From Ordinal Ranking to Binary Classification

Thesis by

Hsuan-Tien Lin

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2008

(Defended May 12, 2008)

Acknowledgements

First of all, I thank my advisor, Professor Yaser Abu-Mostafa, for his encouragement and guidance on my study, research, career, and life. It is truly a pleasure to be embraced by his wisdom and friendliness, and a privilege to work in the free and easy atmosphere that he creates for the learning systems group.

I also thank my thesis committee, Professors Yaser Abu-Mostafa, Jehoshua Bruck, Pietro Perona, and Christopher Umans, for reviewing the thesis and for stimulating many interesting research ideas during my presentations.

I have enjoyed numerous discussions with Dr. Ling Li and Dr. Amrit Pratap, my fellow members of the group. I thank them for their valuable input and feedback. I am particularly indebted to Dr. Ling Li, who not only collaborated in many projects with me throughout the years, but also gave me lots of valuable suggestions in research and in life. Furthermore, I want to thank Lucinda Acosta, who just makes everything in the group much simpler.

Special gratitude goes to Professor Chih-Jen Lin, who not only brought me into the fascinating area of machine learning eight years ago, but also continued to provide me very helpful suggestions in research and in career.

I thank Kai-Min Chung, Dr. John Langford, and the anonymous reviewers for their valuable comments on the earlier publications that led to this thesis. I am also grateful for the financial support from the Caltech Center for Neuromorphic Systems Engineering under the US NSF Cooperative Agreement EEC-9402726, the Caltech Bechtel Fellowship, and the Caltech Division of Engineering and Applied Science Fellowship.

Finally, I thank my friends, especially the members of the Association of Caltech

Taiwanese, for making my life colorful. Most importantly, I thank my family for their endless love and support. I want to express my deepest gratitude to my parents, who faithfully believe in me; to my brother, who selflessly shares everything with me; to my soulmate Yung-Han Yang, who passionately cherishes our long-distance relationship with me.

Abstract

We study the ordinal ranking problem in machine learning. The problem can be viewed as a classification problem with additional ordinal information or as a regression problem without actual numerical information. From the classification perspective, we formalize the concept of ordinal information by a cost-sensitive setup, and propose some novel cost-sensitive classification algorithms. The algorithms are derived from a systematic cost-transformation technique, which carries a strong theoretical guarantee. Experimental results show that the novel algorithms perform well both in a general cost-sensitive setup and in the specific ordinal ranking setup.

From the regression perspective, we propose the threshold ensemble model for ordinal ranking, which allows the machines to estimate a real-valued score (like regression) before quantizing it to an ordinal rank. We study the generalization ability of threshold ensembles and derive novel large-margin bounds on its expected test performance. In addition, we improve an existing algorithm and propose a novel algorithm for constructing large-margin threshold ensembles. Our proposed algorithms are efficient in training and achieve decent out-of-sample performance when compared with the state-of-the-art algorithm on benchmark data sets.

We then study how ordinal ranking can be reduced to weighted binary classification. The reduction framework is simpler than the cost-sensitive classification approach and includes the threshold ensemble model as a special case. The framework allows us to derive strong theoretical results that tightly connect ordinal ranking with binary classification. We demonstrate the algorithmic and theoretical use of the reduction framework by extending SVM and AdaBoost, two of the most popular binary classification algorithms, to the area of ordinal ranking. Coupling SVM with the

reduction framework results in a novel and faster algorithm for ordinal ranking with superior performance on real-world data sets, as well as a new bound on the expected test performance for generalized linear ordinal rankers. Coupling AdaBoost with the reduction framework leads to a novel algorithm that boosts the training accuracy of any cost-sensitive ordinal ranking algorithms theoretically, and in turn improves their test performance empirically.

From the studies above, the key to improve ordinal ranking is to improve binary classification. In the final part of the thesis, we include two projects that aim at understanding binary classification better in the context of ensemble learning. First, we discuss how AdaBoost is restricted to combining only a finite number of hypotheses and remove the restriction by formulating a framework of infinite ensemble learning based on SVM. The framework can output an infinite ensemble through embedding infinitely many hypotheses into an SVM kernel. Using the framework, we show that binary classification (and hence ordinal ranking) can be improved by going from a finite ensemble to an infinite one. Second, we discuss how AdaBoost carries the property of being resistant to overfitting. Then, we propose the SeedBoost algorithm, which uses the property as a machinery to prevent other learning algorithms from overfitting. Empirical results demonstrate that SeedBoost can indeed improve an overfitting algorithm on some data sets.

Contents

List of Figures

List of Tables

List of Selected Algorithms

Chapter 1

Introduction

Machine learning, the study that allows computational systems to adaptively improve their performance with experience accumulated from the data observed, is becoming a major tool in many fields. Furthermore, the growing application needs in the Internet age keep supplementing machine learning research with new types of problems. This thesis is about one of them—the ordinal ranking problem. It belongs to a family of learning problems, called *supervised learning*, which will be introduced below.

1.1 Supervised Learning

In the supervised learning problems, the machine is given a *training set* $Z = \{z_n\}_{n=1}^{N}$, which contains *training examples* $z_n = (\mathbf{x}_n, y_n)$. We assume that each *feature vector* $\mathbf{x}_n \in \mathcal{X} \subseteq \mathbb{R}^D$, each *label* $y_n \in \mathcal{Y}$, and each training example z_n is drawn independently from an unknown probability measure $dF(\mathbf{x}, y)$ on $\mathcal{X} \times \mathcal{Y}$. We focus on the case where $dF(y \,|\, \mathbf{x})$, the random process that generates y from $\mathbf{x}$, is governed by

$$y = g_*(\mathbf{x}) + \epsilon_{\mathbf{x}} \; .$$

Here $g_* \colon \mathcal{X} \to \mathcal{Y}$ is a deterministic but unknown component called the *target function*, which denotes the *best* function that can predict y from $\mathbf{x}$. The exact notion of "best" varies by application needs and will be formally defined later in this section. The other

part of y, which cannot be perfectly explained by $g_*(\mathbf{x})$, is represented by a random component $\epsilon_{\mathbf{x}}$.

With the given training set, the machine should return a *decision function* $\hat{g}$ as the inference of the target function. The decision function is chosen from a *learning model* $\mathcal{G} = \{g\}$, which is a collection of candidate functions $g\colon \mathcal{X} \to \mathcal{Y}$. Briefly speaking, the task of supervised learning is to use the information in the training set Z to find some decision function $\hat{g} \in \mathcal{G}$ that is almost as good as g_* under $dF(\mathbf{x}, y)$.

For instance, we may want to build a recognition system that transforms an image of a written digit to its intended meaning. We can first ask someone to write down N digits and represent their images by the feature vectors $\mathbf{x}_n$. We then label the images by $y_n \in \{0, 1, \ldots, 9\}$ according to their meanings. The target function g_* here encodes the process of our human-based recognition system and $\epsilon_{\mathbf{x}}$ represents the mistakes we may make in our brain. The task of this learning problem is to set up an automatic recognition system (decision function) $\hat{g}$ that is almost as good as our own recognition system, even on the yet unseen images of written digits in the future.

The machine conquers the task with a *learning algorithm* $\mathcal{A}$. Generally speaking, the algorithm takes the learning model $\mathcal{G}$ and the training set Z as inputs. It then returns a decision function $\hat{g} \in \mathcal{G}$ by minimizing a predefined objective function $E(g, Z)$ over $g \in \mathcal{G}$. The full scenario of learning is illustrated in Figure 1.1.

Let us take one step back and look at what we mean by g_* being the "best" function to predict y from $\mathbf{x}$. To evaluate the predicting ability of any $g\colon \mathcal{X} \to \mathcal{Y}$, we define its *out-of-sample cost*

$$\pi(g, F) = \int_{\mathbf{x},y} C\big(y, g(\mathbf{x})\big)\, dF(\mathbf{x}, y)\,.$$

Here $C(y, k)$ is called the *cost function*, which quantifies the price to be paid when an example of label y is predicted as k. The value of $\pi(g, F)$ reflects the expected test cost on the (mostly) unseen examples drawn from $dF(\mathbf{x}, y)$. Then, the "best"

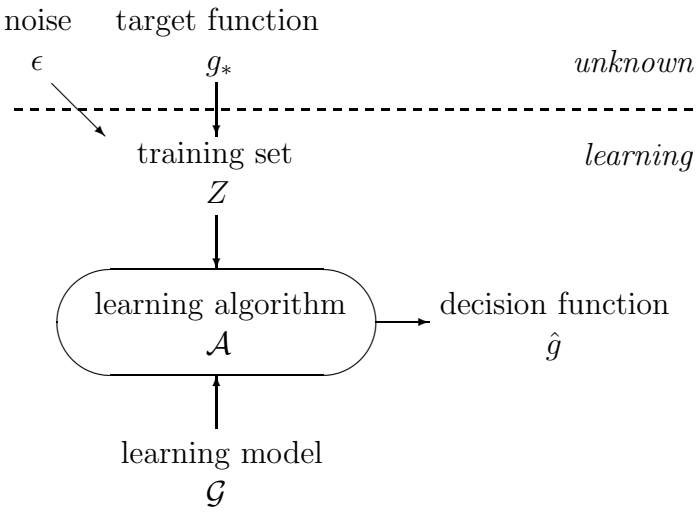


Figure 1.1: Illustration of the learning scenario

.

function g_* should satisfy

$$\pi(g_*, F) \leq \pi(g, F) \quad \forall g \colon \mathcal{X} \to \mathcal{Y} \, .$$

One of such a g_* can be defined by

$$g_*(\mathbf{x}) \equiv \underset{k \in \mathcal{Y}}{\operatorname{argmin}} \left(\int_y C\big(y, k\big) \; dF(y \,|\, \mathbf{x}) \right). \tag{1.1}$$

In this thesis, we assume that such a g_* exists with ties in argmin arbitrarily broken, and denote $\pi(g, F)$ by $\pi(g)$ when F is clear from the context.

Recall that the task of supervised learning is to find some $\hat{g} \in \mathcal{G}$ that is almost as good as g_* under $dF(\mathbf{x}, y)$. Since $\pi(g_*)$ is the lower bound, we desire $\pi(\hat{g})$ to be as small as possible. Note that $\mathcal{A}$ minimizes $E(g, Z)$ to get $\hat{g}$, and hence ideally we want to set $E(g, Z) = \pi(g)$. Nevertheless, because $dF(\mathbf{x}, y)$ is unknown, it is not possible to compute such an $E(g, Z)$ nor to minimize it directly. A substitute quantity that

depends only on Z is called the *in-sample cost*

$$\nu(g) = \sum_{i=1}^{N} C\big(y_n, g(\mathbf{x}_n)\big) \cdot \frac{1}{N} \,.$$

Note that $\nu(g)$ can also be defined by $\pi(g, Z_u)$ where Z_u denotes a uniform distribution over the training set Z. Because $\nu(g)$ is an unbiased estimate of $\pi(g)$ for any given single g, many learning algorithms take $\nu(g)$ as a major component of $E(g, Z)$.

A small $\nu(g)$, however, does not always imply a small $\pi(g)$ (Abu-Mostafa 1989; Vapnik 1995). When the decision function $\hat{g}$ comes with a small $\nu(\hat{g})$ and a large $\pi(\hat{g})$, we say that $\hat{g}$ (or the learning algorithm $\mathcal{A}$) *overfits* the training set Z. For instance, consider a training set Z with $\mathbf{x}_n \neq \mathbf{x}_m$ for all $n \neq m$, and $C\,(y, k) = |y - k|$. Assume that a

$$\hat{g}(\mathbf{x}) = \begin{cases} y_n, & \text{for } \mathbf{x} \in \{\mathbf{x}_n\}_{n=1}^{N}\,; \\ \text{some constant } \Delta, & \text{otherwise.} \end{cases}$$

Then, we see that $\nu(\hat{g}) = 0$ (the smallest possible value) and $\pi(\hat{g})$ can be as large as we want by varying the constant. That is, there exists a decision function like $\hat{g}$ that leads to serious overfitting. Preventing overfitting is one of the most important objective when designing learning models and algorithms. Generally speaking, the objective can be achieved when the complexity of $\mathcal{G}$ (and hence the chosen $\hat{g}$) is reasonably controlled (Abu-Mostafa 1989; Abu-Mostafa et al. 2004; Vapnik 1995).

One important type of supervised learning problem is (univariate) *regression*, which deals with the case when $\mathcal{Y}$ is a metric space isometric to $\mathbb{R}$. For simplicity, we shall restrict ourselves to the case where $\mathcal{Y} = \mathbb{R}$. Although not strictly required, common regression algorithms usually not only work on some $\mathcal{G}$ that contains continuous functions, but also desires $\hat{g}$ to be reasonably smooth as a control of its complexity (Hastie, Tibshirani and Friedman 2001). The metric information is thus important in determining the smoothness of the function.

For instance, a widely used cost function for regression is $C_s(y, k) = (y - k)^2$, the

squared cost. With the cost in mind, the ridge regression algorithm (Hastie, Tibshirani and Friedman 2001) works on a linear regression model

$$\mathcal{G} = \{g_{\mathbf{v},b} \colon g_{\mathbf{v},b}(\mathbf{x}) = \langle \mathbf{v}, \mathbf{x} \rangle + b\} \,,$$

with $\hat{g}$ being the optimal solution of

$$\begin{aligned} \min_{g \in \mathcal{G}} \quad & E(g, Z) \,, \\ \text{where} \quad & E(g_{\mathbf{v},b}, Z) = \frac{\lambda}{2} \langle \mathbf{v}, \mathbf{v} \rangle + \frac{1}{N} \sum_{n=1}^{N} C_s\Big(y_n, g_{\mathbf{v},b}(\mathbf{x}_n)\Big). \end{aligned}$$

The first part of $E(g, Z)$ controls the smoothness of the decision function chosen, and the second part is $\nu(g_{\mathbf{v},b})$.

Another important type of supervised learning problem is called *classification*, in which $\mathcal{Y}$ is a finite set $\mathcal{Y}_c = \{1, 2, \ldots, K\}$. Each label in $\mathcal{Y}_c$ represents a different category. For instance, the digit recognition system described earlier can be formulated as a classification problem. A function of the form $\mathcal{X} \to \mathcal{Y}_c$ is called a *classifier*. In the special case where $|\mathcal{Y}_c| = 2$, the classification problem is called *binary classification*, in which the classifier g is called a *binary classifier*.

To evaluate whether a classifier predicts the desired category correctly, a commonly used cost function is the *classification cost* $C_c(y, k) = [\![y \neq k]\!]$.[1] In some classification problems, however, it may be desired to treat different kinds of classification mistakes differently (Margineantu 2001). For instance, when designing a system to classify cells as $\{\text{cancerous}, \text{noncancerous}\}$, in terms of the possible loss on human life, the cost of classifying some cancerous cell as a noncancerous one should be significantly higher than the other way around. These classification problems would thus include cost functions other than C_c. We call them *cost-sensitive* classification problems to distinguish them from the *regular* classification problems, which uses only C_c.

[1] $[\![\cdot]\!] = 1$ when the inner condition is true, and 0 otherwise.

Note that in classification problems, for any given $(\mathbf{x}, y)$, the cost function C would be evaluated only on parameters (y, k) where $k \in \mathcal{Y}_c$. We can then represent the needed part of C by a *cost vector* $\mathbf{c}$ with respect to y, where $\mathbf{c}[k] = C(y, k)$. In this thesis, we take a more general setup of cost-sensitive classification and allow different cost functions to be used on different examples (Abe, Zadrozny and Langford 2004). In the setup, we assume that the vector $\mathbf{c}$ is drawn from some probability measure $dF(\mathbf{c} \,|\, \mathbf{x}, y)$ on a collection $\mathcal{C}$ of possible cost functions. We call the tuple $(\mathbf{x}, y, \mathbf{c})$ the *cost-sensitive example* to distinguish it from a regular example $(\mathbf{x}, y)$. The learning algorithm now receives a cost-sensitive training set $\{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^N$ to work with.[2] Using the cost-sensitive examples $(\mathbf{x}, y, \mathbf{c})$, the out-of-sample cost becomes

$$\pi(g) \;=\; \int_{\mathbf{x},y} \left(\int_{\mathbf{c}} \mathbf{c}[g(\mathbf{x})] \; dF(\mathbf{c} \,|\, \mathbf{x}, y) \right) \; dF(\mathbf{x}, y) \;\; ,$$

and the in-sample cost becomes

$$\nu(g) = \sum_{i=1}^{N} \mathbf{c}_n[g(\mathbf{x}_n)] \cdot \frac{1}{N} \; .$$

As can be seen from the updated definitions of $\pi(g)$ and $\nu(g)$, our setup do not explicitly need the label y. We shall, however, keep the notation for clarity and assume that $y = \underset{k \in \mathcal{Y}}{\operatorname{argmin}} \; \mathbf{c}[k]$.

A special instance of cost-sensitive classification takes the cost vector $\mathbf{c}$ to be of the form $\mathbf{c}[k] = w \cdot C_c(y, k)$ for every cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$ with some $w \geq 0$. We call the instance *weighted classification*, in which a cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$ can be simplified to a *weighted example* $(\mathbf{x}, y, w)$. It is known that weighted classification problems can be readily solved by regular classification algorithms with rejection-based sampling (Zadrozny, Langford and Abe 2003).

[2] In many applications, the exact form of $dF(\mathbf{c} \,|\, \mathbf{x}, y)$ is known by application needs. In all of our theoretical results, however, $dF(\mathbf{c} \,|\, \mathbf{x}, y)$ can be either known or unknown, as long as the learning algorithm receives a cost-sensitive training set where $\mathbf{c}_n$ is drawn independently from $dF(\mathbf{c} \,|\, \mathbf{x}_n, y_n)$.

1.2 Ordinal Ranking

Ordinal ranking is another type of supervised learning problem. It is similar to classification in the sense that $\mathcal{Y}$ is a finite set $\mathcal{Y}_r = \{1, 2, \ldots, K\} = \mathcal{Y}_c$. Therefore, ordinal ranking is also called *ordinal classification* (Cardoso and da Costa 2007; Frank and Hall 2001). Nevertheless, in addition to representing the nominal categories (as the usual classification labels), now those $y \in \mathcal{Y}_r$ also carry the ordinal information. That is, two different labels in $\mathcal{Y}_r$ can be *compared* by the usual "$<$" operation. We call those y the *ranks* to distinguish them from the usual classification labels. We use a *ranker* $r(\mathbf{x})$ to denote a function from $\mathcal{X} \to \mathcal{Y}_r$. In an ordinal ranking problem, the decision function is denoted by $\hat{r}(\mathbf{x})$, and the target function is denoted by $r_*(\mathbf{x})$.

Because ranks can be naturally used to represent human preferences, ordinal ranking lends itself to many applications in social science, psychology, and information retrieval. For instance, we may want to build a recommendation system that predicts how much a user likes a movie. We can first choose N movies and represent each movie by a feature vector $\mathbf{x}_n$. We then ask the user to (see and) rate each movie by {one star, two star, ..., five star}, depending on how much she or he likes the movie. The set $\mathcal{Y}_r = \{1, 2, \ldots, 5\}$ includes different levels of preference (numbers of stars), which are ordered by "$<$" to represent "worse than." The task of this learning problem is to set up an automatic recommendation system (decision function) $\hat{r} \colon \mathcal{X} \to \mathcal{Y}_r$ that is almost as good as the user, even on the yet unseen movies in the future.

Ordinal ranking is also similar to regression, in the sense that ordinal information is similarly encoded in $y \in \mathbb{R}$. Therefore, ordinal ranking is also popularly called *ordinal regression* (Chu and Ghahramani 2005; Chu and Keerthi 2007; Herbrich, Graepel and Obermayer 2000; Li and Lin 2007b; Lin and Li 2006; Shashua and Levin 2003; Xia, Tao, et al. 2007; Xia, Zhou, et al. 2007). Nevertheless, unlike the real-valued regression labels, the discrete ranks $y \in \mathcal{Y}_r$ do not carry metric information. For instance, we cannot say that a five-star movie is 2.5 times better than a two-star one. In other words, the rank serves as a qualitative indication rather than a quantitative outcome. The lack of metric information violates the assumption of

many regression algorithms, and hence they may not perform well on ordinal ranking problems.

The ordinal information carried by the ranks introduce the following two properties, which are important for modeling ordinal ranking problems.

- **Closeness in the rank space $\mathcal{Y}_r$**: The ordinal information suggests that the mislabeling cost depend on the "closeness" of the prediction. For example, predicting a two-star movie as a three-star one is less costly than predicting it as a five-star one. Hence, the cost vector $\mathbf{c}$ should be *V-shaped* with respect to y (Li and Lin 2007b), that is,

$$\begin{cases} \mathbf{c}[k-1] \geq \mathbf{c}[k]\,, & \text{for } 2 \leq k \leq y\,; \\ \mathbf{c}[k+1] \geq \mathbf{c}[k]\,, & \text{for } y \leq k \leq K-1. \end{cases} \tag{1.2}$$

  Briefly speaking, a V-shaped cost vector says that a ranker needs to pay more if its prediction on $\mathbf{x}$ is further away from y. We shall assume that every cost vector $\mathbf{c}$ generated from $dF(\mathbf{c}\,|\,\mathbf{x}, y)$ is V-shaped with respect to $y = \operatorname*{argmin}_{1 \leq k \leq K} \mathbf{c}[k]$. With this assumption, ordinal ranking can be casted as a cost-sensitive classification problem with V-shaped cost vectors.

  In some of our results, we need a stronger condition: The cost vectors should be *convex* (Li and Lin 2007b), that is,

$$\mathbf{c}[k+1] - \mathbf{c}[k] \geq \mathbf{c}[k] - \mathbf{c}[k-1]\,, \text{ for } 2 \leq k \leq K-1\,. \tag{1.3}$$

  When using convex cost vectors, a ranker needs to pay *increasingly* more if its prediction on $\mathbf{x}$ is further away from y. It is not hard to see that any convex cost vector $\mathbf{c}$ is V-shaped with respect to $y = \operatorname*{argmin}_{1 \leq k \leq K} \mathbf{c}[k]$.

