediction. Here we avoid directly using the output scores obtained from each algorithm because each fold is a different optimization problem and the range of scores obtained for test cases may not be consistent across folds of the cross-validation. This causes inconsistency once the scores from different folds are combined and results in reduced overall performance.

Since we know a true partial ordering of the cases, we are able to evaluate the performance by counting the number of pairs that are ordered correctly among the ones that can actually be ordered (remember that all subject pairs are not comparable because of censoring). This ratio of correct pairs to all possible pairs provides a metric for ranking performance which is similar but not equivalent to AUC. The higher this ratio, the better the performance of the algorithm in terms of ordering subjects by risk. In our experiments we repeated five 10-fold cross-validations and for each cross-validation we obtained a ranking ratio for each algorithm. The ratios for the two competing algorithms were then compared for statistical significance using a paired t-test at the 0.05 level.

We utilized Kaplan-Meier curves to visually compare the performance of the two algorithms. For all datasets, the ordered data is repeatedly split into 2 groups (high-risk and low-risk) by varying the cut-off points by the number of subjects. The split was performed

at every 10% level (from 10% to 90%) to visualize and compare performance for different cut-off points. Groups representing each split can be plotted with a Kaplan-Meier curve. In the following section, we show only those splits with domain-specific relevance, i.e., we isolate the top (and possibly bottom) 10% of the cases. We have also used the Wilcoxon test to evaluate if the two curves are significantly different for each split. Our ranking algorithm is optimized for all possible cut-offs, therefore we expected it to perform well for any given split.

Table 5.1: RBF Parameters for LPR

Dataset	C	γ
WPBC	1	0.05
SEER	1	0.05
BURN	1	0.10
CHURN	10	0.01

Our ranking algorithm has three basic input parameters: C, γ and $\mathbf{w}$ as introduced in Chapter 3. After preliminary analysis we obtained the parameters to be used for each dataset (Table 5.1). The last parameter, which is the weight vector that adjusts each pairwise ordering error in the objective function, is set to a vector of 1's ($\mathbf{w} = \mathbf{1}$).

5.4 Results and Discussions

Table 5.2 provides a comprehensive evaluation of ranking performance of both algorithms (using the *ratio* defined in Section 4.2). For all the datasets, we observe that our algorithm is significantly better in general, independent of what split is chosen. For example, in the case of customer churn data, our algorithm correctly orders 95.36% of all the possible pairs right, compared to 92% for Cox's hazard regression.

Table 5.2: Comparison of Algorithms: % of correctly ordered pairs averaged over five cross-validation runs

Dataset	LPR	Cox's Regression	p-value
WPBC	0.8926	0.8744	0.038
SEER	0.9238	0.8915	0.001
BURN	0.9097	0.8829	0.004
CHURN	0.9536	0.9200	0.001

Table 5.3: Comparison of 2 LPR's with different weighing schemes. LPR: using regular survival constraints, LPRex: using additional weak constraints

Dataset	LPR	LPRex	p-value
WPBC	0.8926	0.8928	0.1797
SEER	0.9238	0.9214	0.0025
BURN	0.9097	0.8949	0.0003
CHURN	0.9536	0.9458	0.00003

We have experimented using different levels of error weighing schemes. For example, we set w_i's to be a function of the time difference between each compared pair, penalizing more for incorrect ordering of pairs where the difference in time-to-event is larger. We tried a few variations but these changes did not consistently achieve better ranking performance for any of the datasets. Also, intuitively one might think that longer event-free times should be forced to rank lower than shorter ones, however, in our experiments we observed that explicitly including these constraints (possibly with a different weight) does not improve the results significantly for any of the datasets we used. Table 5.3 shows that

the results using additional (*weaker*) constraints (with $w = 0.5$) between non-recurrent patients (LPRex) either has negative or no effect on performance. We attribute this to our lack of knowledge on the correct cost of relative errors. We believe that a domain expert who can correctly hard-code such weights would benefit from such formulation.
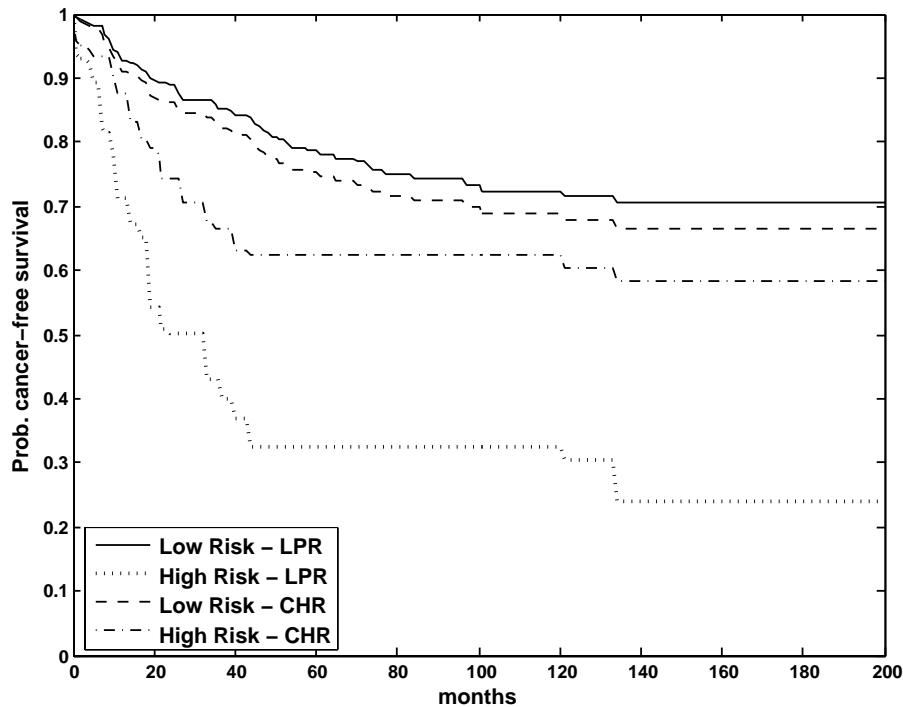

Figure 5.2: Kaplan-Meier curve for WPBC data: High-risk: Top 10%, Low-risk: Bottom 90%

As mentioned in the previous section, we evaluate the utility of our method on all the datasets by splitting the top and/or bottom 10% to designate high and low-risk groups. This threshold can be set differently based on domain knowledge. The first two Kaplan-Meier curves (Figures 5.2 and 5.3) illustrate the comparison between our ranking algorithm (LPR) and Cox's proportional hazard regression (CHR) on the WPBC data. In cancer treatment it is very important for the doctor to be able to identify a low-risk group, who might consider foregoing some postoperative treatments, and a high-risk group, who might choose more radical treatments. For this purpose we split the top 10% of the rank-ordered data from both

algorithms to represent the high-risk group. Each of these groups has the same number of people and is considered to have the highest risk of cancer recurrence. In Figure 5.2 we observe that our algorithm is more successful in predicting the high-risk cancer patients, while as expected the low-risk groups (remaining 90%) are not significantly different. At a typical time horizon of five years (60 months), we observed that only 30% of the high-risk group identified by our ranking algorithm were cancer-free, compared to about 60% for the Cox's regression model. This is a significant result because a doctor who evaluates these curves will have more accurate information on which patients have a higher risk of cancer recurrence.
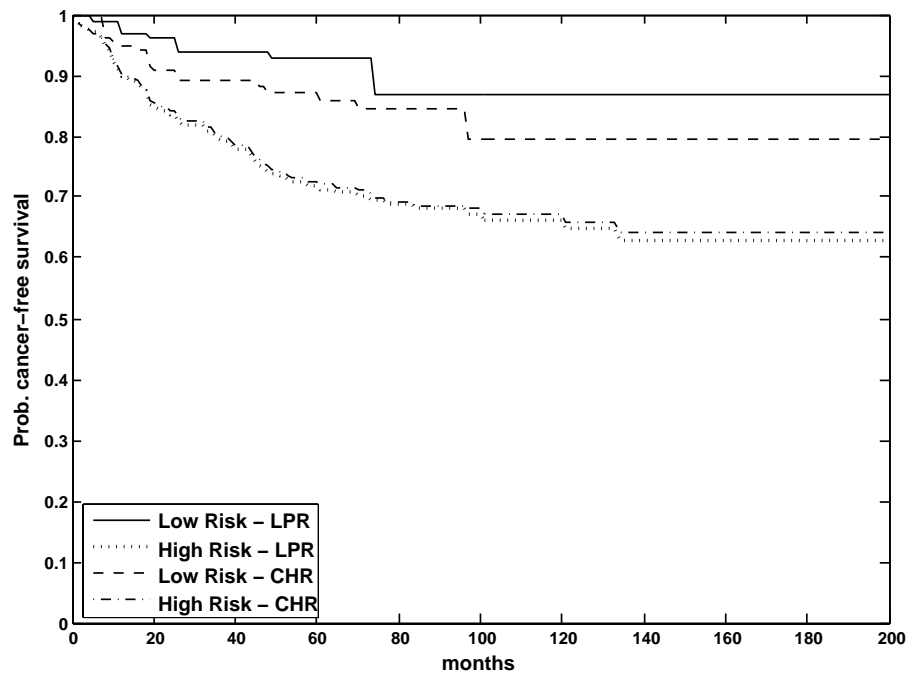

Figure 5.3: Kaplan-Meier curve for WPBC data: High-risk: Top 90%, Low-risk: Bottom 10%