- **Structure in the feature space $\mathcal{X}$**: Note that the classification cost vectors $\left\{\mathbf{c}_c^{(\ell)} \colon \mathbf{c}_c^{(\ell)}[k] = [\![\ell \neq k]\!]\right\}_{\ell=1}^{K}$, which are associated with the classification cost function C_c, are also V-shaped. If those cost vectors (and hence C_c) are used,

what distinguishes ordinal ranking and regular classification?

Note that the total order within $\mathcal{Y}_r$ and the target function r_* introduces a total preorder in $\mathcal{X}$ (Herbrich, Graepel and Obermayer 2000). That is,

$$\mathbf{x} \lesssim \mathbf{x}' \Longleftrightarrow r_*(\mathbf{x}) \leq r_*(\mathbf{x}').$$

The total preorder allows us to naturally group and compare vectors in the feature space $\mathcal{X}$. For instance, a two-star movie is "worse than" a three-star one, which is in term "worse than" a four-star one; movies of less than three stars are "worse than" movies of at least three stars.

It is the *meaningfulness* of the grouping and the comparison that distinguishes ordinal ranking from regular classification, even when the classification cost vectors $\left\{\mathbf{c}_c^{(\ell)}\right\}_{\ell=1}^{K}$ are used. For instance, if apple = 1, banana = 2, grape = 3, orange = 4, strawberry = 5, we can intuitively see that comparing fruits $\{1, 2\}$ with fruits $\{3, 4, 5\}$ is not as meaningful as comparing "movies of less than three stars" with "movies of at least three stars."

Ordinal ranking has been studied from the statistics perspective in detail by McCullagh (1980), who viewed ranks as contiguous intervals on an unobservable real-valued random variable. From the machine learning perspective, many ordinal ranking algorithms have been proposed in recent years. For instance, Herbrich, Graepel and Obermayer (2000) followed the view of McCullagh (1980) and designed an algorithm with support vector machines (Vapnik 1995). One key idea of Herbrich, Graepel and Obermayer (2000) is to compare training examples by their ranks in a pairwise manner. Har-Peled, Roth and Zimak (2003) proposed a constraint classification algorithm that also compares training examples in a pairwise manner. Another instance of the pairwise comparison approach is the RankBoost algorithm (Freund et al. 2003; Lin and Li 2006), which will be further described in Chapter 3. Nevertheless, because there are $O(N^2)$ pairwise comparisons out of N training examples, it is harder to apply those algorithms on large-scale ordinal ranking problems.

There are some other algorithms that do not lead to such a quadratic expansion, such as perceptron ranking (Crammer and Singer 2005, PRank), ordinal regression boosting (Lin and Li 2006, ORBoost, which will be further introduced in Chapter 3), support vector ordinal regression (Chu and Keerthi 2007, SVOR), and the data replication method (Cardoso and da Costa 2007). As we shall see later in Chapter 4, these algorithms can be unified under a simple reduction framework (Li and Lin 2007b).

Still some other algorithms fall into neither of the approaches above, such as C4.5-ORD (Frank and Hall 2001), Gaussian process ordinal regression (Chu and Ghahramani 2005, GPOR), recursive feature extraction (Xia, Tao, et al. 2007), and Weighted-LogitBoost (Xia, Zhou, et al. 2007).

1.3 Overview

In Chapter 2, we study the ordinal ranking problem from a classification perspective. That is, we cast ordinal ranking as a cost-sensitive classification problem with V-shaped costs. We propose the cost-transformation technique to systematically extend regular classification algorithms to their cost-sensitive versions. The technique carries strong theoretical guarantees. Based on the technique, we derive two novel cost-sensitive classification algorithms based on their popular versions in regular classification and test their performance on both cost-sensitive classification and ordinal ranking problems.

In Chapter 3, we study the ordinal ranking problem from a regression perspective. That is, we solve ordinal ranking by thresholding an estimation of a latent continuous variable. Learning models associated with this approach are called threshold models. We propose an novel instance of the threshold model called the threshold ensemble model and prove its theoretical properties. We not only extend RankBoost for constructing threshold ensemble rankers, but also propose a more efficient algorithm called ORBoost. The proposed algorithm roots from the famous adaptive boosting (AdaBoost) approach and carries promising properties from its ancestor.

In Chapter 4, we show that ordinal ranking can be reduced to binary classification

theoretically and algorithmically, which includes the threshold model as a special case. We derive theoretical foundations of the reduction framework and demonstrate a surprising equivalence: Ordinal ranking is as hard (easy) as binary classification. In addition to extending support vector machines (SVM) to ordinal ranking with the reduction framework, we also propose a novel algorithm called AdaBoost.OR, which efficiently constructs an ensemble of ordinal rankers as its decision function.

In Chapter 5, we include two concrete research projects that aim at understanding and improving binary classification. The results can in turn be coupled with the reduction framework to improve ordinal ranking. First, we propose a novel framework of infinite ensemble learning based on SVM. The framework is not limited by the finiteness restriction of existing ensemble learning algorithms. Using the framework, we show that binary classification (and hence ordinal ranking) can be improved by going from a finite ensemble to an infinite one. Second, we discuss how AdaBoost carries the property of being resistant to overfitting. We then propose the SeedBoost algorithm, which uses the property as a machinery to prevent other learning algorithms from overfitting.

Some of the results in Chapters 3, 4, and 5 were jointly developed by Dr. Ling Li and the author (Li and Lin 2007b; Lin and Li 2006, 2008). The results that should be credited to Dr. Ling Li will be properly acknowledged in the coming chapters. The results without such acknowledgment are the original contributions of the author.

Chapter 2

Ordinal Ranking by Cost-Sensitive Classification

As discussed in Section 1.2, ordinal ranking can be casted as a cost-sensitive classification problem with V-shaped cost vectors. In this chapter, we study the cost-sensitive classification problem in general and propose a systematic technique to transform it to a regular classification problem. We first derive the theoretical foundations of the technique. Then, we use the technique to extend two popular algorithms for regular classification, namely one-versus-one and one-versus-all, to their cost-sensitive versions. We empirically demonstrate the usefulness of the new cost-sensitive algorithms on general cost-sensitive classification problems as well as on ordinal ranking problems.

2.1 Cost-Sensitive Classification

Cost-sensitive classification fits the needs of many practical applications of machine learning and data mining, such as targeted marketing, fraud detection, and medical decision systems (Abe, Zadrozny and Langford 2004). Margineantu (2001) discussed three kinds of approaches for solving the problem: manipulating the training examples, modifying the learning algorithm, or manipulating the decision function. There is also the fourth kind: designing a new learning algorithm to solve the problem directly.

For manipulating the training examples, Domingos (1999) proposed the MetaCost algorithm, which takes any classification algorithm to estimate $dF(y \mid \mathbf{x})$, uses the estimate to relabel the training examples, and then retrains a classifier with the relabeled examples. The algorithm, however, depends strongly on how well $dF(y \mid \mathbf{x})$ is estimated, which is hard to be theoretically guaranteed. In addition, the algorithm needs to know the cost collection $\mathcal{C}$ in advance and only accepts some restricted forms of $dF(\mathbf{c} \mid \mathbf{x}, y)$. These shortcomings make it difficult to use the algorithm on our more general cost-sensitive setup. Approaches that manipulate the decision function suffer from similar shortcomings (Abe, Zadrozny and Langford 2004; Margineantu 2001).

There are many cost-sensitive classification approaches that come from modifying some regular classification algorithm (see, for instance, Margineantu 2001, Subsection 2.3.2). These approaches are usually constructed by identifying where the classification cost vectors are used in $E(g, Z)$ (or some intermediate quantity within $\mathcal{A}$), and then replacing them with cost-sensitive ones. Nevertheless, the modifications are usually ad hoc and heuristic based. In other words, those approaches usually do not come with a strong theoretical guarantee, either.

Recently, some authors proposed new algorithms for solving cost-sensitive classification problems directly (Beygelzimer et al. 2005; Beygelzimer, Langford and Ravikumar 2007; Langford and Beygelzimer 2005). These algorithms come with stronger theoretical guarantee, but because of their novelty, they have not been as widely tested nor as successful as some popular algorithms for regular classification.

Our work takes the route of modifying existing algorithms, along with the objective of being systematic as well as providing a strong theoretical guarantee. In addition, our proposed modifications are based on the *cost-transformation technique*, which is related to manipulating training examples in a principled manner. Then, we can easily extend successful algorithms for regular classification to their cost-sensitive versions. Next, we illustrate the cost-transformation technique.

2.2 Cost-Transformation Technique

The key of the cost-transformation technique is to decompose a cost vector $\mathbf{c}$ to a conic combination of the *classification cost vectors* $\left\{\mathbf{c}_c^{(\ell)}\right\}_{\ell=1}^{K}$, where

$$\mathbf{c}_c^{(\ell)}[k] = C_c(\ell, k) = [\![\ell \neq k]\!] .$$

For instance, consider a cost vector $\tilde{\mathbf{c}} = (4, 3, 2, 3)$, we see that

$$\tilde{\mathbf{c}} = 0 \cdot \underbrace{(0,1,1,1)}_{\mathbf{c}_c^{(1)}} + 1 \cdot \underbrace{(1,0,1,1)}_{\mathbf{c}_c^{(2)}} + 2 \cdot \underbrace{(1,1,0,1)}_{\mathbf{c}_c^{(3)}} + 1 \cdot \underbrace{(1,1,1,0)}_{\mathbf{c}_c^{(4)}} .$$

Why is such a decomposition useful? If there is a cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$, where $\mathbf{c} = \sum_{\ell=1}^{K} \tilde{\mathbf{q}}[\ell] \cdot \mathbf{c}_c^{(\ell)}$, then for any classifier g,

$$\mathbf{c}[g(\mathbf{x})] = \sum_{\ell=1}^{K} \tilde{\mathbf{q}}[\ell] \cdot \mathbf{c}_c^{(\ell)}[g(\mathbf{x})] = \sum_{\ell=1}^{K} \tilde{\mathbf{q}}[\ell] \cdot [\![\ell \neq g(\mathbf{x})]\!] .$$

That is, if we sample ℓ proportional to $\tilde{\mathbf{q}}[\ell]$ and replace the cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$ by a regular one $(\mathbf{x}, \ell)$, then the cost that g needs to pay for its prediction on $\mathbf{x}$ is proportional to the expected classification cost. Thus, if a classifier g performs well on the "relabeled" problem using the expected classification cost, it would also perform well on the original cost-sensitive problem. The nonnegativity of $\tilde{\mathbf{q}}[\ell]$ ensures that $\tilde{\mathbf{q}}$ can be scaled to form a probability distribution $dF(\ell \,|\, \tilde{\mathbf{q}})$.[1]

Nevertheless, can every cost vector $\mathbf{c}$ be decomposed to a conic combination of $\left\{\mathbf{c}_c^{(\ell)}\right\}_{\ell=1}^{K}$? The short answer is no. For instance, the cost vector $\mathbf{c} = (2, 1, 0, 1)$ cannot be decomposed to any conic combination of $\left\{\mathbf{c}_c^{(\ell)}\right\}_{\ell=1}^{4}$, because $\mathbf{c}$ comes with a unique linear decomposition:

$$(2,1,0,1) = -\frac{2}{3} \cdot (0,1,1,1) + \frac{1}{3} \cdot (1,0,1,1) + \frac{4}{3} \cdot (1,1,0,1) + \frac{1}{3} \cdot (1,1,1,0) .$$

[1] We take a minor assumption that not all $\tilde{\mathbf{q}}[\ell]$ are zero. Otherwise $\mathbf{c} = \mathbf{0}$ and the example $(\mathbf{x}, y, \mathbf{c})$ can be simply dropped.

Thus, $\mathbf{c}$ cannot be represented by any conic combination of $\left\{\mathbf{c}_c^{(\ell)}\right\}_{\ell=1}^{4}$. The unique existence of a linear combination is formalized in the following lemma.

Lemma 2.1. *Any* $\mathbf{c} \in \mathbb{R}^K$ *can be uniquely decomposed to* $\mathbf{c} = \sum_{\ell=1}^{K} \mathbf{q}[\ell] \cdot \mathbf{c}_c^{(\ell)}$, *where* $\mathbf{q}[\ell] \in \mathbb{R}$ *for* $\ell = 1, 2, \ldots, K$.

Proof. Note that $\mathbf{q}[\ell]$ needs to satisfy the following matrix equation:

$$\underbrace{\begin{pmatrix} \mathbf{c}[1] \\ \mathbf{c}[2] \\ \cdots \\ \mathbf{c}[K] \end{pmatrix}}_{\mathbf{c}^T} = \underbrace{\begin{pmatrix} 0 & 1 & 1 & \ldots & 1 \\ 1 & 0 & 1 & \ldots & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & 1 & 1 & \ldots & 0 \end{pmatrix}}_{\mathbf{M}} \underbrace{\begin{pmatrix} \mathbf{q}[1] \\ \mathbf{q}[2] \\ \cdots \\ \mathbf{q}[K] \end{pmatrix}}_{\mathbf{q}^T} .$$

Because $\mathbf{M}$ is invertible with

$$\mathbf{M}^{-1} = \frac{1}{K-1} \begin{pmatrix} -(K-2) & 1 & 1 & \ldots & 1 \\ 1 & -(K-2) & 1 & \ldots & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & 1 & 1 & \ldots & -(K-2) \end{pmatrix},$$

the vector $\mathbf{q}$ can be uniquely computed by $\left(\mathbf{M}^{-1}\mathbf{c}^T\right)^T$. That is,

$$\mathbf{q}[\ell] = \left(\frac{1}{K-1}\sum_{k=1}^{K} \mathbf{c}[k]\right) - \mathbf{c}[\ell]\,. \qquad \square$$

Although $\tilde{\mathbf{c}} = (4, 3, 2, 3)$ yields a conic decomposition but $\mathbf{c} = (2, 1, 0, 1)$ does not, the two cost vectors are not very different when being used to evaluate the performance of a classifier g. Note that for every $\mathbf{x}$, $\tilde{\mathbf{c}}[g(\mathbf{x})] = \mathbf{c}[g(\mathbf{x})] + 2$. The constant shifting from $\mathbf{c}$ to $\tilde{\mathbf{c}}$ does not affect the relative cost difference between the prediction $g(\mathbf{x})$ and the best prediction y. That is, using $\tilde{\mathbf{c}}$ is equivalent to using $\mathbf{c}$ plus a constant cost of 2 on every example. We call a cost vector $\tilde{\mathbf{c}}$ *similar* to $\mathbf{c}$ by Δ

when

$$\tilde{\mathbf{c}}[\cdot] = \mathbf{c}[\cdot] + \Delta\,,$$

with some constant Δ. If we only use $\mathbf{c}$ additively, as what we did in the definition of $\pi(g)$ and $\nu(g)$, using $\tilde{\mathbf{c}}$ instead of $\mathbf{c}$ would introduce only a constant shift.

Although we cannot decompose any $\mathbf{c}$ to a conic combination of $\left\{\mathbf{c}_c^{(\ell)}\right\}_{\ell=1}^{K}$, there exists infinitely many cost vectors $\tilde{\mathbf{c}}$ that allow a conic combination while being similar to $\mathbf{c}$. To see this, note that

$$(K-1)\cdot(\Delta,\Delta,\ldots,\Delta) = \Delta\sum_{\ell=1}^{K}\mathbf{c}_c^{(\ell)}.$$

Then, consider $\mathbf{c} = \sum_{\ell=1}^{K}\mathbf{q}[\ell]\cdot\mathbf{c}_c^{(\ell)}$,

$$\tilde{\mathbf{c}} = \mathbf{c} + (K-1)\cdot(\Delta,\Delta,\ldots,\Delta) = \sum_{\ell=1}^{K}(\mathbf{q}[\ell]+\Delta)\cdot\mathbf{c}_c^{(\ell)}.$$

We can easily make $\tilde{\mathbf{q}}[\ell] = \mathbf{q}[\ell] + \Delta \geq 0$ by choosing $\Delta \geq \max\limits_{1\leq\ell\leq K}(-\mathbf{q}[\ell])$.

Lemma 2.2. *Consider some* $\mathbf{c} = \sum_{\ell=1}^{K}\mathbf{q}[\ell]\cdot\mathbf{c}_c^{(\ell)}$. *If* $\tilde{\mathbf{c}}$ *is similar to* $\mathbf{c}$ *by* $(K-1)\cdot\Delta$, *then* $\tilde{\mathbf{c}}$ *yields a conic combination of* $\left\{\mathbf{c}_c^{(\ell)}\right\}_{\ell=1}^{K}$ *if and only if* $\Delta \geq \max\limits_{1\leq\ell\leq K}(-\mathbf{q}[\ell])$.

Proof. By Lemma 2.1, the decomposition of $\tilde{\mathbf{c}}$ by

$$\sum_{\ell=1}^{K}(\mathbf{q}[\ell]+\Delta)\cdot\mathbf{c}_c^{(\ell)}$$

is unique. Then, it is not hard to see that $\tilde{\mathbf{q}}[\ell] = \mathbf{q}[\ell]+\Delta \geq 0$ for every $\ell = 1, 2, \ldots, K$ if and only if $\Delta \geq \max\limits_{1\leq\ell\leq K}(-\mathbf{q}[\ell])$. □

Thus, we can first transform each cost vector $\mathbf{c}$ to a similar one $\tilde{\mathbf{c}}$ that yields a conic combination, get the vector $\tilde{\mathbf{q}}$, and randomly relabel $(\mathbf{x}, y, \mathbf{c})$ to $(\mathbf{x}, \ell)$ with probability $dF(\ell \,|\, \mathbf{c})$ proportional to $\tilde{\mathbf{q}}[\ell]$. The procedure above transforms the original cost-sensitive classification problem to an equivalent regular classification one. From

Lemma 2.2, there are infinitely many $\tilde{\mathbf{c}}$ that we can use. The next question is, which is more preferable? Since the proposed procedure relabels with probability

$$dF(\ell \,|\, \mathbf{c}) = \tilde{\mathbf{p}}[\ell] = \frac{\tilde{\mathbf{q}}[\ell]}{\sum_{k=1}^{K} \tilde{\mathbf{q}}[k]} \,,$$

we would naturally desire the discrete probability distribution $\tilde{\mathbf{p}}[\cdot]$ to be of the least entropy. That is, we want the distribution to come from the optimal solution of

$$\begin{aligned}
\min_{\tilde{\mathbf{p}},\Delta} \quad & \sum_{\ell=1}^{K} \tilde{\mathbf{p}}[\ell] \log \frac{1}{\tilde{\mathbf{p}}[\ell]} \,, && (2.1)\\
\text{subject to} \quad \Delta \;&\geq\; \max_{1\leq \ell \leq K} \left(-\mathbf{q}[\ell]\right) \,, \\
\tilde{\mathbf{p}}[\ell] \;&=\; \frac{\tilde{\mathbf{q}}[\ell]}{\sum_{k=1}^{K} \tilde{\mathbf{q}}[k]} \,, && \ell = 1, 2, \ldots, K, \\
\tilde{\mathbf{q}}[\ell] \;&=\; \mathbf{q}[\ell] + \Delta \,, && \ell = 1, 2, \ldots, K, \\
\mathbf{q}[\ell] \;&=\; \left(\frac{1}{K-1} \sum_{k=1}^{K} \mathbf{c}[k] \right) - \mathbf{c}[\ell] \;, && \ell = 1, 2, \ldots, K.
\end{aligned}$$

Theorem 2.3. *If not all* $\mathbf{c}[\ell]$ *are equal, the unique optimal solution to* (2.1) *is*

$$\tilde{\mathbf{p}}[\ell] = \frac{c_{\max} - \mathbf{c}[\ell]}{\sum_{k=1}^{K} \left(c_{\max} - \mathbf{c}[k]\right)} \text{ and } \Delta = \max_{1\leq \ell \leq K} \left(-\mathbf{q}[\ell]\right), \text{ where } c_{\max} = \max_{1\leq \ell \leq K} \mathbf{c}[\ell] \,.$$

Proof. If not all $\mathbf{c}[\ell]$ are equal, not all $\mathbf{q}[\ell]$ are equal. Now we substitute those $\tilde{\mathbf{p}}$ in the objective function by the right-hand sides of the equality constraints. Then, the objective function becomes

$$f(\Delta) \;=\; -\sum_{\ell=1}^{K} \frac{\mathbf{q}[\ell] + \Delta}{\sum_{k=1}^{K} \mathbf{q}[k] + K\Delta} \log \frac{\mathbf{q}[\ell] + \Delta}{\sum_{k=1}^{K} \mathbf{q}[k] + K\Delta} \,.$$

The constraint on Δ ensures that all the $p \log p$ operations above are well defined.[2]

[2] We take the convention that $0 \log 0 \equiv \lim_{\epsilon \to 0} \epsilon \log \epsilon = 0$.