For the low-risk group comparison we split the data from the bottom 10% for both algorithms. The Kaplan-Meier curves for those groups are in Figure 5.3. We observe that our ranking algorithm performs better on the low-risk group than Cox's regression model. Within five years we observe that about 95% of the subjects in the low-risk group from our

algorithm have no recurrence of the illness. For the Cox's regression model this percentage is about 87%. The Wilcoxon test shows that the two curves for both high and low-risk comparisons are significantly different. We note that our method does a much better job, relative to Cox's regression, at identifying a high-risk subgroup.
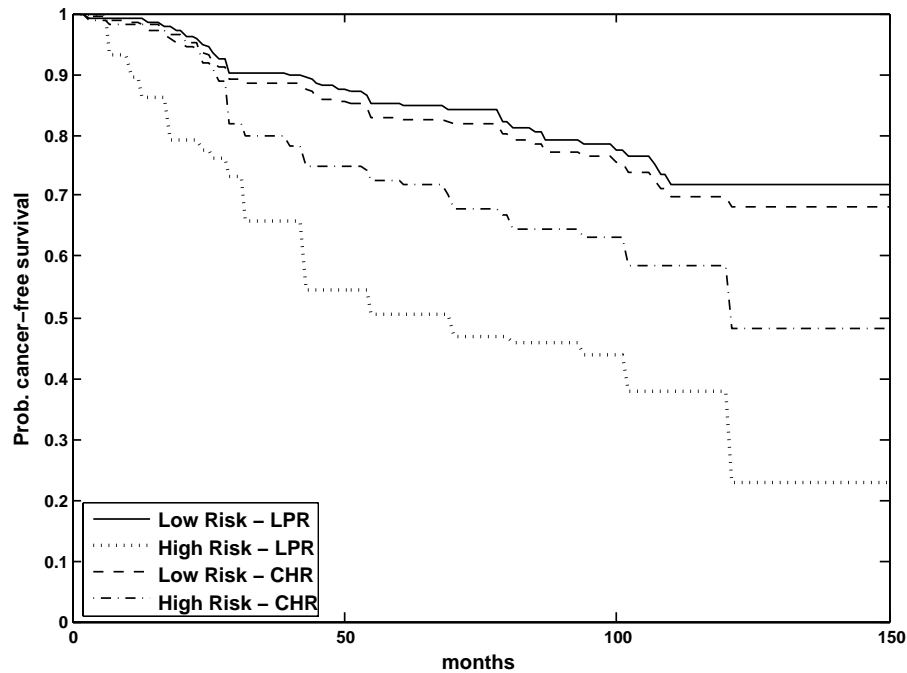

Figure 5.4: Kaplan-Meier curve for Seer data: High-risk: Top 10%, Low-risk: Bottom 90%

We carried out the identical procedure for the Seer data. Figure 5.4 shows that our approach is superior in predicting the high-risk group compared to Cox's regression. Again looking at a five-year horizon, about 50% of the high-risk group die, while the number is about 30% for the Cox's model. This means that our model is significantly better in ordering the top 10% of the patients based on their risk of cancer recurrence.

For the comparison of the low-risk groups composed of the bottom 10% presented in Figure 5.5, we observe that for a five-year horizon more then 95% of the low-risk group from our algorithm is still alive while for Cox's regression this number is about 90%. Overall, two curves are not significantly different when evaluated with the Wilcoxon test. Again,
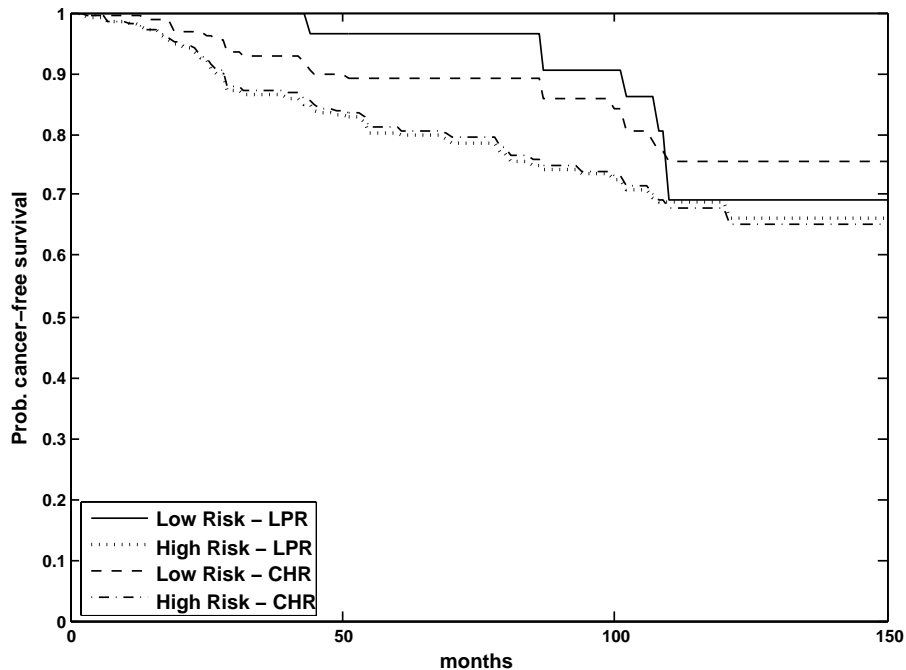
Figure 5.5: Kaplan-Meier curve for Seer data: High-risk: Top 90%, Low-risk: Bottom 10%

we see that the high-risk group is easier for our method to identify than the low-risk group.

Next, we compare the two algorithms on the burn patients data. In this problem the target is to predict the group of burn patients who have the highest risk of getting an infection again set as top 10%. Similar to the previous cases, if we set a time horizon of 30 days to predict infection potential the Kaplan-Meier curves in Figure 5.6 suggest that about 65% of the subjects in the high-risk group from our algorithm got infection while this number is about 35% for the Cox's regression model. A Wilcoxon test shows that our algorithm is significantly better in terms of predicting the burn patients who have a high risk of infection. Such a significant difference would benefit doctors in term of correctly targeting the patients that need more intensive care while potentially reducing expenses by limiting extensive care to a smaller group of people. We also observed that Cox's regression model performed poorly here as its high-risk group did not show a survival rate that was significantly different then the rest of the burn patients. We attribute this to the underlying model being non-linear. Table 5.1 shows that our ranking algorithm picked a higher γ value
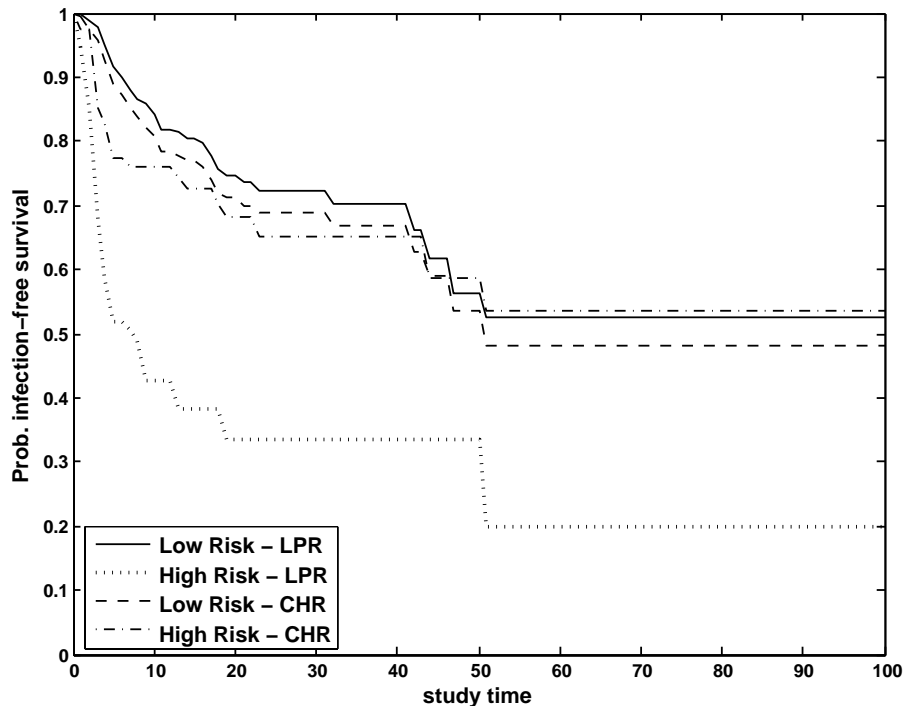
Figure 5.6: Kaplan-Meier Curve for Burn patients data: High-risk: Top 10%, Low-risk: Bottom 90%

which controls the non-linearity of the model supporting our reasoning.