Now, let $\bar{q} \equiv \frac{1}{K}\sum_{k=1}^{K}\mathbf{q}[k]$. We get

$$\begin{aligned}
\frac{df}{d\Delta} &= -\frac{1}{K\left(\bar{q}+\Delta\right)^2}\sum_{\ell=1}^{K}\left(-\mathbf{q}[\ell]+\bar{q}\right)\cdot\left(\log\left(\frac{\mathbf{q}[\ell]+\Delta}{\bar{q}+\Delta}\right)-\log K+1\right)\\
&= -\frac{1}{K\left(\bar{q}+\Delta\right)^2}\sum_{\ell=1}^{K}\left(-\underbrace{\left(\mathbf{q}[\ell]+\Delta\right)}_{a_\ell}+\underbrace{\left(\bar{q}+\Delta\right)}_{b_\ell}\right)\cdot\log\frac{\mathbf{q}[\ell]+\Delta}{\bar{q}+\Delta}\\
&= \frac{1}{K\left(\bar{q}+\Delta\right)^2}\sum_{\ell=1}^{K}\big(a_\ell-b_\ell\big)\cdot\big(\log a_\ell-\log b_\ell\big). \qquad \square
\end{aligned}$$

When not all $\mathbf{q}[\ell]$ are equal, there exists at least one a_ℓ that is not equal to b_ℓ. Therefore, $\frac{df}{d\Delta}$ is strictly positive, and hence the unique minimum of $f(\Delta)$ happens when Δ is of the smallest possible value. That is, for the unique optimal solution,

$$\begin{cases}
\Delta = \max\limits_{1\le\ell\le K}\left(-\mathbf{q}[\ell]\right) = c_{\max}-\left(\frac{1}{K-1}\sum_{k=1}^{K}\mathbf{c}[k]\right)\ ;\\
\tilde{\mathbf{q}}[\ell] = c_{\max}-\mathbf{c}[\ell]\,,\ \tilde{\mathbf{p}}[\ell] = \frac{c_{\max}-\mathbf{c}[\ell]}{\sum_{k=1}^{K}(c_{\max}-\mathbf{c}[k])}.
\end{cases} \tag{2.2}$$

Using Theorem 2.3, we can define the following probability measure $dF_c(\mathbf{x},\ell)$ from $dF(\mathbf{x},y,\mathbf{c})$:

$$dF_c(\mathbf{x},\ell) \propto \int_{y,\mathbf{c}} \tilde{\mathbf{q}}[\ell]\ dF(\mathbf{x},y,\mathbf{c})\,,$$

where $\tilde{\mathbf{q}}[\ell]$ is computed from $\mathbf{c}$ using (2.2).[3] More precisely, let

$$\Lambda_1 = \int_{x,y,\mathbf{c}}\sum_{\ell=1}^{K}\tilde{\mathbf{q}}[\ell]\ dF(\mathbf{x},y,\mathbf{c})\,. \tag{2.3}$$

Note that from (2.2), $\sum_{\ell=1}^{K}\tilde{\mathbf{q}}[\ell] > 0$ if not all $\mathbf{c}[\ell]$ are equal. Thus, we can generally

[3]Even when all $\mathbf{c}[\ell]$ are equal, (2.2) can still be used to get $\tilde{\mathbf{q}}[\ell]=0$ for all ℓ, which means the example $(\mathbf{x},y,\mathbf{c})$ can be dropped instead of relabeled.

assume that the integral results in a nonzero value. That is, $\Lambda_1 > 0$, and

$$dF_c(\mathbf{x}, \ell) = \Lambda_1^{-1} \cdot \int_{y,\mathbf{c}} \tilde{\mathbf{q}}[\ell]\ dF(\mathbf{x}, y, \mathbf{c})\,.$$

Then, we can derive the following cost-transformation theorem:

Theorem 2.4. *For any classifier g,*

$$\pi(g, F) = \Lambda_1 \cdot \pi(g, F_c) - \Lambda_2\,,$$

where $\Lambda_2 = (K-1) \cdot \int_{\mathbf{x},y,\mathbf{c}} \Delta \cdot dF(\mathbf{x}, y, \mathbf{c})$ and each Δ in the integral is computed from $\mathbf{c}$ with (2.2)*.*

Proof.

$$\begin{aligned}
\pi(g, F) &= \int_{\mathbf{x}} \left(\int_{y,\mathbf{c}} \mathbf{c}[g(\mathbf{x})]\ dF(y, \mathbf{c} \,|\, \mathbf{x}) \right)\ dF(\mathbf{x}) \\
&= \int_{\mathbf{x}} \left(\int_{y,\mathbf{c}} \sum_{\ell=1}^{K} \mathbf{q}[\ell] \cdot \mathbf{c}_c^{(\ell)}[g(\mathbf{x})]\ dF(y, \mathbf{c} \,|\, \mathbf{x}) \right)\ dF(\mathbf{x}) \\
&= \int_{\mathbf{x}} \left(\int_{y,\mathbf{c}} \sum_{\ell=1}^{K} (\tilde{\mathbf{q}}[\ell] - \Delta) \cdot \mathbf{c}_c^{(\ell)}[g(\mathbf{x})]\ dF(y, \mathbf{c} \,|\, \mathbf{x}) \right)\ dF(\mathbf{x}) \\
&= -\Lambda_2 + \int_{\mathbf{x}} \left(\int_{y,\mathbf{c}} \sum_{\ell=1}^{K} \tilde{\mathbf{q}}[\ell] \cdot \mathbf{c}_c^{(\ell)}[g(\mathbf{x})]\ dF(y, \mathbf{c} \,|\, \mathbf{x}) \right)\ dF(\mathbf{x}) \\
&= -\Lambda_2 + \int_{\mathbf{x}} \sum_{\ell=1}^{K} \mathbf{c}_c^{(\ell)}[g(\mathbf{x})] \cdot \left(\int_{y,\mathbf{c}} \tilde{\mathbf{q}}[\ell]\ dF(y, \mathbf{c} \,|\, \mathbf{x}) \right)\ dF(\mathbf{x}) \\
&= -\Lambda_2 + \Lambda_1 \cdot \int_{\mathbf{x},\ell} \mathbf{c}_c^{(\ell)}[g(\mathbf{x})] \cdot dF_c(\mathbf{x}, \ell) \\
&= -\Lambda_2 + \Lambda_1 \cdot \pi(g, F_c).
\end{aligned}$$

□

An immediate corollary of Theorem 2.4 is:

Corollary 2.5. *If g_* is the target function under $dF(\mathbf{x}, y, \mathbf{c})$, and $\tilde{g}_*$ is the target function under $dF_c(\mathbf{x}, \ell)$, then $\pi(g_*, F) = \pi(\tilde{g}_*, F)$ and $\pi(g_*, F_c) = \pi(\tilde{g}_*, F_c)$.*

That is, if a regular classification algorithm $\mathcal{A}_c$ is able to return a decision func-

tion $\hat{g} = \tilde{g}_*$, the decision function is as good as the target function g_* under the original $dF(\mathbf{x}, y, \mathbf{c})$. Furthermore, as formalized in the following regret transformation theorem, if any classifier g is close to $\tilde{g}_*$ under $dF_c(\mathbf{x}, \ell)$, it is also close to g_* under $dF(\mathbf{x}, y, \mathbf{c})$.

Theorem 2.6. *Consider $dF_c(\mathbf{x}, \ell)$ defined from $dF(\mathbf{x}, y, \mathbf{c})$ above, for any classifier g,*

$$\pi(g, F) - \pi(g_*, F) = \Lambda_1 \cdot \Big(\pi(g, F_c) - \pi(\tilde{g}_*, F_c)\Big).$$

Thus, to deal with a cost-sensitive classification problem generated from $dF(\mathbf{x}, y, \mathbf{c})$, it seems that the learning algorithm $\mathcal{A}$ can take the following steps:

Algorithm 2.1 (Cost transformation with relabeling, preliminary).

1. *Compute $dF_c(\mathbf{x}, \ell)$ and obtain N independent training examples $Z_c = \{(\mathbf{x}_n, \ell_n)\}_{n=1}^N$ from $dF_c(\mathbf{x}, \ell)$.*

2. *Use a regular classification algorithm $\mathcal{A}_c$ on Z_c to obtain a decision function $\hat{g}_c$ that ideally yields a small $\pi(\hat{g}_c, F_c)$.*

3. *Return $\hat{g} \equiv \hat{g}_c$.*

There is, however, a caveat in the algorithm above. Recall that $dF(\mathbf{x}, y, \mathbf{c})$ is unknown, and $dF_c(\mathbf{x}, \ell)$ depends on $dF(\mathbf{x}, y, \mathbf{c})$. Thus, $dF_c(\mathbf{x}, \ell)$ cannot be actually computed. Nevertheless, we know that the training set $Z = \{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^N$ contains examples that are generated independently from $dF(\mathbf{x}, y, \mathbf{c})$. Then, the first step can be implemented (almost equivalently) as follows.

Algorithm 2.2 (Cost transformation with relabeling).

1. *Obtain N' independent training examples $Z_c = \{(\mathbf{x}_n, \ell_n)\}_{n=1}^{N'}$ from $dF_c(\mathbf{x}, \ell)$:*

   (a) *Transform each $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ to $(\mathbf{x}_n, \tilde{\mathbf{q}}_n)$ by (2.2).*

   (b) *Apply the rejection-based sampling technique (Zadrozny, Langford and Abe 2003) and accept $(\mathbf{x}_n, \tilde{\mathbf{q}}_n)$ with probability proportional to $\sum_{\ell=1}^K \tilde{\mathbf{q}}_n[\ell]$.*

(c) *For those* $(\mathbf{x}_n, \tilde{\mathbf{q}}_n)$ *that survive from rejection-based sampling, randomly assign its label* ℓ_n *with probability* $\tilde{\mathbf{p}}_n[\ell] \propto \tilde{\mathbf{q}}_n[\ell]$.

2. *Use a regular classification algorithm* $\mathcal{A}_c$ *on* Z_c *to obtain a decision function* $\hat{g}_c$ *that ideally yields a small* $\pi(\hat{g}_c, F_c)$.

3. *Return* $\hat{g} \equiv \hat{g}_c$.

It is easy to check that the new training set Z_c contains N' (usually less than N) independent examples from $dF_c(\mathbf{x}, \ell)$.

While the steps above are supported with theoretical guarantees from Theorems 2.4 and 2.6, they may not work well in practice. For instance, if we look at an example $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ with $y_n = 1$ and $\mathbf{c}_n = (0, 1, 1, 334)$, the resulting $\tilde{\mathbf{q}}_n = (334, 333, 333, 0)$. Because of the large value in $\mathbf{c}_n[4]$, the example looks almost like a uniform mixture of labels $\{1, 2, 3\}$, with only 0.334 of probability to keep its original label. In other words, for the purpose of encoding some large components in a cost vector, the relabeling process could pay a huge variance and relabel (or mislabel) the example more often than not. Then, the regular classification algorithm $\mathcal{A}_c$ would receive some Z_c that contains lots of misleading labels, making it hard for the algorithm to return a decent $\hat{g}_c$.

One remedy to the difficulty above is to use the following algorithm, called *training set expansion and weighting* (TSEW), instead of relabeling:

Algorithm 2.3 (TSEW: training set expansion and weighting).

1. *Obtain* NK *weighted training examples* $Z_w = \{(\mathbf{x}_{n\ell}, y_{n\ell}, w_{n\ell})\}$:

   (a) *Transform each* $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ *to* $(\mathbf{x}_n, \tilde{\mathbf{q}}_n)$ *by* (2.2).

   (b) *For every* $1 \le \ell \le K$, *let* $(\mathbf{x}_{n\ell}, y_{n\ell}, w_{n\ell}) = (\mathbf{x}_n, \ell, \tilde{\mathbf{q}}_n[\ell])$ *and add* $(\mathbf{x}_{n\ell}, y_{n\ell}, w_{n\ell})$ *to* Z_w.

2. *Use a weighted classification algorithm* $\mathcal{A}_w$ *on* Z_w *to obtain a decision function* $\hat{g}_w$.

3. *Return* $\hat{g} \equiv \hat{g}_w$.

It is not hard to show that $dF_c(\mathbf{x}, \ell) \propto w \cdot dF_w(\mathbf{x}, \ell, w)$, and Z_w contains (dependent) examples generated from $dF_w(\mathbf{x}, \ell, w)$. We can think of Z_w, which trades independence for smaller variance, as a more stable version of Z_c. The expanded training set Z_w contains all possible ℓ, and hence always includes the correct label y_n (along with the largest weight on $\tilde{\mathbf{q}}_n[y_n]$). The $\mathcal{A}_w$ in TSEW can also be performed by a regular classification algorithm $\mathcal{A}_c$ using the rejection-based sampling technique (Zadrozny, Langford and Abe 2003). Then, Algorithm 2.2 is simply a special (and less-stable) case of TSEW.

The TSEW algorithm is a basic instance of our proposed cost-transformation technique. It is the same as the *data space expansion* (DSE) algorithm (Abe, Zadrozny and Langford 2004). Nevertheless, our derivation from the minimum entropy perspective is novel, and our theoretical results on the out-of-sample cost $\pi(g)$ are more general than the in-sample cost analysis by Abe, Zadrozny and Langford (2004). Recently, Xia, Zhou, et al. (2007) also proposed an algorithm similar to TSEW using LogitBoost as $\mathcal{A}_w$ based on a restricted version of Theorem 2.4. It should be noted that the results discussed in this section are partially influenced by the work of Abe, Zadrozny and Langford (2004) but are independent from the work of Xia, Zhou, et al. (2007).

From the experimental results, TSEW (DSE) does not perform well in practice (Abe, Zadrozny and Langford 2004). A possible reason is that common $\mathcal{A}_w$ still find Z_w too difficult (Xia, Zhou, et al. 2007), because a training feature vector $\mathbf{x}_n$ could be multilabeled in Z_w, which may confuse $\mathcal{A}_w$. One could improve the basic TSEW algorithm by using (or designing) an $\mathcal{A}_w$ that is more robust with multilabeled training feature vectors, as discussed in the next section.

2.3 Algorithms

In this section, we propose two novel cost-sensitive classification algorithms by coupling the cost-transformation technique with popular algorithms for regular (weighted) classification.

2.3.1 Cost-Sensitive One-Versus-All

The *one-versus-all* (OVA) algorithm is a popular algorithm for weighted classification. It solves the weighted classification problem by decomposing it to several weighted binary classification problems, as shown below.

Algorithm 2.4 (One-versus-all, see, for instance, Hsu and Lin 2002).

1. *For each* $1 \leq \ell \leq K$,
   (a) *Take the original training set* $Z = \{(\mathbf{x}_n, y_n, w_n)\}_{n=1}^N$ *and construct a binary classification training set* $Z_b^{(\ell)} = \left\{(\mathbf{x}_n, y_n^{(\ell)}, w_n) \colon y_n^{(\ell)} = [\![y_n = \ell]\!]\right\}_{n=1}^N$.
   (b) *Use a weighted binary classification algorithm* $\mathcal{A}_b$ *on* $Z_b^{(\ell)}$ *to get a decision function* $\hat{g}_b^{(\ell)}$.
2. *Return* $\hat{g}(\mathbf{x}) = \underset{1 \leq \ell \leq K}{\operatorname{argmax}}\, \hat{g}_b^{(\ell)}(\mathbf{x})$.

Each $\hat{g}_b^{(\ell)}(\mathbf{x})$ intends to predict whether $\mathbf{x}$ belongs to category ℓ. Thus, if a feature vector $\mathbf{x}$ should be of category 1, and all $\hat{g}_b^{(\ell)}$ are mistake free, then ideally $\hat{g}_b^{(1)}(\mathbf{x}) = 1$ and $\hat{g}_b^{(\ell)}(\mathbf{x}) = 0$ for $\ell \neq 1$, and hence $\hat{g}(\mathbf{x})$ could make a correct prediction. Nevertheless, if some of the binary decision functions $\hat{g}_b^{(\ell)}$ make mistakes, the performance of OVA can be affected by the ties in the argmax operation. In practice, the OVA algorithm usually allows the decision functions $\hat{g}_b^{(\ell)}$ to output a soft prediction (say, in $[0, 1]$) rather than a hard one of $\{0, 1\}$. The soft prediction represents the support (confidence) on whether $\mathbf{x}$ belongs to category ℓ, and $\hat{g}(\mathbf{x})$ returns the prediction associated with the highest support.

What would happen if we directly use OVA as $\mathcal{A}_w$ in TSEW? Recall that a cost-sensitive training example $\big(\mathbf{x}_1, 1, (0, 1, 1, 334)\big)$ in Z would introduce the following multilabeled examples in Z_w:

$$(\mathbf{x}_1, 1, 334), (\mathbf{x}_1, 2, 333), (\mathbf{x}_1, 3, 333).$$

If we feed Z_w directly to OVA, the underlying binary classification algorithm $\mathcal{A}_b$ would use the following examples to get $\hat{g}_b^{(1)}$:

$$(\mathbf{x}_1, 1, 334), (\mathbf{x}_1, 0, 666).$$

That is, even though $\mathbf{x}_1$ is of category 1, paradoxically we prefer $\mathcal{A}_b$ to return some $\hat{g}_b^{(1)}$ that predicts $\mathbf{x}_1$ as 0 rather than 1. The paradox is similar to what we encountered when sampling Z_c from Z in Algorithm 2.1: The relabeling process results in a misleading label more often than not. Thus, directly plugging OVA into the TSEW algorithm does not work.

Nevertheless, we can modify OVA and make it more robust when given multi-labeled training examples. In fact, a variant of the OVA algorithm can readily be used for multilabeled classification in literature (see, for instance, Joachims 2005, Section 2). For a training feature vector $\mathbf{x}_n$ that can be labeled either as 1 or 2, the OVA algorithm for a multilabeled training set Z would pair $\mathbf{x}_n$ with $y_n^{(1)} = 1$ when constructing $Z_b^{(1)}$, and with $y_n^{(2)} = 1$ when constructing $Z_b^{(2)}$ as well. That is, when Z contains both (and only) $(\mathbf{x}_n, 1)$ and $(\mathbf{x}_n, 2)$, the feature vector $\mathbf{x}_n$ "supports" both category 1 and 2, while it does not support categories 3, 4, ..., K.

The support perspective can also be understood with the cost vectors and the cost-transformation technique. Note that the expanded training set Z_w contains both $(\mathbf{x}_n, 1)$ and $(\mathbf{x}_n, 2)$ (with weights 1) if and only if the original cost-sensitive training example $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ comes with a cost vector $\mathbf{c}_n$ that is similar to $(0, 0, 1, 1, \ldots, 1)$. Thus, $\mathbf{x}_n$ supports both categories 1 and 2 because no cost needs to be paid when the prediction falls in them. Note that $\tilde{\mathbf{q}}_n = (1, 1, 0, 0, \ldots, 0)$ in this case. Equivalently speaking, we can say that $\mathbf{x}_n$ supports those categories ℓ with $\tilde{\mathbf{q}}_n[\ell] = 1$ and does not support those ℓ with $\tilde{\mathbf{q}}_n[\ell] = 0$.

From the observation above, we can define $\mathbf{s}[\ell] = \frac{\tilde{\mathbf{q}}[\ell]}{\tilde{q}_{\max}}$ as the support for category ℓ, where $\tilde{q}_{\max} = \max_{1 \le \ell \le K} \tilde{\mathbf{q}}[\ell]$. Thus, $\mathbf{s}[\ell] \in [0, 1]$, and from (2.2),

$$\begin{cases} \mathbf{s}[\ell] = 0 & \text{when } \mathbf{c}[\ell] = c_{\max}, \\ \mathbf{s}[\ell] = 1 & \text{when } \mathbf{c}[\ell] = c_{\min} = \min_{1 \le k \le K} \mathbf{c}[k]\,. \end{cases}$$

With the definition above, we propose the generalized OVA algorithm, which takes the original OVA algorithm as a special case.

Algorithm 2.5 (Generalized one-versus-all).

1. *For each $1 \le \ell \le K$, use $\mathcal{A}_b$ to learn a binary classifier $\hat{g}_b^{(\ell)}(\mathbf{x})$ with the hope that*

$$\int_{\mathbf{x},y,\mathbf{c}} \tilde{q}_{\max} \cdot \left(\mathbf{s}[\ell] - \hat{g}_b^{(\ell)}(\mathbf{x})\right)^2 \, dF(\mathbf{x}, y, \mathbf{c}) \tag{2.4}$$

   is small.

2. *Return $\hat{g}(\mathbf{x}) = \underset{1 \le \ell \le K}{argmax}\ \hat{g}_b^{(\ell)}(\mathbf{x})$.*

How can $\mathcal{A}_b$ learn a binary classifier? Equation (2.4) is deliberately formulated as a learning problem. Then, for each training example $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ obtained from $dF(\mathbf{x}, y, \mathbf{c})$, we can compute a new training example $(\mathbf{x}_n, y_n, \mathbf{s}_n[\ell]\,, (\tilde{q}_{\max})_n)$ to provide information for solving such a learning problem. Assume that we keep the convention $\mathbf{x}_n^{(\ell)} = \mathbf{x}_n$ and $y_n^{(\ell)} = [\![y_n = \ell]\!]$ in Algorithm 2.4 and try to approximately deal with (2.4) by casting it as a weighted binary classification problem. One simple method to obtain the weight $w_n^{(\ell)}$ from $\left(\mathbf{x}_n, y_n, \mathbf{s}_n[\ell]\,, (\tilde{q}_{\max})_n\right)$ is[4]

$$w_n^{(\ell)} = \begin{cases} (\tilde{q}_{\max})_n \cdot \mathbf{s}_n[\ell] & \text{when } y_n = \ell\ ; \\ (\tilde{q}_{\max})_n \cdot (1 - \mathbf{s}_n[\ell]) & \text{when } y_n \neq \ell\ . \end{cases} \tag{2.5}$$

[4]There are more than one methods to do the transformation, and it is not theoretically clear which of them should be preferred. We choose the simple approach in (2.5) because of its promising performance in practice.

By replacing step 1 of generalized OVA with the weighted binary classification problem, we get the *cost-sensitive one-versus-all* (CSOVA) algorithm, as formalized below.

Algorithm 2.6 (CSOVA: Cost-sensitive one-versus-all).

1. *For each* $1 \leq \ell \leq K$,

   (a) *Take the original training set* $Z = \{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^N$ *and construct a binary classification training set* $Z_b^{(\ell)} = \left\{(\mathbf{x}_n, y_n^{(\ell)}, w_n^{(\ell)})\right\}$ *from* (2.5).

   (b) *Use a weighted binary classification algorithm* $\mathcal{A}_b$ *on* $Z_b^{(\ell)}$ *to get a decision function* $\hat{g}_b^{(\ell)}$.

2. *Return* $\hat{g}(\mathbf{x}) = \underset{1 \leq \ell \leq K}{argmax}\, \hat{g}_b^{(\ell)}(\mathbf{x})$.

We can easily see Algorithm 2.4 is a special case of Algorithm 2.6 when all $\mathbf{c}_n$ are classification cost vectors.

2.3.2 Cost-Sensitive One-Versus-One

The *one-versus-one* (OVO) algorithm is another popular algorithm for weighted classification. It is suitable for practical use when K is not too large (Hsu and Lin 2002). Similar to the OVA algorithm, it also solves the weighted classification problem by decomposing it to several weighted binary classification problems. Unlike OVA, however, each binary classification problem consists of comparing examples from two categories only.