The last dataset is from a business domain where the problem is detecting customers with a high potential of churning. For this problem we set the split of high-risk customers to be the top 10%. Figure 5.7 shows the comparison of Kaplan-Meier curves for two groups from both algorithms. If we consider a time horizon of 3 years (about 1000 days) we observe that about 50% of the customers in the high-risk group from our algorithm have terminated services while this number is 20% for the Cox's regression model. This would be a significant result for a company with millions of customers. With more accurate targeting it is possible to focus on those customers who posses high potential of leaving the company and try to keep them with additional promotions. This also saves cost on marketing campaigns by directly marketing to smaller segments.
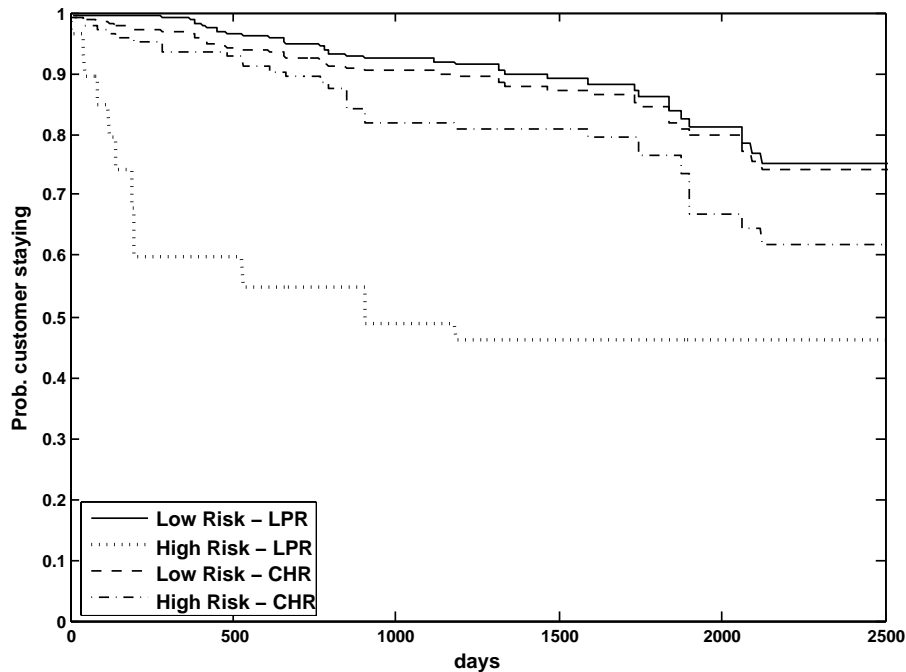
Figure 5.7: Kaplan-Meier Curve for Customer churn data: High-risk: Top 10%, Low-risk: Bottom 90%

5.5 Chapter Recap

Accurate prediction of survival times for individual subjects or sub-populations is a very important problem in many domains. Prediction of survival rates for cancer patients, survival of marriages, time to violation of parole, time to machine part failure, time to a customer defaulting a service, etc. are just a few examples. The common feature of such problems that separates them from others is the presence of censored data. Machine learning algorithms designed for classification or regression tasks are in general not appropriate for handling censored data.

In this chapter we introduced a modified version of LPR for survival analysis. Our algorithm is constructed to handle censored data and has great flexibility for allowing available domain knowledge to be incorporated to improve performance. Although the formulation is constructed as a linear program, our algorithm can handle non-linear problems by

utilizing kernel functions in the formulation. Our experiments demonstrate that our algorithm is superior to Cox's regression, the most commonly used survival analysis method. Here we used four real world problems from two domains, however it is not hard to see that our algorithm will perform well independent of domain or type of survival problem.

CHAPTER VI
CONCLUSIONS AND FUTURE DIRECTIONS

Typical machine learning algorithms are generally built to optimize predictive performance (usually measured in accuracy) by minimizing classification error. However, there are many real world problems where correct ordering of instances is of equal or greater importance than correct classification. Learning algorithms that are built to minimize classification error are often not effective when ordering within or among classes. This gap in research created a necessity to alter the objective of such algorithms to focus on correct ranking rather then classification.

In this work we present a linear programming formulation (LPR), similar to 1-norm SVM, for ranking instances with binary outputs by maximizing an approximation to the WMW statistic. Our formulation handles non-linear problems by making use of kernel functions. The results on several well-known benchmark datasets show that our approach ranks better than 2-norm SVM and faster than the SVR. We believe that a logical next step in this line of research would be to investigate extending the formulation for multi-class problems.

Optimization-based learning algorithms are usually not robust to an increasing number of data points. The solution times can increase dramatically with the number of data points making it infeasible to apply such algorithms to large or even moderately-sized problems. We introduced a number of exact and approximate heuristics to speed-up LPR. We used variations to the chunking approach to construct two exact heuristics. Although we were able to reduce the size of the problem, we were not able to consistently improve LP solving times by using these heuristics. Next, we introduced two approximation heuristics. The first heuristic is a hierarchical local clustering approach that reduces the size of the problem by systematically reducing the number of data points. The clustering approach reduced the number of constraints by 76% on average, while improving the solution times

by up to seven times with respect to LPR. The second heuristic is based on the subgradient method which approximates the LP formulation. The subgradient approximation was significantly faster then both LPR and LPR(clus) especially for larger datasets.

As a real world application of LPR with speed up heuristics, we used the CoIL Challenge 2000, caravan insurance dataset. We were able to solve the problem, which produced close to 1 million constraints without any memory limitations. The quality of the results were on par with the ones that are produced by the best algorithms from the CoIL challenge. We used a subgradient approximation to LPR and we were able to obtain good solutions as early as 20 iterations. Although the solution times heavily depend on the selected parameters, we were able to show that LPR is scalable when paired with the subgradient method proposed here.