Algorithm 2.7 (One-versus-one, see, for instance, Hsu and Lin 2002).

1. *For each* i, j *that* $1 \leq i < j \leq K$,

   (a) *Take the original training set* $Z = \{(\mathbf{x}_n, y_n, w_n)\}_{n=1}^N$ *and construct a binary classification training set* $Z_b^{(i,j)} = \{(\mathbf{x}_n, y_n, w_n) \colon y_n = i \text{ or } y_n = j\}$.

   (b) *Use a weighted binary classification algorithm* $\mathcal{A}_b$ *on* $Z_b^{(i,j)}$ *to get a decision function* $\hat{g}_b^{(i,j)}$.

2. *Return* $\hat{g}(\mathbf{x}) = \underset{1\le\ell\le K}{argmax} \sum_{i<j} \left[\!\left[\hat{g}_b^{(i,j)}(\mathbf{x}) = \ell \right]\!\right]$.

In short, each $\hat{g}_b^{(i,j)}(\mathbf{x})$ intends to predict whether $\mathbf{x}$ "prefers" i or category j, and $\hat{g}$ predicts with the preference votes gathered from those $\hat{g}_b^{(i,j)}$. The goal of $\mathcal{A}_b$ is to locate decision functions $\hat{g}_b^{(i,j)}$ with a small $\pi\left(\hat{g}_b^{(i,j)}, F^{(i,j)}\right)$, where $dF^{(i,j)}(\mathbf{x}, y) = dF(\mathbf{x}, y \,|\, y = i \text{ or } j)$, because it can be proved that (Beygelzimer et al. 2005)

$$\pi(\hat{g}) \le 2 \sum_{i<j} \text{Prob}\left[y = i \text{ or } j\right] \cdot \pi\left(\hat{g}_b^{(i,j)}, F^{(i,j)}\right).$$

Let us see if we can use OVO as $\mathcal{A}_w$ in TSEW for cost-sensitive classification problems. Again, consider a cost-sensitive training example $\left(\mathbf{x}_1, 1, (0, 1, 1, 334)\right)$ in Z. Recall that it would introduce the following multilabeled examples in Z_w:

$$(\mathbf{x}_1, 1, 334), (\mathbf{x}_1, 2, 333), (\mathbf{x}_1, 3, 333).$$

If we directly use OVO as $\mathcal{A}_w$ in TSEW, the underlying binary classification algorithm $\mathcal{A}_b$ would use the following two examples in $Z_b^{(1,2)}$ to get $\hat{g}_b^{(1,2)}$:

$$(\mathbf{x}_1, 1, 334), (\mathbf{x}_1, 2, 333).$$

Note that these weighted examples can be equivalently generated by labeling $\mathbf{x}_1$ as 1 with probability $\frac{334}{667}$ and as 2 with probability $\frac{333}{667}$. Because the probabilities are both close to $\frac{1}{2}$, the labels are almost as if decided by throwing a fair coin. Therefore, the binary classification algorithm $\mathcal{A}_b$ may be confused by the two examples.

For any classifier $g_b^{(i,j)}\colon \mathcal{X} \to \{i, j\}$, and a given example $(\mathbf{x}_1, y_1, \mathbf{c}_1)$ above, we see that the classifier needs to pay a constant cost of 333 first, regardless of its prediction. Now, we can again use the technique of shifting costs by a constant as we did in constructing similar cost vectors. Then, the two examples $(\mathbf{x}_1, 1, 334), (\mathbf{x}_1, 2, 333)$ is the same as one single example of $(\mathbf{x}_1, 1, 1)$. The shifting not only simplifies $Z_b^{(i,j)}$ by eliminating one unnecessary example, but also removes the random relabeling ambiguity that caused the confusion discussed above.

Recall that Z_w consists of examples $(\mathbf{x}_{n\ell}, y_{n\ell}, w_{n\ell}) = (\mathbf{x}_n, \ell, \tilde{\mathbf{q}}_n[\ell])$. Thus, $(\mathbf{x}_n, i)$ would be of weight $\tilde{\mathbf{q}}_n[i]$ and $(\mathbf{x}_n, j)$ would be of weight $\tilde{\mathbf{q}}_n[j]$. By the discussion above, the simplified $Z_b^{(i,j)}$ consists of

$$\begin{aligned} Z_b^{(i,j)} &= \left\{ \left(\mathbf{x}_n, \underset{\ell=i \text{ or } j}{\operatorname{argmax}} \tilde{\mathbf{q}}_n[\ell], \left| \tilde{\mathbf{q}}_n[i] - \tilde{\mathbf{q}}_n[j] \right| \right) \right\} \\ &= \left\{ \left(\mathbf{x}_n, \underset{\ell=i \text{ or } j}{\operatorname{argmin}} \mathbf{c}_n[\ell], \left| \mathbf{c}_n[i] - \mathbf{c}_n[j] \right| \right) \right\}. \end{aligned} \tag{2.6}$$

Then, we get our proposed *cost-sensitive one-versus-one* (CSOVO) algorithm.

Algorithm 2.8 (CSOVO: Cost-sensitive one-versus-one).

1. *For each i, j that $1 \le i < j \le K$,*
   (a) *Take the original training set $Z = \{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^N$ and construct a binary classification training set by* (2.6)*.*
   (b) *Use a weighted binary classification algorithm $\mathcal{A}_b$ on $Z_b^{(i,j)}$ to get a decision function $\hat{g}_b^{(i,j)}$.*
2. *Return* $\hat{g}(\mathbf{x}) = \underset{1 \le \ell \le K}{argmax} \sum_{i<j} \left[\!\left[\hat{g}_b^{(i,j)}(\mathbf{x}) = \ell \right]\!\right]$.

We can easily see that CSOVO takes OVO as a special case when the using only the classification cost vectors. In addition, we can think of each created example $\left(\mathbf{x}_n, \underset{\ell=i \text{ or } j}{\operatorname{argmin}} \mathbf{c}_n[\ell], \left| \mathbf{c}_n[i] - \mathbf{c}_n[j] \right| \right)$ as if coming from a (possibly unnormalized) measure

$$dF_b^{(i,j)}(\mathbf{x}, k, w) = \int_{y,\mathbf{c}} \left[\!\left[k = \underset{\ell=i \text{ or } j}{\operatorname{argmin}} \mathbf{c}_n[\ell] \right]\!\right] \left[\!\left[w = \left| \mathbf{c}[i] - \mathbf{c}[j] \right| \right]\!\right] dF(\mathbf{x}, y, \mathbf{c}).$$

Define

$$\pi_b^{(i,j)}(g^{(i,j)}) = \int_{\mathbf{x},k,w} w \left[\!\left[k \neq g^{(i,j)}(\mathbf{x}) \right]\!\right] \, dF_b^{(i,j)}(\mathbf{x}, k, w).$$

We then get the following theorem:

Theorem 2.7. *For any family of classifiers* $\left\{g_b^{(i,j)} : 1 \le i < j \le K\right\}$*, where the binary classifiers* $g_b^{(i,j)} : \mathcal{X} \to \{i, j\}$*. Let*

$$g(\mathbf{x}) = \underset{1 \le \ell \le K}{argmax} \sum_{i<j} \left[\!\left[g_b^{(i,j)}(\mathbf{x}) = \ell \right]\!\right].$$

Then,

$$\pi(g) - \int_{\mathbf{x},y,\mathbf{c}} c_{\min}\, dF(\mathbf{x}, y, \mathbf{c}) \le 2 \sum_{i<j} \pi_b^{(i,j)} \left(g_b^{(i,j)} \right).$$

Proof. For each $(\mathbf{x}, y, \mathbf{c})$ generated from $dF(\mathbf{x}, y, \mathbf{c})$, if $\mathbf{c}[g(\mathbf{x})] = \mathbf{c}[y] = c_{\min}$, its contribution on the left-hand side is 0, which is trivially less than its contribution on the right-hand side.

Without loss of generality (by sorting the elements of the cost vector $\mathbf{c}$ and shuffling the labels $y \in \mathcal{Y}$), consider an example $(\mathbf{x}, y, \mathbf{c})$ such that

$$c_{\min} = \mathbf{c}[1] \le \mathbf{c}[2] \le \ldots \le \mathbf{c}[K] = c_{\max}.$$

From the results of Beygelzimer et al. (2005, Lemma 1), suppose $g(\mathbf{x}) = k$, then for each $1 \le \ell \le k-1$, there are at least $\lceil k/2 \rceil$ pairs (i, j), where $i \le k < j$, and $\underset{\ell=i \text{ or } j}{\operatorname{argmin}}\, \mathbf{c}_n[\ell] \ne g_b^{(i,j)}(\mathbf{x})$. Therefore, the contribution of $(\mathbf{x}, y, \mathbf{c})$ on the right-hand side is no less than

$$\begin{aligned}
\sum_{\ell=1}^{k-1} (\mathbf{c}[\ell+1] - \mathbf{c}[\ell]) \left\lceil \frac{\ell}{2} \right\rceil &\ge \frac{1}{2} \sum_{\ell=1}^{k-1} \ell \left(\mathbf{c}[\ell+1] - \mathbf{c}[\ell]\right) \\
&= \frac{1}{2} \sum_{\ell=2}^{k} (\ell-1)\mathbf{c}[\ell] - \frac{1}{2} \sum_{\ell=1}^{k-1} \ell \mathbf{c}[\ell] \\
&= \frac{1}{2}(k-1)\mathbf{c}[k] - \frac{1}{2} \sum_{\ell=1}^{k-1} \mathbf{c}[\ell] \\
&= \frac{1}{2} \sum_{\ell=1}^{k-1} \left(\mathbf{c}[k] - \mathbf{c}[\ell]\right) \\
&\ge \frac{1}{2} \left(\mathbf{c}[k] - c_{\min}\right),
\end{aligned}$$

and the left-hand-side contribution is $(\mathbf{c}[k] - c_{\min})$. The desired result can be proved by integrating over all $dF(\mathbf{x}, y, \mathbf{c})$. □

Theorem 2.7 provides a theoretical guarantee for CSOVO: If each $\hat{g}_b^{(i,j)}$ yields a small $\pi_b^{(i,j)}$, the resulting $\hat{g}$ would yield a small π. Beygelzimer et al. (2005) proposed another algorithm, called *weighted all-pairs* (WAP), that shared a similar theoretical guarantee and some algorithmic structures, as listed below.

Algorithm 2.9 (A special case of WAP, Beygelzimer et al. 2005).

1. *For each i, j that $1 \le i < j \le K$,*

   (a) *Take the original training set $Z = \{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^N$ and construct a binary classification training set by*

   $$Z_b^{(i,j)} = \left\{ \left(\mathbf{x}_n, \underset{\ell=i \text{ or } j}{argmin}\, \mathbf{c}_n[\ell], \left| \int_{\mathbf{c}_n[j]}^{\mathbf{c}_n[i]} \frac{1}{|\{k\colon \mathbf{c}_n[k] \le t\}|} dt \right| \right) \right\} \quad (2.7)$$

   (b) *Use a weighted binary classification algorithm $\mathcal{A}_b$ on $Z_b^{(i,j)}$ to get a decision function $\hat{g}_b^{(i,j)}$.*

2. *Return* $\hat{g}(\mathbf{x}) = \underset{1 \le \ell \le K}{argmax} \sum_{i<j} \left[\!\left[\hat{g}_b^{(i,j)}(\mathbf{x}) = \ell \right]\!\right]$.

We see that WAP is similar to CSOVO (Algorithm 2.8), except for how the weights of the binary examples are computed. CSOVO uses $\left|\mathbf{c}_n[i] - \mathbf{c}_n[j]\right|$, which is equivalent to

$$\left| \int_{\mathbf{c}_n[j]}^{\mathbf{c}_n[i]} (1)\, dt \right| .$$

We can think of both CSOVO and WAP as special instances of the cost-transformation technique that interprets $\tilde{\mathbf{q}}$ (or the original $\mathbf{c}$) differently. While the two algorithms are very similar in terms of theoretical guarantees and algorithmic structures, it should be noted that CSOVO enjoys the advantage of efficiency because (2.6) is simpler than (2.7).

Table 2.1: Classification data sets

data set	# examples	# categories (K)	# features (D)
vehicle	846	4	18
vowel	990	11	10
segment	2310	7	19
dna	3186	3	180
satimage	6435	6	36
usps	9298	10	256

2.4 Experiments

In this section, we compare the proposed CSOVA and CSOVO algorithms derived from the cost-transformation technique with their original versions. We also compare CSOVO with its closely related sibling, the WAP algorithm. All these algorithms obtains a decision function $\hat{g}$ by calling a binary classification algorithm $\mathcal{A}_b$ several times. We take the support vector machine (SVM) with the perceptron kernel (Lin and Li 2008, which will be further introduced in Chapter 5) as $\mathcal{A}_b$ in all the experiments and use LIBSVM (Chang and Lin 2001) as our SVM solver.

2.4.1 Comparison on Classification Data Sets

We first compare the algorithms with six benchmark classification data sets: vehicle, vowel, segment, dna, satimage, usps (Table 2.1).[5] The first five comes from the UCI machine learning repository (Hettich, Blake and Merz 1998) and the last one comes from Hull (1994).

The six data sets in Table 2.1 were originally gathered as regular classification problems. We adopt two kinds of setup to compare cost-sensitive algorithms. In the first one, called the *randomized proportional (RP) cost setup*, we follow the procedure used by Abe, Zadrozny and Langford (2004). In particular, we generate the cost vectors from a cost function $C(y, k)$ that does not depends on $\mathbf{x}$. $C(y, y)$ is set as 0 and $C(y, k)$ is a random variable sampled uniformly from $\left[0, 2000\frac{|\{n:\, y_n=k\}|}{|\{n:\, y_n=y\}|}\right]$.

Another setup is the *absolute cost setup*, which considers the *absolute cost vec-*

[5] They are downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`

tors $\left\{\mathbf{c}_a^{(\ell)}\right\}_{\ell=1}^{K}$ with

$$\mathbf{c}_a^{(\ell)}[k] = C_a(\ell, k) = |\ell - k| \,.$$

Note that the absolute cost vectors are not only V-shaped but also convex, and they are widely used in evaluating ordinal ranking algorithms (Chu and Keerthi 2007; Li and Lin 2007b).

We randomly choose 75% of the examples in each data set for training and leave the other 25% of the examples as the test set. Then, each feature in the training set is linearly scaled to $[-1, 1]$, and the feature in the test set is scaled accordingly. The results reported are all averaged over 20 trials of different training/test splits, along with the standard error.

SVM with the perceptron kernel takes a regularization parameter (Lin and Li 2008), which is chosen within $\{2^{-17}, 2^{-15}, \ldots, 2^3\}$ with a 5-fold cross-validation (CV) procedure on the training set (Hsu, Chang and Lin 2003). For the original OVA and OVO, the CV procedure selects the parameter that results in the smallest cross-validation regular classification cost. For the other algorithms, the CV procedure selects the parameter that results in the smallest cross-validation cost-sensitive classification cost based on the given setup. We then rerun each algorithm on the whole training set with the chosen parameter to get the decision function $\hat{g}$. Finally, we evaluate the average performance of $\hat{g}$ on the test set.

In Tables 2.2 and 2.3, we first compare CSOVO with WAP using the RP cost setup and the absolute cost setup, respectively. We see that the two algorithms perform similarly on almost all data sets. Because CSOVO is indistinguishable with WAP in terms of performance while enjoying an advantage of a simpler and more efficient implementation (see Subsection 2.3.2), it should be preferred in practice.

Next, we compare CSOVA and CSOVO with their original versions in Tables 2.4 and 2.5. The results in Table 2.4 come from the RP cost setup. We see that CSOVA and CSOVO are often significantly better than their original version respectively, which justifies the validity of the cost-transformation technique and our proposed

Table 2.2: Test RP cost of CSOVO and WAP

data set	CSOVO	WAP
vehicle	**145.745±18.404**	**153.494±20.148**
vowel	**19.277±1.899**	**19.279±2.242**
segment	**25.618±2.664**	**24.437±2.251**
dna	**51.961±4.543**	**49.546±4.388**
satimage	**65.812±4.463**	71.782±5.020
usps	**22.103±0.721**	**21.612±0.583**

(those within one standard error of the lowest one are marked in bold)

Table 2.3: Test absolute cost of CSOVO and WAP

data set	CSOVO	WAP
vehicle	**0.225±0.007**	**0.225±0.007**
vowel	**0.030±0.005**	**0.031±0.004**
segment	**0.045±0.003**	**0.046±0.003**
dna	**0.067±0.002**	**0.066±0.002**
satimage	**0.127±0.003**	**0.128±0.003**
usps	**0.089±0.002**	**0.090±0.003**

(those within one standard error of the lowest one are marked in bold)

algorithms.

In Table 2.5, however, we see that CSOVA and CSOVO are very similar to, and sometimes slightly worse than, their original versions using the absolute cost setup. The results in Tables 2.4 and 2.5 reflect a trade-off in using the cost-transformation technique. The cost-transformation technique makes it possible to plug cost vectors into an algorithm in a principled manner and hence improves the performance of the algorithm. On the other hand, the technique may introduce more complicated problems (see the discussions in Section 2.2) and thus deteriorates the performance of the algorithm. In Table 2.4, the cost-sensitive versions take a great advantage from the former perspective and thus outperform their original versions. In Table 2.5, however, given that the cost vectors are simpler for these data sets, the original versions already perform decently under this setup. Thus, the cost-sensitive versions cannot take much advantage from the former perspective and lose some performance because of the latter perspective.

Table 2.4: Test RP cost of cost-sensitive classification algorithms

data set	one-versus-all		one-versus-one	
	OVA	CSOVA	OVO	CSOVO
vehicle	189.064±17.866	**158.215±19.833**	185.378±17.235	**145.745±18.404**
vowel	14.654±1.766	14.386±1.717	**11.896±1.955**	19.277±1.899
segment	**25.263±2.015**	**25.434±2.208**	**25.153±2.109**	**25.618±2.664**
dna	44.480±2.771	**39.424±2.521**	48.152±3.333	51.961±4.543
satimage	93.381±5.712	77.101±4.762	94.075±5.488	**65.812±4.463**
usps	23.087±0.709	**22.793±0.710**	23.622±0.660	**22.103±0.721**

(those within one standard error of the lowest one are marked in bold)

Table 2.5: Test absolute cost of cost-sensitive classification algorithms

data set	one-versus-all		one-versus-one	
	OVA	CSOVA	OVO	CSOVO
vehicle	**0.222±0.007**	**0.226±0.007**	**0.221±0.007**	**0.225±0.007**
vowel	0.029±0.005	0.030±0.005	**0.023±0.004**	0.030±0.005
segment	**0.042±0.003**	**0.043±0.003**	**0.041±0.003**	0.045±0.003
dna	**0.053±0.002**	**0.054±0.002**	0.056±0.002	0.067±0.002
satimage	**0.124±0.003**	**0.123±0.003**	**0.125±0.003**	0.127±0.003
usps	**0.077±0.002**	**0.077±0.002**	0.080±0.002	0.089±0.002

(those within one standard error of the lowest one are marked in bold)

2.4.2 Comparison on Ordinal Ranking Data Sets

Next, we compare the algorithms with eight benchmark ordinal ranking data sets: pyrimdines, machineCPU, boston, abalone, bank, computer, california, census (Table 2.6), which were used by Chu and Keerthi (2007). Similar to the original setup of Chu and Keerthi (2007), we kept the same training/test split ratios, used the absolute cost vectors for evaluating the performance and averaged the results over 20 trials.

We compare OVA, CSOVA, OVO, and CSOVO on the eight data sets and list the results in Table 2.7. First, we see that CSOVA and CSOVO are significantly better than their regular classification versions. The results demonstrate that because of the ordinal information carried by the ranks, taking a V-shaped cost into consideration can lead to significant improvement. In addition, we see that CSOVO is better than CSOVA on these ordinal ranking data sets. Recall that the underlying binary classification algorithm in CSOVO tries to distinguish between two ranks i and j,

Table 2.6: Ordinal ranking data sets

data set	# training/# test	# levels (K)	# features (D)
pyrimdines	50/24	10	27
machineCPU	150/59	10	6
boston	300/206	10	13
abalone	1000/3177	10	10
bank	3000/5192	10	32
computer	4000/4192	10	21
california	5000/15640	10	8
census	6000/16784	10	16

Table 2.7: Test absolute cost of cost-sensitive classification algorithms on ordinal ranking data sets

data	one-versus-all		one-versus-one	
set	OVA	CSOVA	OVO	CSOVO
pyrimdines	1.765±0.103	1.627±0.055	1.746±0.076	**1.337±0.054**
machineCPU	0.983±0.027	0.975±0.024	1.009±0.024	**0.842±0.023**
boston	0.987±0.019	0.946±0.017	0.882±0.017	**0.789±0.015**
abalone	1.719±0.007	1.674±0.007	1.632±0.013	**1.422±0.006**
bank	1.863±0.007	1.801±0.004	1.632±0.004	**1.414±0.003**
computer	0.685±0.003	0.644±0.003	0.620±0.002	**0.575±0.002**
california	1.164±0.004	1.121±0.002	1.055±0.003	**0.951±0.002**
census	1.381±0.004	1.329±0.003	1.272±0.002	**1.135±0.001**

(those within one standard error of the lowest one are marked in bold)

which uses the ordinal information "<" implicitly. On the other hand, the underlying binary classification algorithm in CSOVA needs to distinguish between rank ℓ with ranks $\{1, \ldots, \ell-1, \ell+1, \ldots, K\}$, which is not naturally coherent with the ordinal information. The significant performance difference between CSOVA and CSOVO demonstrate the importance of using ordinal information in comparing examples from different ranks.

Chapter 3

Ordinal Ranking by Threshold Regression

As discussed in Section 1.2, ordinal ranking is similar to regression because the labels in either $\mathcal{Y}_r$ and $\mathbb{R}$ represent ordinal information. Nevertheless, unlike the real-valued regression labels in $\mathbb{R}$, the discrete ranks in $\mathcal{Y}_r$ do not carry metric information. That is, ordinal ranking deals with qualitative, fuzzy ranks while regression focuses on quantitative, real-valued outcomes. To model ordinal ranking problems from a regression perspective, it is often assumed that some underlying real-valued outcomes exist, but are unobservable (McCullagh 1980). The hidden local scales "around" different ranks can be quite different, but the actual scale (metric) information is not encoded in the ranks.