Although not addressed in this work, we believe it is possible to combine the two approximate heuristic in order to solve even larger problems. Also, for larger problems we speculate that it would be more time-efficient to use ensemble methods. Each individual ranker in the ensemble maybe trained with a subset of training points such that every training point is used in at least one of the rankers in the ensemble. Then, orderings from each ranker can be aggregated to obtain a total ranking of all test points. We leave this as a future research direction.

Clustering in the higher-dimensional feature space is investigated in the literature yielding promising results [40]. As a promising future work, it is possible to perform clustering in the transformed space as a data reduction heuristic such as the one presented in Chapter 4. It is possible that an effective transformation to a higher dimensional feature space may yield a better reduction in the data by creating a better isolation of negative chunks from the positive ones.

Accurate prediction of survival times for individual subjects or sub-populations is a very important problem in many domains. The common feature of such problems that separates them from others is the presence of censored data. Machine learning algorithms

designed for classification or regression tasks are in general not appropriate for handling censored data. In Chapter 5, we introduced a modified version of LPR for survival analysis. Our algorithm is constructed to handle censored data and has great flexibility for allowing available domain knowledge to be incorporated to improve performance. Our experiments demonstrate that our algorithm is superior to Cox's proportional hazard regression, the most commonly used survival analysis method.

Ensemble approaches are highly parallelizable and are robust to any size problem. They have been applied in the context of survival analysis [50, 49] such as gradient boosting or random forests. In this work we left out the comparison of our approach to such an ensemble algorithm because we did not believe it would be fair to compare a single optimization based learner to an ensemble of learners. However, we see such comparisons of ensemble methods as a future work.

REFERENCES

[1] K. Ataman and W.N. Street. Approximation methods for LP-based rank optimization. Under review.

[2] K. Ataman and W.N. Street. Linear programming-based rank optimization for survival analysis. Under review.

[3] K. Ataman and W.N. Street. Hierarchical local clustering for constraint reduction in rank-optimizing linear programs. In *First INFORMS Workshop on Artificial Intelligence and Data Mining*, November 2006.

[4] K. Ataman, W.N. Street, and Y. Zhang. Learning to rank by maximizing AUC with linear programming. In *International Joint Conference on Neural Networks (IJCNN 2006)*, pages 123–129, 2006.

[5] J. Berkson and R.P. Gage. Calculation of survival rates for cancer. *Mayo Clinic Proceedings*, 25(11):270–86, 1950.

[6] C.L. Blake and C.J. Merz. UCI repository of machine learning databases. `http://mlearn.ics.uci.edu/`, 1998.

[7] A.P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145–1159, 1997.

[8] P. Bradley, O. Mangasarian, and D. Musicant. Optimization methods in massive datasets. In J. Abello, P.M. Pardalos, and M.G.C. Resende, editors, *Handbook of Massive Datasets*, pages 439–471. Kluwer Academic, 2001.

[9] P.S. Bradley and O.L. Mangasarian. Feature selection via concave minimization and support vector machines. In J. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)*, pages 82–90, San Francisco, California, 1998. Morgan Kaufmann.

[10] U. Brefeld and T. Scheffer. AUC maximizing support vector learning. In *Preceedings of ICML 2005 workshop on ROC Analysis in Machine Learning*, 2005.

[11] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 2003.

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

[13] C. L. Carter, C. Allen, and D. E. Henson. Relation of tumor size, lymph node status, and survival in 24,740 breast cancer cases. *Cancer*, 63:181–187, 1989.

[14] R. Caruana, S. Baluja, and T. Mitchell. Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 959–965. The MIT Press, 1996.

[15] R. Caruana and A. Niculescu-Mizil. An empirical evaluation of supervised learning for ROC area. *First Workshop of ROC Analysis in AI (ROCAI '04)*, 2004.

[16] W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning (ICML)*, pages 115–123, 1995.

[17] W.W. Cohen, R.E. Schapire, and Y. Singer. Learning to order things. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 10, pages 243–270. The MIT Press, 1998.

[18] C. Cortes and M. Mohri. AUC optimization vs. error rate minimization. In S. Thrun, L. Saul, and B. Scholkopf, editors, *Advances in Neural Information Processing Systems (NIPS 2003)*. MIT Press, Cambridge, MA, 2004.

[19] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[20] D.R. Cox. Regression models and life table. *Journal of the Royal Statistical Society*, (34):187–220, 1972.

[21] D.R. Cox and D. Oakes. *Analysis of Survival Data*. Chapman and Hall, London, UK.

[22] K. Crammer and Y. Singer. Pranking with ranking. In *Advances in Neural Information Processing Systems (NIPS)*, 2005.

[23] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[24] B.V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[25] G.E. Hinton D.E. Rumelhart and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing*, volume 1, Cambridge, MA, 1986. MIT Press.

[26] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. *Knowledge Discovery and Data Mining (KDD)*, pages 155–164, 1999.