Under the assumption above, each rank represents a contiguous interval on the real line. Then, ordinal ranking can be approached by the following abstract algorithm called *threshold regression*.

Algorithm 3.1 (Threshold regression).

1. *Estimate a potential function $H(\mathbf{x})$ that predicts (a monotonic transform of) the real-valued outcomes. Ideally, feature vectors of larger ranks should be associated with larger potential values.*

2. *Determine a threshold vector $\theta \in \mathbb{R}^{K-1}$ to represent the intervals in the range of $H(\mathbf{x})$, where $\theta_1 \leq \theta_2 \leq \ldots \leq \theta_{K-1}$.*

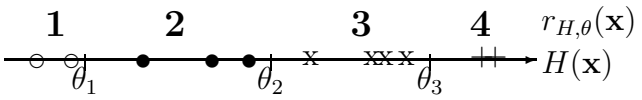


Figure 3.1: Prediction procedure of a threshold ranker

In the threshold regression algorithm, the potential function intends to uncover the nature of the assumed underlying outcome, and the threshold vector estimates the possibly different scales around different ranks. The two abstract steps of the algorithm are indeed taken by many existing ordinal ranking algorithms. For instance, in the GPOR algorithm of Chu and Ghahramani (2005), the potential function $H(\mathbf{x})$ is assumed to follow a Gaussian process, and the threshold vector θ is determined by Bayesian inference with respect to some noise distribution. In the PRank algorithm of Crammer and Singer (2005), the potential function $H_{\mathbf{v}}$ is taken to be a linear function of the form $H_{\mathbf{v}}(\mathbf{x}) = \langle \mathbf{v}, \mathbf{x} \rangle$, and the pair $(\mathbf{v}, \theta)$ are updated simultaneously. Some other algorithms are based on SVM, and they work on potential functions of the form $H_{\mathbf{v}}(\mathbf{x}) = \langle \mathbf{v}, \phi(\mathbf{x}) \rangle$, where $\phi(\mathbf{x})$ maps $\mathbf{x} \in \mathbb{R}^D$ to some Hilbert space (Chu and Keerthi 2007; Herbrich, Graepel and Obermayer 2000; Shashua and Levin 2003).

The learning model associated with the threshold regression algorithm is called the *threshold model*. Formally speaking, the *threshold model* $\mathcal{R}_{\mathcal{P}} = \{r_{H,\theta}\}$, where $H \in \mathcal{P}$ and $-\infty = \theta_0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_{K-1} \leq \theta_K = \infty$. As illustrated in Figure 3.1, each member $r_{H,\theta}$ of the threshold model is a *threshold ranker*, which makes its prediction by

$$r_{H,\theta}(\mathbf{x}) = \min \{k \colon H(\mathbf{x}) \leq \theta_k\} = \max \{k \colon H(\mathbf{x}) > \theta_{k-1}\} = 1 + \sum_{k=1}^{K-1} [\![H(\mathbf{x}) > \theta_k]\!] \,. \quad (3.1)$$

We denote $r_{H,\theta}$ as r_θ when H is clear from the context.

In this chapter, we propose the *threshold ensemble model*, which is a novel instance of the threshold model. We discuss its theoretical properties and design suitable algorithms based on them. Our study on the threshold ensemble model improves our general understanding of the threshold model, which will be further discussed in

Section 3.1 and Chapter 4. Each ranker in the threshold ensemble model is called a *threshold ensemble*, which uses an ensemble H_T of confidence functions as the potential function H. The ensemble is of the form

$$H_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}), \quad \alpha_t \in \mathbb{R}. \tag{3.2}$$

We assume that each confidence function $h_t \colon \mathcal{X} \to [-1, +1]$ comes from a hypothesis set $\mathcal{H}$. That is, $H_T \in \text{span}(\mathcal{H}) = \mathcal{P}$. The confidence function reflects a possibly imperfect ordering preference. Note that a special instance of the confidence function is a binary classifier $\mathcal{X} \to \{-1, +1\}$, which matches the fact that binary classification is a special case of ordinal ranking with $K = 2$ (Rudin et al. 2005). The ensemble linearly combines the ordering preferences with α. Note that we allow α_t to be any real value, which means that it is possible to reverse the ordering preference of h_t in the ensemble when necessary.

Ensemble models in general have been successfully used for classification and regression (Meir and Rätsch 2003). They not only introduce more stable predictions through the linear combination, but also provide sufficient power for approximating complicated target functions. The threshold ensemble model extends existing ensemble models to ordinal ranking and inherits many useful theoretical properties from them. Next, we discuss one such property: the large-margin bounds.

3.1 Large-Margin Bounds of Threshold Ensembles

The margin concept in binary classification says large margins produced by a binary classifier g should indicate a small $\pi(g)$. The concept is widely used to evaluate binary classifiers (Li et al. 2005; Vapnik 1995) and can be justified with many large-margin cost bounds of the form

$$\pi(g) \leq \nu(g, \Delta) + \text{complexity term},$$

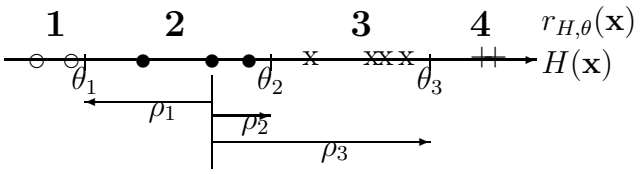


Figure 3.2: Margins of a correctly predicted example

where $\nu(g, \Delta) = \pi(g, Z_u, \Delta)$ is an extended form of training cost with respect to a margin parameter Δ, and the complexity term decreases as N or Δ increases.

For ordinal ranking using threshold models, Herbrich, Graepel and Obermayer (2000) derived a large-margin bound for any threshold ranker r_θ with a potential function $H_{\mathbf{v}} = \langle \mathbf{v}, \phi(\mathbf{x}) \rangle$. Unfortunately the bound is quite restricted since it is only applicable when $\nu(r_\theta, \Delta) = 0$ with respect to the classification cost function C_c. In addition, the bound uses a margin definition that contains $O(N^2)$ terms, which makes it more complicated to design algorithms that relate to the bound. Another bound was derived by Shashua and Levin (2003). The bound is based on a margin definition of only $O(KN)$ terms and is applicable to the threshold ensemble model. Nevertheless, the bound is loose when T, the size of the ensemble, is large, because its complexity term grows with T.

Next, we derive novel large-margin bounds of the threshold ensemble model with two widely used cost functions: the classification cost function C_c and the absolute cost function C_a. Similar bounds for more general cost-sensitive setup will be discussed in Chapter 4. The bounds are extended from the results of Schapire et al. (1998) and are based on a margin definition of $O(KN)$ terms. In addition, our bounds do not require $\nu(r_\theta, \Delta) = 0$ with respect to C_c, and their complexity terms do not grow with T.

We start by defining the margins of a threshold ensemble, which are illustrated in Figure 3.2. Intuitively, we expect the potential value $H(\mathbf{x})$ to be in the desired interval $(\theta_{y-1}, \theta_y]$, and we want $H(\mathbf{x})$ to be far from the boundaries (thresholds):

Definition 3.1. *Consider a given threshold ensemble $r_{H,\theta}$, where $H = H_T$.*

1. *The margin of an example $(\mathbf{x}, y)$ with respect to θ_k is defined as*

$$\rho_k(\mathbf{x}, y) = \begin{cases} H_T(\mathbf{x}) - \theta_k, & \text{if } y > k; \\ \theta_k - H_T(\mathbf{x}), & \text{if } y \leq k. \end{cases}$$

2. *The normalized margin $\tilde{\rho}_k(\mathbf{x}, y)$ is defined as*

$$\tilde{\rho}_k(\mathbf{x}, y) = \rho_k(\mathbf{x}, y) \Big/ \left(\sum_{t=1}^{T} |\alpha_t| + \sum_{k=1}^{K-1} |\theta_k| \right).$$

Definition 3.1 is similar to the definition of the SVM margin by Shashua and Levin (2003) and is analogous to the definition of margins in binary classification. A negative $\rho_k(\mathbf{x}, y)$ would indicate an incorrect prediction.

For each example $(\mathbf{x}, y)$, we can obtain $(K-1)$ margins from Definition 3.1. Two of them are of the most importance. The first one is $\rho_{y-1}(\mathbf{x}, y)$, which is the margin to the left (lower) boundary of the desired interval. The other is $\rho_y(\mathbf{x}, y)$, which is the margin to the right (upper) boundary. We will give them special names: the *left-margin* $\rho_L(\mathbf{x}, y)$ and the *right-margin* $\rho_R(\mathbf{x}, y)$. Note that by definition, $\rho_L(\mathbf{x}, 1) = \rho_R(\mathbf{x}, K) = \infty$.

Next, we take a closer look at the out-of-sample cost $\pi(r)$ and see how it connects to the margin definition above.

Δ-classification cost: For the classification cost function C_c, if we make a minor assumption that the degenerate cases $\tilde{\rho}_R(\mathbf{x}, y) = 0$ are of an infinitesimal probability,

$$\begin{aligned} \pi_c(r_\theta, F) &= \int_{\mathbf{x},y} [\![y \neq r_\theta(\mathbf{x})]\!] \; dF(\mathbf{x}, y) \\ &= \int_{\mathbf{x},y} [\![\tilde{\rho}_L(\mathbf{x}, y) \leq 0 \text{ or } \tilde{\rho}_R(\mathbf{x}, y) \leq 0]\!] \; dF(\mathbf{x}, y) \,. \end{aligned}$$

The definition could be generalized by expecting both margins to be larger than Δ.

That is, we can define the Δ-*classification cost* as

$$\pi_c(r_\theta, F, \Delta) \equiv \int_{\mathbf{x},y} [\![\tilde{\rho}_L(\mathbf{x}, y) \le \Delta \text{ or } \tilde{\rho}_R(\mathbf{x}, y) \le \Delta]\!] \; dF(\mathbf{x}, y)\,.$$

Then, $\pi_c(r_\theta, F)$ is just a special case with $\Delta = 0$. We can also define the *in-sample* Δ-*classification cost* $\nu_c(r_\theta, \Delta) \equiv \pi_c(r_\theta, Z_u, \Delta)$.

Δ-boundary cost: The "or" operation of $\pi_c(r_\theta, F, \Delta)$ is not easy to handle in the proof of the coming bounds. An alternative choice is the Δ-*boundary cost*:

$$\pi_b(r_\theta, F, \Delta) \equiv \int_{\mathbf{x},y} dF(\mathbf{x}, y) \cdot \begin{cases} [\![\tilde{\rho}_R(\mathbf{x}, y) \le \Delta]\!]\,, & \text{if } y = 1; \\ [\![\tilde{\rho}_L(\mathbf{x}, y) \le \Delta]\!]\,, & \text{if } y = K; \\ \frac{1}{2} \cdot \big([\![\tilde{\rho}_L(\mathbf{x}, y) \le \Delta]\!] + [\![\tilde{\rho}_R(\mathbf{x}, y) \le \Delta]\!]\big), & \text{otherwise.} \end{cases}$$

Similarly, the *in-sample* Δ-*boundary cost* ν_b is defined by $\nu_b(r_\theta, \Delta) \equiv \pi_b(r_\theta, Z_u, \Delta)$. Note that the Δ-boundary cost and the Δ-classification cost are equivalent up to a constant. That is, for any (r_θ, F, Δ),

$$\tfrac{1}{2}\pi_c(r_\theta, F, \Delta) \le \pi_b(r_\theta, F, \Delta) \le \pi_c(r_\theta, F, \Delta). \tag{3.3}$$

Δ-absolute cost: Next, we look at the out-of-sample cost based on the absolute cost function C_a. Again, if we make a minor assumption that the degenerate cases $\tilde{\rho}_k(\mathbf{x}, y) = 0$ are of an infinitesimal probability,

$$\pi_a(r_\theta, F) = \int_{\mathbf{x},y} \Big| y \neq r_\theta(\mathbf{x}) \Big| \; dF(\mathbf{x}, y) = \int_{\mathbf{x},y} \sum_{k=1}^{K-1} [\![\tilde{\rho}_k(\mathbf{x}, y) \le 0]\!] \; dF(\mathbf{x}, y)\,.$$

Then, the Δ-*absolute cost* can be defined as

$$\pi_a(r_\theta, F, \Delta) \equiv \int_{\mathbf{x},y} \sum_{k=1}^{K-1} [\![\tilde{\rho}_k(\mathbf{x}, y) \le \Delta]\!] \; dF(\mathbf{x}, y)\,,$$

which takes $\pi_a(r_\theta, F)$ as a special case with $\Delta = 0$. The *in-sample* Δ-*absolute cost* ν_a

is similarly defined by $\nu_a(r_\theta, \Delta) \equiv \pi_a(r_\theta, Z_u, \Delta)$.

An important observation for deriving our bounds is that $\pi_a(g, F, \Delta)$ can be written with respect to an additional sampling procedure on k. That is,

$$\begin{aligned}\pi_a(r_\theta, F, \Delta) &= \int_{\mathbf{x},y} \sum_{k=1}^{K-1} [\![\tilde{\rho}_k(\mathbf{x}, y) \le \Delta]\!]\ dF(\mathbf{x}, y) \\ &= (K-1) \int_{\mathbf{x},y,k} [\![\tilde{\rho}_k(\mathbf{x}, y) \le \Delta]\!] \left(\tfrac{1}{K-1} \cdot dF(\mathbf{x}, y)\right).\end{aligned}$$

Equivalently, we can define a probability measure $dF_E(\mathbf{x}, y, k)$ from $dF(\mathbf{x}, y)$ and an uniform distribution over $\{1, \ldots, K-1\}$ to generate the tuple $(\mathbf{x}, y, k)$. Then $\frac{1}{K-1}\pi_a(r_\theta, F, \Delta)$ is the portion of nonpositive $\tilde{\rho}_k(\mathbf{x}, y)$ under $dF_E(\mathbf{x}, y, k)$. Consider an extended training set $Z_E = \{(\mathbf{x}_n, y_n, k)\}$ with $N(K-1)$ elements. Each element is a possible outcome from $dF_E(\mathbf{x}, y, k)$. Note, however, that these elements are not all independent. For example, $(\mathbf{x}_n, y_n, 1)$ and $(\mathbf{x}_n, y_n, 2)$ are dependent. Thus, we cannot directly use the whole Z_E as a set of independent outcomes from $dF_E(\mathbf{x}, y, k)$.

Some subsets of Z_E contain independent outcomes from $dF_E(\mathbf{x}, y, k)$. One way to extract such a subset is to choose one k_n uniformly and independently from $\{1, \ldots, K-1\}$ for each example $(\mathbf{x}_n, y_n)$. The resulting subset would be named $Z_S = \{(\mathbf{x}_n, y_n, k_n)\}_{n=1}^N$. Then, we can obtain a large-margin bound of $\pi_a(r, F)$:

Theorem 3.2. *Consider a negation complete*[1] *set $\mathcal{H}$, which contains only binary classifiers $h\colon \mathcal{X} \to \{-1, 1\}$ and is of VC-dimension d. Assume that $\delta > 0$, and $N > d + K - 1 = d_E$. Then with probability at least $1 - \delta$ over the random choice of the training set Z, every threshold ensemble defined from* (3.1) *and* (3.2) *satisfies the following bound for all $\Delta > 0$:*

$$\pi_a(r_\theta, F) \le \nu_a(r_\theta, \Delta) + O\left(\frac{K}{\sqrt{N}}\left(\frac{d_E \log^2(N/d_E)}{\Delta^2} + \log\frac{1}{\delta}\right)^{1/2}\right).$$

[1] $\mathcal{H}$ is negation complete if and only if $h \in \mathcal{H} \iff (-h) \in \mathcal{H}$, where $(-h)(\mathbf{x}) = -\big(h(\mathbf{x})\big)$ for all x.

Proof. The key is to use the examples $(\mathbf{x}_n, y_n, k_n) \in Z_S$. Let

$$(\mathbf{X}_n^{(k_n)}, Y_n^{(k_n)}) = \begin{cases} \big((\mathbf{x}_n, \mathbf{1}_{k_n}), +1\big), & \text{if } y_n > k_n; \\ \big((\mathbf{x}_n, \mathbf{1}_{k_n}), -1\big), & \text{if } y_n \le k_n, \end{cases} \tag{3.4}$$

where $\mathbf{1}_k$ is a vector of length $(K-1)$ with a single 1 at the k-th dimension and 0 elsewhere. The test examples are constructed similarly with

$$\left(\mathbf{X}^{(k)}, Y^{(k)}\right) \equiv \left((\mathbf{x}, \mathbf{1}_k), 2\,[\![y > k]\!] - 1\right),$$

where $(\mathbf{x}, y, k)$ is generated from $dF_E(\mathbf{x}, y, k)$. Then, large-margin bounds for the ordinal ranking problem can be inferred from those for the binary classification problem. We first consider an ensemble function $g(\mathbf{X}^{(k)})$ defined by a linear combination of the functions in

$$\mathcal{G} = \left\{\tilde{h} \colon \tilde{h}(\mathbf{X}^{(k)}) = h(\mathbf{x}), h \in \mathcal{H}\right\} \cup \{s_\ell\}_{\ell=1}^{K-1}. \tag{3.5}$$

Here $s_\ell(\mathbf{X}^{(k)})$ is a decision stump on dimension $D + \ell$ (see Subsection 5.1.2). If the output space of s_ℓ is $\{-1, 1\}$, it is not hard to show that the VC-dimension of $\mathcal{G}$ is no more than $d_E = d + K - 1$. Since the proof of Schapire et al. (1998, Theorem 2), which will be applied on $\mathcal{G}$ later, only requires a combinatorial counting bound on the possible outputs of s_ℓ, we let

$$s_\ell(\mathbf{X}^{(k)}) = -\frac{\operatorname{sign}\left(\mathbf{X}^{(k)}[D+\ell] - 0.5\right) + 1}{2} = -\,[\![k = \ell]\!] \in \{-1, 0\}$$

to get a cosmetically cleaner proof. Some different versions of the bound can be obtained by considering $s_\ell(\mathbf{X}^{(k)}) \in \{-1, 1\}$ or by bounding the number of possible outputs of s_ℓ directly by a tighter term.

Without loss of generality, we normalize r_θ such that $\sum_{t=1}^{T} |\alpha_t| + \sum_{\ell=1}^{K-1} |\theta_\ell|$ is 1.

Then, consider an ensemble function

$$g(\mathbf{X}^{(k)}) = g(\mathbf{x}, \mathbf{1}_k) = H_T(\mathbf{x}) - \theta_k = \sum_{t=1}^{T} \alpha_t \tilde{h}_t(\mathbf{X}^{(k)}) + \sum_{k=1}^{K-1} \theta_\ell s_\ell(\mathbf{X}^{(k)}).$$

An important property for the transform is that for every $(\mathbf{X}^{(k)}, Y^{(k)})$ derived from the tuple $(\mathbf{x}, y, k)$, the term $\Big(Y^{(k)} \cdot g\big(\mathbf{X}^{(k)}\big)\Big) = \tilde{\rho}_k(\mathbf{x}, y)$.

Because Z_S contains N independent outcomes from $dF_E(\mathbf{x}, y, k) = dF_E\big(\mathbf{X}^{(k)}, Y^{(k)}\big)$, the large-margin theorem (Schapire et al. 1998, Theorem 2) states that with probability at least $1 - \frac{\delta}{2}$ over the choice of Z_S,

$$\begin{aligned}
&\int_{\mathbf{x},y,k} \left[\!\!\left[Y^{(k)} g\big(\mathbf{X}^{(k)}\big) \le 0 \right]\!\!\right] \; dF_E\big(\mathbf{X}^{(k)}, Y^{(k)}\big) \\
&\qquad \le \frac{1}{N} \sum_{n=1}^{N} \left[\!\!\left[Y_n^{(k)} g\big(\mathbf{X}_n^{(k)}\big) \le \Delta \right]\!\!\right] + O\left(\frac{1}{\sqrt{N}} \left(\frac{d_E \log^2(N/d_E)}{\Delta^2} + \log \frac{1}{\delta} \right)^{1/2} \right). \quad (3.6)
\end{aligned}$$

Since $\Big(Y^{(k)} \cdot g\big(\mathbf{X}^{(k)}\big)\Big) = \tilde{\rho}_k(\mathbf{x}, y)$, the left-hand side is $\frac{1}{K-1}\pi_a(r_\theta, F)$.

Let $b_n = \left[\!\!\left[Y_n^{(k)} g\Big(\mathbf{X}_n^{(k)}\Big) \le \Delta \right]\!\!\right] = \left[\!\!\left[\tilde{\rho}_{k_n}(\mathbf{x}_n, y_n) \le \Delta \right]\!\!\right]$, which is a Boolean random variable with mean $\frac{1}{K-1} \sum_{k=1}^{K-1} \left[\!\!\left[\tilde{\rho}_k(\mathbf{x}_n, y_n) \le \Delta \right]\!\!\right]$. Using Hoeffding's inequality (Hoeffding 1963), when each b_n is chosen independently, with probability at least $1 - \frac{\delta}{2}$ over the choice of b_n,

$$\begin{aligned}
\frac{1}{N} \sum_{n=1}^{N} b_n \;&\le\; \frac{1}{N} \sum_{n=1}^{N} \frac{1}{K-1} \sum_{k=1}^{K-1} \left[\!\!\left[\tilde{\rho}_k(\mathbf{x}_n, y_n) \le \Delta \right]\!\!\right] + O\left(\frac{1}{\sqrt{N}} \left(\log \frac{1}{\delta} \right)^{1/2} \right) \\
&=\; \frac{1}{K-1} \nu_a(r_\theta, \Delta) + O\left(\frac{1}{\sqrt{N}} \left(\log \frac{1}{\delta} \right)^{1/2} \right). \quad (3.7)
\end{aligned}$$

The desired result can be proved by combining (3.6) and (3.7) with a union bound.□

Similarly, if we look at the boundary cost,

$$\pi_b(r_\theta, F, \Delta) = \int_{\mathbf{x},y} \int_k \left[\!\!\left[\tilde{\rho}_k(\mathbf{x}, y) \le \Delta \right]\!\!\right] \; dF_E(k \,|\, \mathbf{x}, y) \; dF(\mathbf{x}, y)\,,$$

for some probability measure $dF_E(k \,|\, \mathbf{x}, y)$ on $\{L, R\}$. Then, a similar proof leads to

the following theorem.

Theorem 3.3. *For the same conditions as stated in Theorem 3.2,*

$$\pi_b(r_\theta, F) \leq \nu_b(r_\theta, \Delta) + O\left(\frac{1}{\sqrt{N}}\left(\frac{d_E \log^2(N/d_E)}{\Delta^2} + \log\frac{1}{\delta}\right)^{1/2}\right).$$

Then, a large-margin bound of the classification cost can be immediately derived by applying (3.3).

Corollary 3.4. *For the same conditions as stated in Theorem 3.2,*

$$\pi_c(r_\theta, F) \leq 2\nu_c(r_\theta, \Delta) + O\left(\frac{1}{\sqrt{N}}\left(\frac{d_E \log^2(N/d_E)}{\Delta^2} + \log\frac{1}{\delta}\right)^{1/2}\right).$$

Note that because of (3.3) and

$$\tfrac{1}{K-1}\pi_a(r_\theta, F, \Delta) \leq \pi_c(r_\theta, F, \Delta) \leq \pi_a(r_\theta, F, \Delta),$$

we can use either the classification, the boundary, or the absolute cost in the right-hand side and left-hand side of the bounds with some changes within $O(K)$.

The bounds above can be generalized when $\mathcal{H}$ contains confidence functions rather than binary classifiers. Even more generally, similar bounds can be derived for any threshold model, as shown below. The bounds provide motivations for building algorithms with margin-related formulations.

Theorem 3.5. *Let* $\mathcal{G}_E = \left\{g\colon g(\mathbf{X}^{(k)}) = H(\mathbf{x}) - \theta_k\right\}_{H\in\mathcal{P}}$. *Consider some* $\epsilon > 0$, *and* $\Delta > 0$. *When each* $(\mathbf{x}_n, y_n)$ *is generated independently from* $dF(\mathbf{x}, y)$,

$$\begin{aligned}&Prob\big[\exists r_\theta \in \mathcal{R}_\mathcal{P} \text{ such that } \pi_a(r_\theta, F) > \nu_a(r_\theta, \Delta) + K\epsilon\big]\\ &\qquad\leq 2\mathcal{N}\big(\mathcal{G}_E, \tfrac{\Delta}{2}, \tfrac{\epsilon}{8}, 2N\big)\exp\left(-\frac{\epsilon^2 N}{32}\right) + \exp\left(-2\epsilon^2 N\right).\end{aligned}$$

Furthermore,

$$\begin{aligned} &Prob\big[\exists r_\theta \in \mathcal{R}_\mathcal{P} \text{ such that } \pi_b(r_\theta, F) > \nu_b(r_\theta, \Delta) + 2\epsilon\big] \\ &\qquad \leq 2\mathcal{N}\big(\mathcal{G}_E, \tfrac{\Delta}{2}, \tfrac{\epsilon}{8}, 2N\big) \exp\left(-\frac{\epsilon^2 N}{32}\right) + \exp\left(-2\epsilon^2 N\right), \end{aligned}$$

and

$$\begin{aligned} &Prob\big[\exists r_\theta \in \mathcal{R}_\mathcal{P} \text{ such that } \pi_c(r_\theta, F) > 2\nu_c(r_\theta, \Delta) + 4\epsilon\big] \\ &\qquad \leq 2\mathcal{N}\big(\mathcal{G}_E, \tfrac{\Delta}{2}, \tfrac{\epsilon}{8}, 2N\big) \exp\left(-\frac{\epsilon^2 N}{32}\right) + \exp\left(-2\epsilon^2 N\right), \end{aligned}$$

where $\mathcal{N}(\mathcal{G}, \Delta, \epsilon, N)$ is maximum size of the smallest ϵ-sloppy Δ-cover of $\mathcal{G}$ over all possible set of N examples as defined by Schapire et al. (1998).

Proof. The proof extends the results of Schapire et al. (1998, Theorem 4) with essentially the same technique discussed in Theorem 3.2. □

Theorem 3.5 implies that if the term $\mathcal{N}(\mathcal{G}_E, \Delta, \epsilon, N)$ is polynomial in N, the probability that the out-of-sample cost deviates much from the in-sample Δ-cost is small. Similar to the work of Bartlett (1998), the theorem can be used to provide cost bounds for threshold rankers based on the neural network, which can be thought as a special form of threshold ensemble ranker.

3.2 Boosting Algorithms for Threshold Ensembles

The bounds in the previous section are applicable to threshold ensembles generated from any learning algorithm. One possible algorithm, for example, is an SVM-based approach (Chu and Keerthi 2007) with special kernels (Lin and Li 2008, which will be further discussed in Chapter 5). In this section, we focus on another branch of approaches: boosting. Boosting approaches can iteratively grow the ensemble $H_T(\mathbf{x})$ and have been successful in classification and regression (Meir and Rätsch 2003). Our

study includes an extension to the RankBoost algorithm (Freund et al. 2003) and two novel formulations that we propose.

3.2.1 RankBoost for Ordinal Ranking

RankBoost (Freund et al. 2003) constructs a weighted ensemble of confidence functions based on the following large-margin concept: For each index pair (m, n) such that $y_m > y_n$, the difference between their potential values, $H_t(\mathbf{x}_m) - H_t(\mathbf{x}_n)$, is desired to be positive and large. Thus, in the t-th iteration, the algorithm chooses (h_t, α_t) to approximately minimize

$$\sum_{y_m > y_n} \exp\Big(-\big(H_{t-1}(\mathbf{x}_m) + \alpha_t h_t(\mathbf{x}_m)\big) + \big(H_{t-1}(\mathbf{x}_n) + \alpha_t h_t(\mathbf{x}_n)\big)\Big). \tag{3.8}$$

Algorithm 3.2 (RankBoost, Freund et al. 2003).

1. *For* $t = 1, 2, \ldots, T$

   (a) *Determine the optimal* $h_t \in \mathcal{H}$ *by minimizing*

   $$\sum_{y_m > y_n} \exp\Big(-H_{t-1}(\mathbf{x}_m) + H_{t-1}(\mathbf{x}_n)\Big) \cdot \big(h_t(\mathbf{x}_m) - h_t(\mathbf{x}_n)\big). \tag{3.9}$$

   (b) *Determine the optimal* $\alpha_t \in \mathbb{R}$ *by minimizing* (3.8) *with respect to* α_t.

2. *Return the ensemble* H_T, *where* $H_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$.

Next, we plug RankBoost into the threshold regression algorithm (Algorithm 3.1) to get a concrete instance called *RankBoost for ordinal ranking* (RankBoost-OR).

Algorithm 3.3 (RankBoost-OR: RankBoost for ordinal ranking).

1. *Run an efficient implementation of RankBoost and obtain* $H = H_T$.

2. *Estimate the threshold* θ *by*

$$\theta = \underset{\vartheta}{\mathit{argmin}}\ \nu(r_{H,\vartheta}). \tag{3.10}$$

3. Return $r_{H,\theta}$.

Some details of the algorithm are discussed below.

Choosing h_t**:** Since there are $O(N^2)$ terms in (3.9), a naive implementation even for evaluating the equation takes at least $O(N^2)$ operations. Freund et al. (2003) showed that when $|\mathcal{Y}| = 2$, an efficient implementation of $O(N)$ operations exists. Here we extend the result to the case of ordinal ranking problems, in which $|\mathcal{Y}| = K$. Note that

$$
\begin{aligned}
&\sum_{y_m > y_n} \exp\Big(-H_{t-1}(\mathbf{x}_m) + H_{t-1}(\mathbf{x}_n)\Big) \cdot \big(h_t(\mathbf{x}_m) - h_t(\mathbf{x}_n)\big) \\
&= -\sum_{n=1}^{N} \sum_{y_m > y_n} \exp\Big(-H_{t-1}(\mathbf{x}_m) + H_{t-1}(\mathbf{x}_n)\Big) \cdot h_t(\mathbf{x}_n) \\
&\quad + \sum_{n=1}^{N} \sum_{y_m < y_n} \exp\Big(H_{t-1}(\mathbf{x}_m) - H_{t-1}(\mathbf{x}_n)\Big) \cdot h_t(\mathbf{x}_n) \\
&= -\sum_{n=1}^{N} \underbrace{\exp\Big(H_{t-1}(\mathbf{x}_n)\Big)}_{\varphi_n} \sum_{k=y_n+1}^{K} \underbrace{\sum_{m=1}^{N} [\![y_m = k]\!] \exp\Big(-H_{t-1}(\mathbf{x}_m)\Big)}_{\varphi_-^{(k)}} \cdot h_t(\mathbf{x}_n) \\
&\quad + \sum_{n=1}^{N} \exp\Big(-H_{t-1}(\mathbf{x}_n)\Big) \sum_{k=1}^{y_n-1} \underbrace{\sum_{m=1}^{N} [\![y_m = k]\!] \exp\Big(H_{t-1}(\mathbf{x}_m)\Big)}_{\varphi_+^{(k)}} \cdot h_t(\mathbf{x}_n) \\
&= \sum_{n=1}^{N} \underbrace{\left(-\varphi_n \cdot \sum_{k=y_n+1}^{K} \varphi_-^{(k)} + \varphi_n^{-1} \cdot \sum_{k=1}^{y_n-1} \varphi_+^{(k)}\right)}_{u_n} \cdot h_t(\mathbf{x}_n) \\
&= \sum_{n=1}^{N} |u_n| \cdot [\![\mathrm{sign}(u_n) \neq h_t(\mathbf{x}_n)]\!] + \text{ some constant that does not depend on } h_t.
\end{aligned}
$$

We see that φ_n can be tracked within $O(N)$ in each iteration, $\varphi_+^{(k)}$ and $\varphi_-^{(k)}$ can be computed in $O(N)$, and thus all u_n can be computed within $O(N+K)$. Note, however, that such an implementation is complicated.

As shown in the last line of the equations above, after computing u_n, a binary classification algorithm $\mathcal{A}_b$ (such as the ones in Subsections 2.3.1 and 2.3.2) can be

used to choose an optimal $h_t \in \mathcal{H}$. The algorithm is called a *base algorithm* in boosting literature.

Computing α_t: Two approaches can be used to determine α_t in RankBoost (Freund et al. 2003):

1. Obtain the optimal α_t by numerical search (confidence functions) or from an analytical solution (binary classifiers).

2. Minimize an upper bound of (3.8). For instance, let

$$\lambda_t = \frac{\sum_{y_m > y_n} \exp\Big(-H_{t-1}(\mathbf{x}_m) + H_{t-1}(\mathbf{x}_n)\Big) \cdot \big(h_t(\mathbf{x}_m) - h_t(\mathbf{x}_n)\big)}{2 \sum_{y_m > y_n} \exp\Big(-H_{t-1}(\mathbf{x}_m) + H_{t-1}(\mathbf{x}_n)\Big)}. \tag{3.11}$$

   An upper bound of (3.8) can be minimized by taking $\alpha_t = \frac{1}{4}\ln(\frac{1+\lambda_t}{1-\lambda_t})$. Similar to the case of computing h_t, an efficient implementation of $O(N + K^2)$ exists for computing λ_t (and hence α_t).

If $h_t(\mathbf{x}_n)$ is monotonic with respect to y_n, the optimal α_t obtained from approach 1 is ∞, and one single h_t would dominate the ensemble. This situation not only makes the ensemble less stable, but also limits its power. For instance, if we have four pairs of $(y_n, h_t(\mathbf{x}_n))$: $\{(1,-1),(2,0),(3,1),(4,1)\}$, then ranks 3 and 4 on the last two examples cannot be distinguished by h_t. We approach 1 is taken, we have frequently observed such a degenerate situation, called *partial matching*, in real-world experiments, even when h_t is as simple as a decision stump. Thus, we use approach 2 for our experiments.[2]

Obtaining θ: We use dynamic programming to solve (3.10). First, we sort the examples by their potential values. Then, without loss of generality, assume that $H(\mathbf{x}_1) \leq \cdots \leq H(\mathbf{x}_{N-1}) \leq H(\mathbf{x}_N)$. Consider the last example $(\mathbf{x}_N, y_N)$. If θ is chosen such that $\mathbf{x}_N$ is predicted as rank k, then a loss of $\frac{1}{N}\mathbf{c}_n[k]$ is introduced to

[2]In our earlier work (Lin and Li 2006), we used approach 2, but with an upper bound that is equivalent to an analytical solution when h_t are binary classifiers. Then (see Subsection 3.3.1) RankBoost may still encounter numerical difficulties. The upper bound listed in this section is "looser" and provides regularization even when h_t are binary classifiers.

ν. In addition, $\mathbf{x}_{N-1}$ can only be predicted as rank k or less. Furthermore, for θ to be optimal for all N examples, $(\theta_1, \ldots, \theta_k)$ must be optimal for the $(N-1)$ examples. Define $E(n,k)$ as the optimal total cost on examples of indices 1 to n and with possible ranks from $\{1, 2, \ldots, k\}$. By the discussion above,

$$E(n,k) = \min_{1 \le \ell \le k} \left(E(n-1,\ell) + \mathbf{c}_n[k] \right), \text{ and } E(0,k) = 0 \text{ for } k = 1, 2, \ldots, K.$$

Then, dynamic programming can efficiently find $E(N,K)$ and determine an optimal vector θ for (3.10). Note that there are many equivalent choices of θ_k between some $H(\mathbf{x}_{n-1})$ and $H(\mathbf{x}_n)$ in terms of (3.10). For stability and simplicity, we assign $\big(H(\mathbf{x}_{n-1}) + H(\mathbf{x}_n)\big)/2$ to θ_k, which achieves the largest minimum margin for a given $H(\mathbf{x})$ if $\nu_c(r_\theta, 0) = 0$. The time complexity for obtaining thresholds is $O(N \log N + KN)$, which is the bottleneck of RankBoost-OR.

3.2.2 ORBoost with Left-Right Margins

Although RankBoost-OR could yield the largest minimum margin under some conditions, it does not directly correspond to the definition of the margins in Section 3.1. Next, we propose two novel algorithms that connect to the margin bounds that we have derived. As indicated by the bounds, we want the margins to be as large as possible. To achieve this goal, our proposed algorithms work on minimizing the so-called *exponential margin loss* $E(r_\theta, Z) = \sum \exp(-\text{unnormalized margin})$. The loss partially explained the success of AdaBoost algorithm for binary classification (Mason et al. 2000; Schapire et al. 1998), and we adopt the same idea to design novel algorithms for ordinal ranking.

First, we introduce a simple formulation called *ORBoost with left-right margins* (ORBoost-LR), which intends to minimize

$$\sum_{n=1}^{N} \Big(\exp\big(-\rho_L(\mathbf{x}_n, y_n)\big) + \exp\big(-\rho_R(\mathbf{x}_n, y_n)\big) \Big). \tag{3.12}$$

The formulation can be thought as maximizing the soft-min of the left- and right-margins. Similar to RankBoost-OR, the minimization is performed in an iterative manner. In each iteration, a confidence function h_t is chosen, its weight α_t is computed, and the vector θ is updated. If we plug in the margin definition to (3.12), we can see that in the t-th iteration, the algorithm should approximately minimize

$$\sum_{n=1}^{N}\Big(\varphi_n \exp\big(\alpha_t h_t(\mathbf{x}_n) - \theta_{y_n}\big) + \varphi_n^{-1} \exp\big(\theta_{y_n-1} - \alpha_t h_t(\mathbf{x}_n)\big)\Big), \tag{3.13}$$

where $\varphi_n = \exp\big(H_{t-1}(\mathbf{x}_n)\big)$. The steps we designed are explained below.

Choosing h_t: Mason et al. (2000) explained AdaBoost as a gradient descent technique in function space. We derive ORBoost-LR using the same technique. We first choose a confidence function h_t that is close to the negative gradient:

$$h_t = \operatorname*{argmin}_{h\in\mathcal{H}} \sum_{n=1}^{N} h(\mathbf{x}_n) \underbrace{\Big(\varphi_n \exp\big(-\theta_{y_n}\big) - \varphi_n^{-1} \exp\big(\theta_{y_n-1}\big)\Big)}_{u_n}. \tag{3.14}$$

Here u_n can be easily computed within $O(N)$. Similar to RankBoost-OR (Algorithm 3.3), this step can also be performed with the help of a base algorithm.

Computing α_t: Again, we minimize an upper bound of (3.13), which is based on a piecewise linear approximation of $\exp(x)$ for $x \in [-1, 0]$ and $x \in [0, 1]$. The bound can be written as $\varphi_+ \exp(\alpha) + \varphi_- \exp(-\alpha)$, with

$$\begin{aligned}
\varphi_+ &= \sum_{h_t(\mathbf{x}_n)>0} h_t(\mathbf{x}_n)\varphi_n \exp\left(-\theta_{y_n}\right) - \sum_{h_t(\mathbf{x}_n)<0} h_t(\mathbf{x}_n)\varphi_n^{-1} \exp\left(\theta_{y_n-1}\right), \\
\varphi_- &= \sum_{h_t(\mathbf{x}_n)>0} h_t(\mathbf{x}_n)\varphi_n^{-1} \exp\left(\theta_{y_n-1}\right) - \sum_{h_t(\mathbf{x}_n)<0} h_t(\mathbf{x}_n)\varphi_n \exp\left(-\theta_{y_n}\right).
\end{aligned}$$

Then, the optimal α_t for the bound can be computed within $O(N)$ by

$$\frac{1}{2}\log\frac{\varphi_-}{\varphi_+}. \tag{3.15}$$

Note that the upper bound is equivalent to minimizing (3.13) if $h_t(\mathbf{x}_n) \in \{-1, 0, 1\}$. Thus, when h_t is a binary classifier, the optimal α_t can be exactly determined. Another remark here is that α_t is finite under some mild conditions. Thus, unlike RankBoost-OR when encountering the partial matching problem, ORBoost-LR rarely sets α_t to ∞.

Updating θ: Note that when the pair (h_t, α_t) is fixed, (3.13) can be reorganized as $\sum_{k=1}^{K-1} \varphi_+^{(k)} \exp(\theta_k) + \varphi_-^{(k)} \exp(-\theta_k)$ for some $\varphi_+^{(k)}$ and $\varphi_-^{(k)}$ that can be computed within $O(N)$. Then, each θ_k can be computed analytically, uniquely, and independently. Nevertheless, when each θ_k is updated independently, the thresholds may not be ordered. Hence, we propose to add an additional ordering constraint to (3.13). That is, we choose θ by solving

$$\begin{aligned} \min_{\vartheta} \quad & \sum_{k=1}^{K-1} \varphi_+^{(k)} \exp(\vartheta_k) + \varphi_-^{(k)} \exp(-\vartheta_k) \ , \\ \text{such that} \quad & \vartheta_1 \le \vartheta_2 \le \cdots \le \vartheta_{K-1}. \end{aligned} \tag{3.16}$$

An efficient algorithm for solving (3.16) can be obtained from by a simple modification of the *pool adjacent violators* (PAV) algorithm for isotonic regression (Robertson, Wright and Dykstra 1988), which is at most $O(K^2)$ of time complexity.

Combination of the steps: ORBoost-LR works by combining the steps above sequentially in each iteration. Note that after h_t is determined, α_t and θ_t can be either jointly optimized, or cyclically updated. Nevertheless, we found that joint or cyclic optimization does not always introduce better performance and could sometimes cause ORBoost-LR to overfit. Thus, we only execute each step once in each iteration.

From the discussions above, the exact steps of ORBoost-LR are as follows.

Algorithm 3.4 (ORBoost-LR: ORBoost with left-right margins).

1. *For* $t = 1, 2, \ldots, T$

   (a) *Determine the optimal* $h_t \in \mathcal{H}$ *in* (3.14) *with a base algorithm* $\mathcal{A}_b$.

(b) Determine the optimal $\alpha_t \in \mathbb{R}$ *by* (3.15).

(c) Update θ *by solving* (3.16).

2. Return the threshold ensemble $r_{H,\theta}$*, where* $H(\mathbf{x}) = H_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$.

3.2.3 ORBoost with All Margins

Another formulation, called *ORBoost with all margins* (ORBoost-All), operates on

$$\sum_{n=1}^{N} \sum_{k=1}^{K-1} \exp\big(-\rho_k(\mathbf{x}_n, y_n)\big) \tag{3.17}$$

instead of (3.13). The derivations for the three steps are almost the same as ORBoost-LR. We shall just make some remarks here.

Updating θ: When using (3.17) to update the thresholds, it can be proved that each θ_k can be updated uniquely and independently, while still being ordered (Li and Lin 2007b). Thus, we do not need to implement the PAV algorithm for ORBoost-All.

Relationship between algorithm and theory: Note that for any $\Delta > 0$,

$$\exp\left(-A\Delta\right) \cdot [\![\tilde{\rho}_k(\mathbf{x}_n, y_n) \leq \Delta]\!] \leq \exp\left(-A\tilde{\rho}_k(\mathbf{x}_n, y_n)\right).$$

Therefore, if we take A to be the normalization term of $\tilde{\rho}_k$, we can see that

- ORBoost-All works on minimizing an upper bound of $\nu_a(r_\theta, \Delta)$.
- ORBoost-LR works to minimizing an upper bound of $\nu_b(r_\theta, \Delta)$, or $\frac{1}{2}\nu_c(r_\theta, \Delta)$.

ORBoost-All not only minimizes an upper bound, but provably also minimizes the term $\nu_a(r_\theta, \Delta)$ exponentially fast with a sufficiently strong choice of h_t. The fact can be proved by applying the training cost theorem of AdaBoost (Schapire et al. 1998, Theorem 5) on Z_E. Similar proof can be also used for ORBoost-LR.