[27] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple Bayesian classifier. *Machine Learning*, 29(2/3):103–130, 1997.

[28] J.P. Egan. *Signal Detection Theory and ROC Analysis*. Academic Press, 1975.

[29] T. Fawcett. ROC graphs: Notes and practical considerations for researchers. Technical report, HP Labs, 2003.

[30] J.E. Fieldsend and R.M. Everson. Formulation and comparison of multi-class ROC surfaces. *Proceedings of the 2nd ROCML Workshop, 22nd International Conference on Machine Learning (ICML 2005)*, pages 41–48, 2005.

[31] P. Flach and S. Wu. Repairing concavities in ROC curves. In *Proceedings of the UK Workshop on Computational Intelligence*, pages 38–44, 2003.

[32] P.A. Flach. The geometry of ROC space: Understanding machine learning metrics through ROC isometrics. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, pages 194–201, 2003.

[33] E. Frank and I. Witten. Generating accurate rule sets without global optimization. In *Proceedings 15th International Conference on Machine Learning*, pages 144–151, 1998.

[34] Y. Freund, R. Iyer, R.E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

[35] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the International Conference on Machine Learning*, pages 148–156, 1996.

[36] G. Fung and O.L. Mangasarian. Data selection for support vector machine classifiers. In R. Ramakrishnan and S. Stolfo, editors, *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 64–70. ACM, 2000.

[37] J. Furnkranz and G.Widmer. Incremental reduced error pruning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 70–77, 1994.

[38] J. Furnkranz and P.Flach. ROC *n*-rule learning towards a better understanding of covering algorithms. *Machine Learning*, pages 39–77, 2005.

[39] E.A. Gehan. A generalised Wilcoxon test for comparing arbitraryly singly censored samples. *Biometrika*, (52):203–223, 1965.

[40] M. Girolami. Mercer kernel-based clustering in feature space. *IEEE Transactions on Neural Networks*, 13(3):780–784, 2002.

[41] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub. Co., 1989.

[42] A. Goswami, R. Jin, and G. Agrawal. Fast and exact out-of-core k-means clustering. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, pages 83–90, 2004.

[43] D.M. Green and J.A. Swets. *Signal Detection Theory and Psychophysics*. Wiley, New York, 1966.

[44] D.J. Hand and R.J.Till. A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186, 2001.

[45] J.A. Hanley and B.J. McNeil. The meaning and the use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982.

[46] R. Herbrich, T. Graepel, P. Bollmann-Sdorra, and K. Obermayer. Learning a preference relation in IR. In *Proceedings of the Workshop on Text Categorization and Machine Learning, International Conference on Machine Learning*, pages 80–84, 1998.

[47] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, pages 115–132. The MIT Press, 2000.

[48] A. Herschtal and B. Raskutti. Optimising area under the ROC curve using gradient descent. In *Proceedings of the Twenty-First International Conference on Machine Learning*. ACM Press, 2004.

[49] T. Hothorn, P. Bühlmann, S. Dudoit, A. Molinaro, and M. J. van der Laan. Survival ensembles. *Biostatistics*, 7(3):355–373, Jul 2006.

[50] T. Hothorn, B. Lausen, A. Benner, and M. Radespiel-Tröger. Bagging survival trees. *Statistics in Medicine*, 23(1):77–91, Jan 2004.

[51] J. Huang, J. Lu, and C.X. Ling. Comparing naive Bayes, decision trees, and SVM with AUC and accuracy. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, 2003.

[52] J.M. Ichida, J.T. Wassell, M.D. Keller, and L.W. Ayers. Evaluation of protocol change in burn-care management using the cox proportional hazards model with time-dependent covariates. *Statistics in Medicine*, 12:301–310, 1993.

[53] ILOG. *CPLEX User Manual*. ILOG, 2002.

[54] The Mathworks Inc. *Matlab: The Language of Technical Computing*. The Mathworks Inc., 1997.

[55] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[56] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142. ACM Press, 2002.

[57] E.L. Kaplan and P. Meier. Nonparametric estimation from incomplete observations. *Journal of American Statistical Association*, (53):457–481, 1958.

[58] Y. Kim, W.N. Street, G.J. Russell, and F. Menczer. Customer targeting: A neural network approach guided by genetic algorithms. *Management Science*, 51(2):264–276, February 2005.

[59] E.T. Lee. *Statistical Methods for Survival Data Analysis*. Belmont, Wadsworth, 1980.

[60] X. Lin. A self-organizing semantic map for information retrieval. In *Proceedings of the 14th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR91)*, pages 262–269, 1991.

[61] C. Ling, J. Huang, and H. Zhang. AUC: A statistically consistent and more discriminating measure than accuracy. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2003.

[62] C. Ling and R. Yan. Decision tree with better ranking. *In Proceedings of the International Conference on Machine Learning (ICML2003), AAAI Press*, 2003.