Connection to other algorithms: ORBoost approaches are direct generalizations of AdaBoost using the gradient descent optimization point-of-view. In the special

case of $K = 2$, both ORBoost approaches are almost the same as AdaBoost with an additional term θ_1, which can be thought as the coefficient of a constant classifier. Interestingly, Rudin et al. (2005) proved the connection between RankBoost and AdaBoost when including a constant classifier in the ensemble. Thus, when $K = 2$, RankBoost-OR, ORBoost-LR, and ORBoost-All, all share some similarity with AdaBoost.

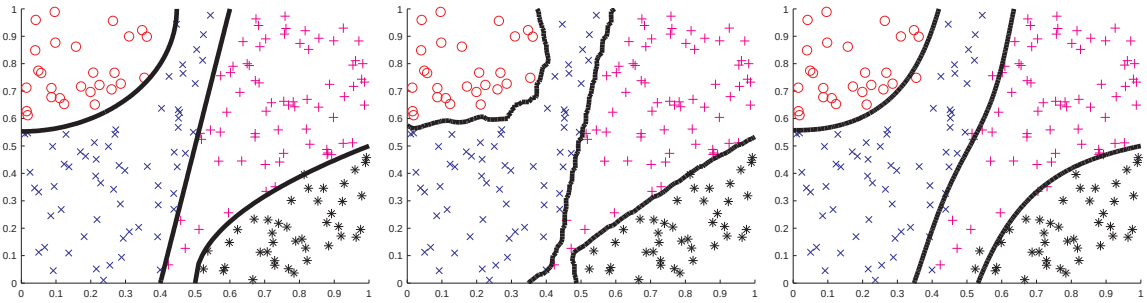
ORBoost formulations also have connections with SVM-based algorithms, such as SVOR by Chu and Keerthi (2007). In particular, ORBoost-LR has a counterpart of SVOR with explicit constraints (SVOR-EXC), and ORBoost-All is related to SVOR with implicit constraints (SVOR-IMC). These connections follow closely with the links between AdaBoost and SVM (Lin and Li 2008; Rätsch et al. 2002).

3.3 Experiments

In this section, we compare the three boosting formulations above for constructing the threshold ensembles. We also compare these formulations with SVM-based algorithms.

Two sets of confidence functions are used in the experiments. The first one is the set of perceptrons $\{\text{sign}(\langle \mathbf{v}, \mathbf{x} \rangle + b) : \mathbf{v} \in \mathbb{R}^D, b \in \mathbb{R}\}$. The RCD-bias algorithm is known to work well with AdaBoost (Li and Lin 2007a) and is adopted as our base algorithm. In all our experiments, RCD-bias is configured with zero seeding and 200 iterations.

The second set is $\{\tanh(\langle \mathbf{v}, x \rangle + b) \colon \langle \mathbf{v}, \mathbf{v} \rangle + b^2 = \gamma^2\}$, which contains normalized sigmoid functions. Note that sigmoid functions smoothen the output of perceptrons, and the smoothness is controlled by the parameter γ. We use a naive base algorithm for normalized sigmoid functions as follows: RCD-bias is first performed to get a perceptron. Then, the weights and bias of the perceptron are normalized, and the outputs are smoothened. Throughout the experiments we use $\gamma = 4$, which was picked with a few experimental runs on some data sets.



(a) the target (b) with perceptron (c) with sigmoid

Figure 3.3: Decision boundaries produced by ORBoost-All on an artificial data set

3.3.1 Artificial Data Set

We first verify that the idea of the threshold ensemble model works with an artificial 2D data set (Figure 3.3(a)). Figure 3.3(b) depicts the separating boundaries of the threshold ensemble of 200 perceptrons constructed by ORBoost-All. By combining perceptrons, ORBoost-All works reasonably well in approximating the nonlinear boundaries. A similar plot can be obtained with ORBoost-LR. Due to partial matching, if RankBoost-OR uses an analytic solution when choosing α_t, it would encounter numerical difficulties (see Subsection 3.2.1) within 5 iterations. On the other hand, when using the upper bound listed under (3.11) for choosing α_t, RankBoost-OR performs similarly to ORBoost algorithms.

If we use a threshold ensemble of 200 normalized sigmoid functions, it is observed that ORBoost-All, ORBoost-LR, and RankBoost-OR perform similarly. The result of ORBoost-All (Figure 3.3(c)) shows that the separating boundaries are much smoother because each sigmoid function is smooth. As we shall discuss later, the smoothness can be important for some ordinal ranking problems.

3.3.2 Benchmark Data Sets

Next, we perform experiments with eight benchmark data sets and the same setup as we did in Subsection 2.4.2. Here we compare two different cost functions: C_c and C_a. Since we use the same setup as Chu and Keerthi (2007), we can compare our proposed

Table 3.1: Test absolute cost of algorithms for threshold ensembles

data set	RankBoost-OR perceptron	RankBoost-OR sigmoid	ORBoost-All perceptron	ORBoost-All sigmoid	SVOR-IMC
pyrimdines	1.352±0.049	1.408±0.050	1.360±0.046	1.398±0.052	**1.294±0.046**
machine	**0.896±0.022**	**0.905±0.025**	**0.889±0.019**	0.969±0.025	0.990±0.026
boston	0.779±0.014	**0.746±0.014**	0.791±0.013	0.777±0.015	**0.747±0.011**
abalone	1.424±0.003	1.385±0.004	1.432±0.003	1.403±0.004	**1.361±0.003**
bank	1.457±0.002	1.456±0.002	1.490±0.002	1.539±0.002	**1.393±0.002**
computer	0.600±0.002	0.606±0.002	0.626±0.002	0.634±0.002	**0.596±0.002**
california	**0.919±0.002**	0.949±0.002	0.977±0.002	0.942±0.002	1.008±0.001
census	1.212±0.002	**1.186±0.002**	1.265±0.002	1.198±0.002	1.205±0.002

(those within one standard error of the lowest one are marked in bold)

Table 3.2: Test classification cost of algorithms for threshold ensembles

data set	RankBoost-OR perceptron	RankBoost-OR sigmoid	ORBoost-LR perceptron	ORBoost-LR sigmoid	SVOR-EXC
pyrimdines	**0.742±0.021**	**0.733±0.018**	**0.731±0.019**	**0.731±0.018**	0.752±0.014
machine	**0.614±0.009**	0.625±0.011	**0.610±0.009**	0.633±0.011	0.661±0.012
boston	0.570±0.005	**0.552±0.007**	0.580±0.006	**0.549±0.007**	0.569±0.006
abalone	0.738±0.002	0.719±0.002	0.740±0.002	**0.716±0.002**	0.736±0.002
bank	0.763±0.001	0.755±0.001	0.767±0.001	0.777±0.002	**0.744±0.001**
computer	0.485±0.002	0.491±0.001	0.498±0.001	0.491±0.001	**0.462±0.001**
california	0.607±0.001	0.620±0.001	0.628±0.001	**0.605±0.001**	0.640±0.001
census	0.706±0.001	0.700±0.001	0.718±0.001	**0.694±0.001**	0.699±0.000

(those within one standard error of the lowest one are marked in bold)

algorithms fairly with their SVM-based results.

We list the mean and standard errors of all test results with $T = 2000$ in Tables 3.1 and 3.2. Table 3.1 compares algorithms with C_a, and Table 3.2 compares algorithms with C_c. We make several remarks here.

RankBoost versus ORBoost: RankBoost-OR can achieve decent performance after we use a loose upper bound to decide α_t (see Subsection 3.2.1 and some of our earlier results of RankBoost-OR (Lin and Li 2006)). Its performance with the absolute cost is better than ORBoost-All, and its performance with the classification cost is slightly worse than ORBoost-LR. Overall we can see that all three algorithms work well on the data sets, while ORBoost ones enjoy an advantage of simplicity in implementation and efficiency.

Perceptron versus sigmoid: The best test performance are mostly achieved with sigmoid functions. One possible reason is that the data sets are quantized from regression ones (Chu and Keerthi 2007). Therefore, they hold some properties such as smoothness of the boundaries. If we only use binary classifiers like perceptrons, as depicted in Figure 3.3(b), the boundaries would not be as smooth. Thus, for ordinal ranking data sets that are quantized from regression data sets (or that follow the assumption of the threshold regression algorithm), smooth confidence functions may be more useful than discrete binary classifiers.

Boosting versus SVM: When comparing the boosting algorithms with SVOR-IMC on the classification cost and SVOR-EXC on absolute cost (Chu and Keerthi 2007), we see that boosting formulations could achieve similar test costs as the SVM-based algorithms. Note, however, that boosting formulations (especially ORBoost) with perceptrons or sigmoid functions are much faster. On the census data set, which contains 6000 training examples, it takes about an hour for ORBoost-LR to finish one trial. But SVM-based approaches, which include a time-consuming automatic parameter selection step, need more than four days. With the comparable performance and significantly less computational cost, ORBoost could be a useful tool for large data sets.

Chapter 4

Ordinal Ranking by Extended Binary Classification

In Chapter 2, we studied ordinal ranking problems from the classification perspective and proposed the novel CSOVA and CSOVO algorithms to tackle ordinal ranking problems via cost-sensitive classification. Both CSOVA and CSOVO decompose the cost-sensitive classification problem to several binary classification problems and call an underlying binary classification algorithm to solve them. In Chapter 3, we studied ordinal ranking problems from the regression perspective and proposed the threshold ensemble model for ordinal ranking. Each threshold ensemble in the model aggregates binary classifiers (confidence functions) to form its final prediction. We also designed RankBoost-OR and ORBoost algorithms, which return a threshold ensemble by calling a base binary classification algorithm several times. RankBoost-OR and ORBoost are derived from AdaBoost, a popular binary classification algorithm.

In other words, binary classification showed up frequently in our proposed approaches to deal with ordinal ranking. Since binary classification is arguably the most widely studied machine learning problem, it is not coincidental that we tackle more complicated machine learning problems, such as ordinal ranking, by reducing them to what we know in binary classification (Beygelzimer et al. 2005; Langford and Zadrozny 2005). A systematic reduction framework from ordinal ranking to binary classification can introduce two immediate benefits. First, well-tuned binary classification approaches can be readily transformed into good ordinal ranking ones,

which saves immense efforts in design and implementation. Second, new theoretical guarantees for ordinal ranking can be easily extended from known ones for binary classification, which saves tremendous efforts in derivation and analysis.

We introduced one such reduction framework in Subsection 2.3.2. The framework not only forms a cost-sensitive classification algorithm (CSOVO) by calling an underlying binary classification algorithm, but also guarantees that a good cost-sensitive classifier can be obtained by combining a set of decent binary classifiers. Since the framework is designed for general cost-sensitive classification rather than for ordinal ranking, arguably it does not use all the properties of ordinal ranking. For instance, it is not clear whether the framework explicitly makes ordinal comparisons between the ranks (see Section 1.2). In this chapter, we study another reduction framework that fully takes the properties of ordinal ranking into account. The framework includes both the classification and the regression perspective of ordinal ranking. Under this framework, we will eventually show an interesting fact: Ordinal ranking (with its full properties) is equivalent to binary classification.

4.1 Reduction Framework

The reduction framework was first proposed in our earlier work, which considered a more restricted cost-sensitive setup (Li and Lin 2007b).[1] The core of the framework is the following *reduction method*, which is composed of three stages: preprocessing, training, and prediction. Next, we introduce the stages of the reduction method and its consequent theoretical guarantees.

Algorithm 4.1 (Reduction to extended binary classification).

1. *Preprocessing: For each training example* $(\mathbf{x}_n, y_n, \mathbf{c}_n) \in Z$ *and for each* $k = 1, 2, \ldots, K-1$, *include an extended training example* $\left(\mathbf{X}_n^{(k)}, Y_n^{(k)}, W_n^{(k)}\right)$ *in* Z_E,

[1] All the results in this chapter are significantly original contributions of the author except Algorithm 4.1, Theorem 4.1 and Table 4.1, which arose from joint discussions between Dr. Ling Li and the author (Li and Lin 2007b). Dr. Ling Li substantially contributed to raising the idea of using extended binary examples, to designing Algorithm 4.1, to proving Theorem 4.1 for the case of convex cost vectors, and to unifying existing ordinal ranking algorithms in Table 4.1.

where

$$\mathbf{X}_n^{(k)} = (\mathbf{x}_n, k),\ Y_n^{(k)} = 2\,[\![y_n > k]\!] - 1,\ W_n^{(k)} = (K-1) \cdot \Big|\mathbf{c}_n[k+1] - \mathbf{c}_n[k]\Big|.$$

2. *Training: Use some binary classification algorithm $\mathcal{A}_b$ on Z_E and get a binary classifier g where $g\big(\mathbf{X}^{(k)}\big) = g(\mathbf{x}, k)$.*

3. *Prediction: For any $\mathbf{x} \in \mathcal{X}$, estimate its rank with*

$$\hat{r}(\mathbf{x}) = r_g(\mathbf{x}) \equiv 1 + \sum_{k=1}^{K-1} [\![g(\mathbf{x}, k) > 0]\!]\,. \tag{4.1}$$

We have encountered the extended examples $\big(\mathbf{X}^{(k)}, Y^{(k)}\big)$ when proving Theorem 3.2. Here the extended examples are weighted and are of a more generic form. In particular, $\mathbf{X}^{(k)}$ now takes an abstract encoding of $(\mathbf{x}, k)$ rather than a concrete encoding $(\mathbf{x}, \mathbf{1}_k)$. The actual encoding of k can then depend on the binary classification algorithm $\mathcal{A}_b$. We can think of the extended training vector $\mathbf{X}^{(k)}$ as a representation of the question "is the rank of $\mathbf{x}$ greater than k?", the binary label $Y^{(k)}$ as a representation of the desired answer to the question, and the classifier $g\big(\mathbf{X}^{(k)}\big)$ as a function that predicts the answer to the question.

A threshold ensemble $r_{H_T,\theta}$, for instance, is the same as r_g using

$$g\big(\mathbf{X}^{(k)}\big) = H_T(\mathbf{x}) - \theta_k = H_T(\mathbf{x}) - \sum_{\ell=1}^{K-1} \theta_\ell \cdot s_\ell(\mathbf{x}, \mathbf{1}_k).$$

Then, ORBoost-LR (Algorithm 3.4) is simply a special case of the reduction method above using the classification cost vectors $\left\{\mathbf{c}_c^{(\ell)}\right\}_{\ell=1}^{K}$ in Z, the vector $(\mathbf{x}, \mathbf{1}_k)$ as the actual encoding of $\mathbf{X}^{(k)}$, and a variant of AdaBoost as $\mathcal{A}_b$ that works on the learning model $\mathcal{G}$ in (3.5).

While the reduction method is simple, it comes with a strong theoretical guarantee: the cost bound theorem, which depends on the following probability mea-

sure $dF_b\big(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\big)$ that generates weighted binary examples.

1. Draw a tuple $(\mathbf{x}, y, \mathbf{c})$ independently from $dF(\mathbf{x}, y, \mathbf{c})$ and draw k uniformly from the set $\{1, 2, \ldots, K-1\}$.

2. Generate $\big(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\big)$ by

$$\begin{cases} \mathbf{X}^{(k)} &= (\mathbf{x}, k)\,, \\ Y^{(k)} &= 2\,[\![y > k]\!] - 1\,, \\ W^{(k)} &= (K-1) \cdot \Big|\mathbf{c}[k+1] - \mathbf{c}[k]\Big|. \end{cases} \tag{4.2}$$

As shown in the proof of Theorem 3.2, the extended training set Z_E contains dependent training examples from $dF_b\big(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\big)$. For any given binary classifier g, we can then define its out-of-sample cost

$$\pi_b(g) \equiv \pi(g, F_b) = \int_{\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}} W^{(k)} \cdot [\![Y^{(k)} \neq g\big(\mathbf{X}^{(k)}\big)]\!]\; dF_b\big(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\big)\,.$$

Next, we introduce the cost bound theorem (Li and Lin 2007b).

Theorem 4.1 (Cost bound of the reduction framework). *Consider a ranker r_g constructed from a binary classifier g using* (4.1)*. Assume that* $\mathbf{c}$ *is V-shaped and* $\mathbf{c}[y] = 0$ *for every example* $(\mathbf{x}, y, \mathbf{c})$*. If $g(\mathbf{x}, k)$ is rank-monotonic*[2] *or if every cost vector* $\mathbf{c}$ *is convex (see Section 1.2), then $\pi(r_g) \leq \pi_b(g)$.*

Proof. When $g(\mathbf{x}, k)$ is rank-monotonic, by (4.1), $[\![g(\mathbf{x}, k) \leq 0]\!] = [\![r_g(\mathbf{x}) \leq k]\!]$. Thus,

$$\begin{aligned} \mathbf{c}[r_g(\mathbf{x})] &= \mathbf{c}[K] + \sum_{k=r_g(\mathbf{x})}^{K-1} \Big(\mathbf{c}[k] - \mathbf{c}[k+1]\Big) \\ &= \mathbf{c}[K] + \sum_{k=1}^{K-1} \Big(\mathbf{c}[k] - \mathbf{c}[k+1]\Big)\, [\![g(\mathbf{x}, k) \leq 0]\!] \\ &= \mathbf{c}[y] + \frac{1}{K-1} \sum_{k=1}^{K-1} W^{(k)}\, [\![g\big(\mathbf{X}^{(k)}\big) \neq Y^{(k)}]\!]\,. \end{aligned} \tag{4.3}$$

[2] A rank-monotonic $g(\mathbf{x}, k)$ means $g(\mathbf{x}, k-1) \geq g(\mathbf{x}, k)$, for $k = 2, 3, \ldots, K-1$.

Then, we get

$$\begin{aligned}
\pi(r_g) &= \int_{\mathbf{x},y,\mathbf{c}} \mathbf{c}[r_g(\mathbf{x})]\ dF(\mathbf{x},y,\mathbf{c}) \\
&= \int_{\mathbf{x},y,\mathbf{c}} \frac{1}{K-1} \sum_{k=1}^{K-1} W^{(k)}\ [\![g\big(\mathbf{X}^{(k)}\big) \neq Y^{(k)}]\!]\ dF(\mathbf{x},y,\mathbf{c}) \\
&= \int_{\mathbf{X}^{(k)},Y^{(k)},W^{(k)}} W^{(k)}\ [\![g\big(\mathbf{X}^{(k)}\big) \neq Y^{(k)}]\!]\ dF_b\big(\mathbf{X}^{(k)},Y^{(k)},W^{(k)}\big) \\
&= \pi_b(g).
\end{aligned}$$

Now consider the case where every cost vector $\mathbf{c}$ is convex. Note that there are exactly $r_g(\mathbf{x}) - 1$ and $K - r_g(\mathbf{x})$ ones in those $[\![g(\mathbf{x},k) \leq 0]\!]$. In addition, there are also exactly $r_g(\mathbf{x}) - 1$ zeros and $K - r_g(\mathbf{x})$ ones in those $[\![r_g(\mathbf{x}) \leq k]\!]$. Because the vector $\Big([\![r_g(\mathbf{x}) \leq k]\!] \Big)_{k=1}^{K}$ is monotonically increasing, and by the convexity condition $\Big(\mathbf{c}[k] - \mathbf{c}[k+1] \Big)_{k=1}^{K}$ is also monotonically increasing,

$$\begin{aligned}
& \mathbf{c}[K] + \sum_{k=r_g(\mathbf{x})}^{K-1} \Big(\mathbf{c}[k] - \mathbf{c}[k+1] \Big) \\
= \ & \mathbf{c}[K] + \sum_{k=1}^{K-1} \Big(\mathbf{c}[k] - \mathbf{c}[k+1] \Big)\ [\![r_g(\mathbf{x}) \leq k]\!] \\
\leq \ & \mathbf{c}[K] + \sum_{k=1}^{K-1} \Big(\mathbf{c}[k] - \mathbf{c}[k+1] \Big)\ [\![g(\mathbf{x},k) \leq 0]\!] \,. \qquad (4.4)
\end{aligned}$$

The desired proof follows by replacing (4.3) with (4.4). □

Theorem 4.1 indicates that if there exists a descent binary classifier g, we can obtain a good ranker r_g. Nevertheless, it does not guarantee how good r_g is in comparison with other rankers. In particular, if we consider the target function g_* under $dF_b\big(\mathbf{X}^{(k)},Y^{(k)},W^{(k)}\big)$, and the target function r_* under $dF(\mathbf{x},y,\mathbf{c})$, does a small regret $\big(\pi_b(g) - \pi_b(g_*)\big)$ in binary classification translate to a small regret $\big(\pi(r_g) - \pi(r_*)\big)$ in ordinal ranking? Furthermore, is $\pi(r_{g_*})$ close to $\pi(r_*)$? Next, we introduce the *reverse-reduction technique*, which helps to answer the questions above.

Reverse reduction goes through the preprocessing and the prediction stages of the reduction method in a different direction. In the preprocessing stage, instead of starting with ordinal examples $(\mathbf{x}_n, y_n, \mathbf{c}_n)$, reverse reduction deals with weighted binary examples $\left(\mathbf{X}_n^{(k)}, Y_n^{(k)}, W_n^{(k)}\right)$. It first combines each set of binary examples sharing the same $\mathbf{x}_n$ to an ordinal example by

$$\begin{cases} y_n = & 1 + \sum_{k=1}^{K-1} \left[\!\left[Y_n^{(k)} > 0 \right]\!\right] \ ; \\ \mathbf{c}_n[k] = & \sum_{\ell=1}^{K-1} \frac{w_{n\ell}}{K-1} \cdot \left[\!\left[y_n \le \ell < k \text{ or } k < \ell \le y_n \right]\!\right]. \end{cases} \tag{4.5}$$

It is easy to verify that (4.5) is the exact inverse transform of (4.2) on the training examples. These ordinal examples are then given to an ordinal ranking algorithm to obtain a ranker r. In the prediction stage, reverse reduction works by decomposing the prediction $r(\mathbf{x})$ to $K-1$ binary predictions, each as if coming from a binary classifier

$$g_r\left(\mathbf{X}^{(k)}\right) = 2\left[\!\left[r(\mathbf{x}) > k \right]\!\right] - 1. \tag{4.6}$$

Then, a lemma on the out-of-sample cost of g_r immediately follows.