[63] L. Lorena and E. Senne. Improving traditional subgradient scheme for Lagrangean relaxation: An application to location problems. *International Journal of Mathematical Algorithms*, 1:133–151, 1999.

[64] L.B. Lusted. Signal detectability and medical decision making. *Science*, 171:1217–1219, 1971.

[65] S.A. Macskassy, F. Provost, and S. Rosset. ROC confidence bands: An empirical evaluation. In *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)*, 2005.

[66] O.L. Mangasarian and D.R. Musicant. Large scale kernel regression via linear programming. *Machine Learning*, 46(1/3):255–269, 2002.

[67] O.L. Mangasarian, W.N. Street, and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(3):570–577, 1995.

[68] M.C. Mozer, R. Dodier, M.D. Colagrosso, C. Guerra-Salcedo, and R. Wolniewicz. Prodding the ROC curve: Constrained optimization of classifier performance. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pages 1409–1415, Cambridge, MA, 2002. MIT Press.

[69] A. Nedic and D. Bertsekas. Convergence rate of incremental subgradient algorithms. In S. Uryasev and P.M. Pardalos, editors, *Stochastic Optimization: Algorithms and Applications*, pages 263–304. Kluwer Academic Publishers, 2000.

[70] L. Ohno-Machado. A comparison of Cox proportional hazards and artificial neural network models for medical prognosis. *Computers in Biology and Medicine*, 27:55–65, 1997.

[71] L. Ohno-Machado and M.A. Musen. Modular neural networks for medical prognosis: Quantifying the benefits of combining neural networks for survival prediction. *Connection Science*, 9:71–96, 1997.

[72] J.C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning.*, pages 185–208. MIT Press, 1998.

[73] F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52:199–215, 2003.

[74] F. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the 15th International Conference on Machine Learning (ICML)*, pages 445–453. Morgan Kaufmann, San Francisco, CA, 1998.

[75] F.J. Provost and T. Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. *Knowledge Discovery and Data Mining*, pages 43–48, 1997.

[76] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

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

[78] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. In *KDD '05: Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*. ACM Press, 2005.

[79] A. Rakotomamonjy. Optimizing area under ROC curve with SVMs. In *Proceedings of the ROC Analysis in Artificial Intelligence*, pages 71–80, 2004.

[80] N.D. Ratliff, J.A. Bagnell, and M.A. Zinkevich. (Online) subgradient methods for structured prediction. In *Eleventh International Conference on Artificial Intelligence and Statistics (AIStats)*, March 2007.

[81] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 386–408, 1958.

[82] R.E. Schapire and Y. Singer. Improved boosting using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[83] B. Scholkopf and A. Smola. *Signal detection theory and ROC*. MIT Press, 2001.

[84] M. Sebag, J. Aze, and N. Lucas. ROC-based evolutionary learning: Application to medical data mining. *Artificial Evolution'03*, pages 384–396, 2004.

[85] N.Z. Shor. *Minimization Methods for Non-differentiable Functions*. Springer-Verlag, 1985.

[86] K.A. Spackman. Signal detection theory: Valuable tools for evaluating inductive learning. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 160–163. Morgan Kaufman, 1989.

[87] W.N. Street. A neural network model for prognostic prediction. In *Fifteenth International Conference on Machine Learning*, pages 540–546, 1998.

[88] W.N. Street, O.L. Mangasarian, and W.H. Wolberg. An inductive learning approach to prognostic prediction. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 522–530, San Francisco, 1995. Morgan Kaufmann.

[89] A.J. Swets. *Signal detection theory and ROC analysis in psychological diagnostics: Collected Papers*. Lawrence Erlbaum Publishers, 1996.

[90] P. van der Putten and M. van Someren. Coil challenge 2000: The insurance company case. Technical report, Leiden Institute of Advanced Computer Science, Leiden, 2000.

[91] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[92] C. Vogt and G. Cottrell. Using d' to optimize rankings. Technical Report CS98-601, U.C. San Diego, CSE Department, 1998.

[93] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.

[94] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000.

[95] L. Yan, R.H. Dodier, M. Mozer, and R.H. Wolniewicz. Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic. In *International Conference on Machine Learning*, pages 848–855, 2003.

[96] H. Yu. SVM selective sampling for ranking with application to data retrieval. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD 2005)*, 2005.

[97] H. Yu, J. Yang, J. Han, and X. Li. Making SVMs scalable to large data sets using hierarchical cluster indexing. *Data Mining and Knowledge Discovery*, 11(3):295–321, 2005.

[98] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-norm support vector machines. In *Advances in Neural Information Processing Systems*, volume 16, 2004.

[99] B. Zupan, J. Demsar, M.W. Kattan, J.R.Beck, and I. Bratko. Machine learning for survival analysis: A case study on recurrence of prostate cancer. *Artificial Intelligence in Medicine*, 20:59–75, Aug 2000.