Lemma 4.2. *With the definitions of $F(\mathbf{x}, y, \mathbf{c})$ and $F_b\left(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\right)$ in Theorem 4.1, for every ordinal ranker r, $\pi(r) = \pi_b(g_r)$.*

Proof. Because g_r is rank-monotonic, the same proof of Theorem 4.1 leads to the desired result. □

The stages of reduction and reverse reduction are illustrated in Figure 4.1. Next, we show how the reverse-reduction technique complements the reduction method and allows us to draw a strong theoretical connection between ordinal ranking and binary classification. In addition, reverse reduction is useful in designing boosting algorithms for ordinal ranking, which will be demonstrated in Subsection 4.2.2.

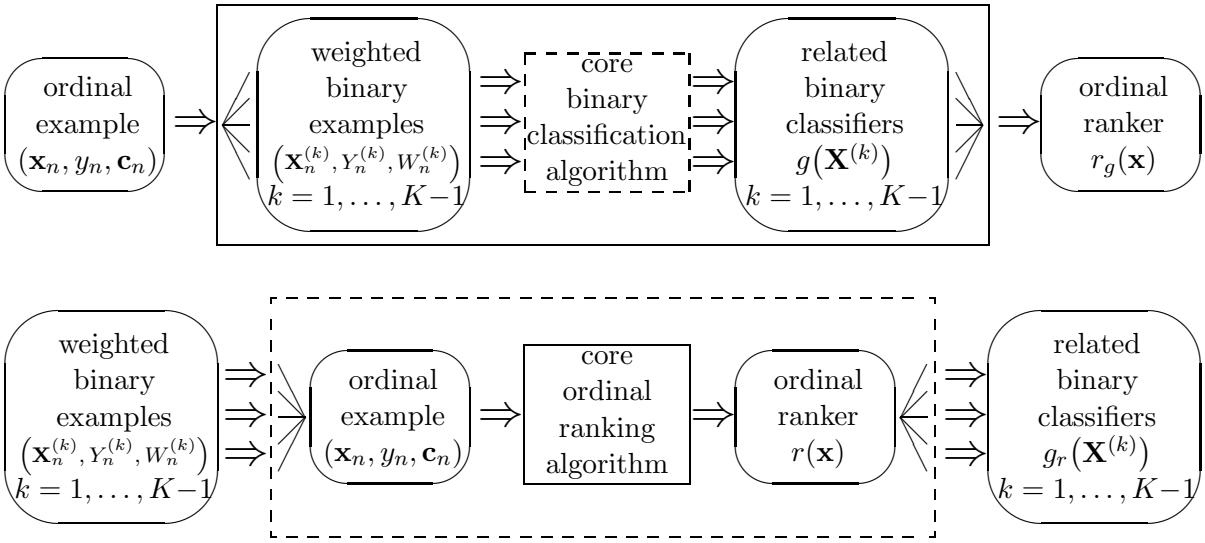


Figure 4.1: Reduction (top) and reverse reduction (bottom)

Recall that by the definition of r_* and g_*, for any ranker r and any binary classifier g,

$$\pi(r) \geq \pi(r_*), \quad \pi_b(g) \geq \pi_b(g_*) \; . \tag{4.7}$$

With the definitions of r_* and g_*, the reverse-reduction technique allows a simple proof of the following regret bound.

Theorem 4.3 (Regret bound of the reduction framework). *If $g(\mathbf{x}, k)$ is rank-monotonic, or if every cost vector $\mathbf{c}$ is convex, then*

$$\pi(r_g) - \pi(r_*) \leq \pi_b(g) - \pi_b(g_*).$$

Proof.

$$\begin{aligned} \pi(r_g) - \pi(r_*) \quad &\leq \quad \pi_b(g) - \pi(r_*) \quad && \text{(from Theorem 4.1)} \\ &= \quad \pi_b(g) - \pi_b(g_{r_*}) \quad && \text{(from Lemma 4.2)} \\ &\leq \quad \pi_b(g) - \pi_b(g_*) \quad && \big(\text{from (4.7)}\big) \qquad . \end{aligned}$$

□

The cost bound (Theorem 4.1) and the regret bound (Theorem 4.3) provide different guarantees for the reduction method. The former describes how the ordinal ranking cost is upper bounded by the binary classification cost in an absolute sense, and the latter describes the upper bound in a relative sense. An immediate implication of the regret bound is as follows. If there exists an optimal binary classifier g_+ that is also rank-monotonic, both the right-hand side and the left-hand side of the equation are 0. That is, every optimal binary classifier under $dF_b\big(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\big)$ corresponds to an optimal ranker under $dF(\mathbf{x}, y, \mathbf{c})$. In other words, there is no gap between ordinal ranking and binary classification in terms of optimality. In the following theorem, we show a general sufficient condition for the correspondence.

Theorem 4.4. *Assume that the effective cost*

$$\mathbf{c}_{\mathbf{x}}[k] = \int_{\mathbf{c},y} \mathbf{c}[k]\ dF(\mathbf{c}, y \,|\, \mathbf{x}) - \min_{1\le\ell\le K} \int_{\mathbf{c},y} \mathbf{c}[\ell]\ dF(\mathbf{c}, y \,|\, \mathbf{x})$$

is V-shaped with respect to $y_{\mathbf{x}} = \underset{1\le\ell\le K}{\mathit{argmin}}\ \mathbf{c}_{\mathbf{x}}[\ell]$ *on every feature vector* $\mathbf{x} \in \mathcal{X}$*. Let*

$$g_+(\mathbf{x}, k) \equiv 2\,[\![y_{\mathbf{x}} > k]\!] - 1.$$

Then g_+ *is rank-monotonic and is optimal under* $dF_b\big(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\big)$.

Proof. By construction g_+ is rank-monotonic. Because the effective cost $\mathbf{c}_{\mathbf{x}}$ is V-shaped, for all $\mathbf{c}_{\mathbf{x}}[k+1] - \mathbf{c}_{\mathbf{x}}[k] \neq 0$,

$$g_+(\mathbf{x}, k) = \operatorname{sign}(\mathbf{c}_{\mathbf{x}}[k+1] - \mathbf{c}_{\mathbf{x}}[k])\,.$$

That is,

$$g_+(\mathbf{x}, k) \cdot \Big(\mathbf{c}_{\mathbf{x}}[k+1] - \mathbf{c}_{\mathbf{x}}[k]\Big) = \Big|\mathbf{c}_{\mathbf{x}}[k+1] - \mathbf{c}_{\mathbf{x}}[k]\Big|.$$

Then, for any binary classifier g,

$$\begin{aligned}
\pi_b(g) &= \int_{\mathbf{X}^{(k)},Y^{(k)},W^{(k)}} W^{(k)} \left[\!\left[Y^{(k)} \neq g\big(\mathbf{X}^{(k)}\big) \right]\!\right] \; dF_b\big(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\big) \\
&= \int_{\mathbf{X}^{(k)},Y^{(k)},W^{(k)}} W^{(k)} \left(Y^{(k)} - g\big(\mathbf{X}^{(k)}\big) \right)^2 \; dF_b\big(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\big) \\
&= \Delta - 2 \int_{\mathbf{X}^{(k)},Y^{(k)},W^{(k)}} W^{(k)} \cdot Y^{(k)} \cdot g\big(\mathbf{X}^{(k)}\big) \; dF_b\big(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\big) \\
&= \Delta - \frac{2}{K-1} \sum_{k=1}^{K-1} \int_{\mathbf{x}} g(\mathbf{x}, k) \int_{y,\mathbf{c}} \big(\mathbf{c}[k+1] - \mathbf{c}[k]\big) \; dF(y, \mathbf{c} \,|\, \mathbf{x}) \; dF(\mathbf{x}) \\
&= \Delta - \frac{2}{K-1} \sum_{k=1}^{K-1} \int_{\mathbf{x}} g(\mathbf{x}, k) \cdot \Big(\mathbf{c}_{\mathbf{x}}[k+1] - \mathbf{c}_{\mathbf{x}}[k] \Big) \; dF(\mathbf{x}) \\
&\geq \Delta - \frac{2}{K-1} \sum_{k=1}^{K-1} \int_{\mathbf{x}} \Big| \mathbf{c}_{\mathbf{x}}[k+1] - \mathbf{c}_{\mathbf{x}}[k] \Big| \; dF(\mathbf{x}) \\
&= \Delta - \frac{2}{K-1} \sum_{k=1}^{K-1} \int_{\mathbf{x}} g_+(\mathbf{x}, k) \cdot \Big(\mathbf{c}_{\mathbf{x}}[k+1] - \mathbf{c}_{\mathbf{x}}[k] \Big) \; dF(\mathbf{x}) \\
&= \pi_b(g_+),
\end{aligned}$$

where Δ is a constant that does not depend on g. Thus, the classifier g_+ is optimal under $dF_b\big(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\big)$. □

Note that if every cost vector $\mathbf{c}$ is convex, the effective cost $\mathbf{c}_{\mathbf{x}}$ would also be convex and hence V-shaped. Thus, the convexity of $\mathbf{c}$ is also a (weaker) sufficient condition for the correspondence between optimal binary classifiers and optimal rankers.

As we can see from the definition of r_* in (1.1), the effective cost $\mathbf{c}_{\mathbf{x}}$ conveys sufficient information for determining the optimal prediction at $\mathbf{x}$. Because ordinal ranking predictions should take "closeness" into account (see Section 1.2), it is reasonable to assume that $\mathbf{c}_{\mathbf{x}}$ is V-shaped. Hence, in general (with such a minor assumption), optimal binary classifiers correspond to optimal rankers.

The results above demonstrate that ordinal ranking can be reduced to binary classification without any loss of optimality. That is, ordinal ranking is "no harder than" binary classification. Intuitively, binary classification is also "no harder than" ordinal ranking, because the former is a special case of the latter with $K = 2$. Next,

we formalize the notion of hardness with the *probably approximately correct* (PAC) setup in computational learning theory (Kearns and Vazirani 1994) and prove that ordinal ranking and binary classification are indeed equivalent in hardness. We use the following definition of PAC in our coming theorems (Kearns and Vazirani 1994; Valiant 1984).

Definition 4.5. *In cost-sensitive classification, a learning model $\mathcal{G}$ is efficiently PAC-learnable (using the same representation class) if there exists a learning algorithm $\mathcal{A}$ satisfying the following property: for every distribution $dF(\mathbf{x}, y, \mathbf{c})$ being considered, where*

$$\mathbf{c}[g_*(\mathbf{x})] = \mathbf{c}[y] = c_{\min} = 0,$$

with some $g_* \in \mathcal{G}$; for all $0 < \epsilon$ and $0 < \delta < \frac{1}{2}$, if $\mathcal{A}$ is given access to an oracle generating examples $(\mathbf{x}, y, \mathbf{c})$ from $dF(\mathbf{x}, y, \mathbf{c})$, as well as inputs ϵ and δ, then $\mathcal{A}$ outputs $\hat{g} \in \mathcal{G}$ such that $\pi(\hat{g}, F) \leq \epsilon$ with probability at least $1 - \delta$ as well as with time polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

Briefly speaking, the definition assumes that the target function g_* is within the learning model $\mathcal{G}$ and is of cost 0 (the minimum cost). In other words, it is the noiseless setup of learning. In the noisy case, we can use the following notion of agnostic PAC learning.

Definition 4.6. *In cost-sensitive classification, a learning model $\mathcal{G}$ is efficiently agnostic PAC-learnable (using the same representation class) if there exists a learning algorithm $\mathcal{A}$ satisfying the following property: for every distribution $dF(\mathbf{x}, y, \mathbf{c})$ being considered, where $\mathbf{c}[y] = c_{\min}$; for all $0 < \epsilon$ and $0 < \delta < \frac{1}{2}$, if $\mathcal{A}$ is given access to an oracle generating examples $(\mathbf{x}, y, \mathbf{c})$ from $dF(\mathbf{x}, y, \mathbf{c})$, as well as inputs ϵ and δ, then $\mathcal{A}$ outputs $\hat{g} \in \mathcal{G}$ satisfying $\pi(\hat{g}, F) - \pi(g_*, F) \leq \epsilon$ with probability at least $1 - \delta$ as well as with time polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.*

With Definitions 4.5 and 4.6, we now introduce the equivalence theorem.

Theorem 4.7 (Equivalence theorem of the reduction framework). *Consider a learning model $\mathcal{R}$ for ordinal ranking, its associated learning model $\mathcal{G} = \{g_r : r \in \mathcal{R}\}$ for binary classification, and distributions $dF(\mathbf{x}, y, \mathbf{c})$ such that all cost vectors $\mathbf{c}$ and effective cost vectors $\mathbf{c}_\mathbf{x}$ are V-shaped.*

1. *$\mathcal{R}$ is efficiently PAC-learnable if and only if $\mathcal{G}$ is efficiently PAC-learnable.*

2. *$\mathcal{R}$ is efficiently agnostic PAC-learnable if and only if $\mathcal{G}$ is efficiently agnostic PAC-learnable.*

Proof. If $\mathcal{G}$ is efficiently PAC-learnable using algorithm $\mathcal{A}_\mathcal{G}$, we can convert $\mathcal{A}_\mathcal{G}$ to an efficient algorithm $\mathcal{A}_\mathcal{R}$ for ordinal ranking.

1. Transform the oracle generating $(\mathbf{x}, y, \mathbf{c})$ from $dF(\mathbf{x}, y, \mathbf{c})$ to an oracle generating $\left(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\right)$ by (4.2).

2. Run $\mathcal{A}_\mathcal{G}$ with the transformed oracle until it outputs some $g\left(\mathbf{X}^{(k)}\right)$.

3. Return r_g.

It is not hard to see that $\mathcal{A}_\mathcal{R}$ is as efficient as $\mathcal{A}_\mathcal{G}$, and the cost guarantee comes from Theorems 4.1 and 4.3.

Now we consider the other direction. If $\mathcal{R}$ is efficiently PAC-learnable using algorithm $\mathcal{A}_\mathcal{R}$, we can convert $\mathcal{A}_\mathcal{R}$ to an efficient algorithm $\mathcal{A}_\mathcal{G}$ for binary classification.

1. Transform the oracle generating $\left(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\right)$ from $dF_b\left(\mathbf{X}^{(k)}, Y^{(k)}, W^{(k)}\right)$ to an oracle generating $(\mathbf{x}, y, \mathbf{c})$ by

$$
\begin{aligned}
\mathbf{x} &= \left(\mathbf{X}^{(k)}[1], \mathbf{X}^{(k)}[2], \ldots, \mathbf{X}^{(k)}[D]\right); \\
\mathbf{c} &= \begin{cases} \frac{W^{(k)}}{K-1} \cdot \left(\underbrace{0, \ldots, 0}_{k}, 1, \ldots, 1\right) & \text{for } Y^{(k)} = -1\,, \\ \frac{W^{(k)}}{K-1} \cdot \left(\underbrace{1, \ldots, 1}_{k}, 0, \ldots, 0\right) & \text{for } Y^{(k)} = +1\,; \end{cases} \\
y &= \operatorname*{argmin}_{1 \le \ell \le K} \mathbf{c}[\ell]\,.
\end{aligned}
$$

2. Run $\mathcal{A}_{\mathcal{R}}$ with the transformed oracle until it outputs some $r(\mathbf{x})$.

3. Return g_r.

We can easily see that $\mathcal{A}_{\mathcal{G}}$ is as efficient as $\mathcal{A}_{\mathcal{R}}$. Denote $dF_R(\mathbf{x}, y, \mathbf{c})$ as the probability measure described in step one. It is simple to prove that F_R with (4.2) would also introduce F_b. Then, by Lemma 4.2,

$$\pi(g_r, F_b) = \pi(r, F_R) \text{ for all } r \in \mathcal{R}.$$

Therefore, $\pi(g_r, F_b) < \epsilon$ after running $\mathcal{A}_{\mathcal{G}}$.

The proof above deals with the noiseless PAC-learnability result. Similar steps can be used to show the agnostic PAC-learnability result. □

Theorem 4.7 demonstrates that ordinal ranking is as easy (hard) as the associated binary classification problem. If we look at $g(\mathbf{x}, k)$ for one particular k, we see that the binary classification problem can be simplified to classifying all $(\mathbf{x}, y)$ with $y > k$ as $Y^{(k)} = +1$, and all $(\mathbf{x}, y)$ with $y \leq k$ as $Y^{(k)} = -1$. Recall that in Section 1.2, we discussed that ordinal ranking allows natural "ordinal comparison between different ranks." When the comparison between "all $\mathbf{x}$ with rank less than k" and "all $\mathbf{x}$ with rank at least k" is natural, it is simple for $\mathcal{A}_b$ to locate a decent $g(\mathbf{x}, k)$. In other words, the underlying binary classification problem is easy. On the other hand, if the comparison is not natural, such as the fruit categorization example in Section 1.2, both the "ordinal ranking" problem and the underlying binary classification are hard.

4.2 Usefulness of Reduction Framework

Li and Lin (2007b) unified existing algorithms under the reduction framework. We list them (and some new ones) as instances of the framework in Table 4.1. In this section, we discuss the last two rows of Table 4.1 further to demonstrate the algorithmic and theoretical usefulness of the reduction framework.

Table 4.1: Instances of the reduction framework

ordinal ranking	cost	binary classification algorithm
PRank (Crammer and Singer 2005)	absolute	modified perceptron rule
kernel ranking (Rajaram et al. 2003)	classification	modified hard-margin SVM
SVOR-EXP SVOR-IMC (Chu and Keerthi 2007)	classification absolute	modified soft-margin SVM
ORBoost-LR ORBoost-All (Section 3.2)	classification absolute	modified AdaBoost
oSVM oNN (Cardoso and da Costa 2007)	absolute absolute	standard soft-margin SVM standard neural network
RED-C4.5 RED-SVM (Li and Lin 2007b)	*any convex* *any convex*	standard C4.5 standard soft-margin SVM
AdaBoost-OR	*any V-shaped*	standard AdaBoost with reverse reduction

4.2.1 SVM for Ordinal Ranking

SVM is a popular binary classification algorithm (Schölkopf and Smola 2002; Vapnik 1995), which will be further introduced in Subsection 5.1.1. It maps the feature vector $\mathbf{x}$ to $\phi(\mathbf{x})$ in a possibly higher-dimensional space and implicitly computes the inner products with a kernel function

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle .$$

If we encode $(\mathbf{x}, k)$ by $(\mathbf{x}, -\gamma \mathbf{1}_k)$, we can then compute the inner products of the extended examples by

$$\mathcal{K}_E\big((\mathbf{x}, k), (\mathbf{x}', k')\big) = \langle (\phi(\mathbf{x}), k) , (\phi(\mathbf{x}'), k') \rangle = \mathcal{K}(\mathbf{x}, \mathbf{x}') + \gamma^2 \, [\![k = k']\!] .$$

With the reduction framework, we can plug in $\mathcal{K}_E$ and $O\big(NK\big)$ extended training

examples into the standard SVM to obtain an ranker

$$r(\mathbf{x}) = 1 + \sum_{k=1}^{K-1} [\![\langle \mathbf{v}, \phi(\mathbf{x}) \rangle + b - \theta_k > 0]\!] ,$$

based on an optimal solution to

$$\begin{aligned} \min_{\mathbf{v}, b, \theta_k, \xi_n^{(k)}} \quad & \frac{1}{2} \langle \mathbf{v}, \mathbf{v} \rangle + \frac{1}{2\gamma^2} \langle \theta, \theta \rangle + \kappa \sum_{n=1}^{N} \sum_{k=1}^{K-1} W_n^{(k)} \xi_n^{(k)} , \\ \text{subject to} \quad & Y_n^{(k)} \left(\langle \mathbf{v}, \phi(\mathbf{x}) \rangle + b - \theta_k \right) \geq 1 - \xi_n^{(k)}, \\ & \xi_n^{(k)} \geq 0, \text{ for } n = 1, \ldots, N, \text{ and } k = 1, \ldots, K-1. \end{aligned} \tag{4.8}$$

If $\theta_1 \leq \theta_2 \leq \ldots \leq \theta_{K-1}$, or if the cost vectors considered are convex, Theorems 4.1 and 4.3 can guarantee the expected out-of-sample cost of $r(\mathbf{x})$ based on the expected out-of-sample cost of the binary classifier

$$g(\mathbf{x}, k) = \text{sign}\Big(\langle \mathbf{v}, \phi(\mathbf{x}) \rangle + b - \theta_k \Big).$$

The oSVM approach of Cardoso and da Costa (2007) is an instance of (4.8) with the absolute cost, in which all $W_n^{(k)}$ are equal. The SVOR-IMC approach of Chu and Keerthi (2007) can also be thought as a modified instance of the formulation with the absolute cost, except that the $\frac{1}{2\gamma^2} \langle \theta, \theta \rangle$ term is dropped. Their SVOR-EXC approach is another modified instance using the classification cost plus an additional constraint to guarantee that $\theta_1 \leq \theta_2 \leq \ldots \leq \theta_{K-1}$.

RED-SVM unifies these algorithms under a generic formulation (4.8) with the reduction framework and allows us to deal with any convex cost vectors by changing $W_n^{(k)}$, or with any cost vectors by changing $W_n^{(k)}$ as well as respecting the constraint $\theta_1 \leq \theta_2 \leq \ldots \leq \theta_{K-1}$.[3]

Chu and Keerthi (2007) found that SVOR-EXP performed better in terms of the

[3]The additional constraint can be respected by a coordinate-descent procedure that switches between optimizing $(\mathbf{v}, b)$ (using the standard SVM solver) and optimizing θ under the constraints (a small quadratic programming problem with an analytic solution).

classification cost, and SVOR-IMC preceded in terms of the absolute cost. Their findings can be well explained through the reduction framework with the formulation above. Note that Chu and Keerthi (2007) paid much efforts in designing and implementing suitable optimizers for the modified formulation that does not contain the $\frac{1}{2\gamma^2}\langle\theta,\theta\rangle$ term. If we use the standard soft-margin SVM instead, we can directly and efficiently use the state-of-the-art SVM software to deal with the ordinal ranking problem. The formulation of Chu and Keerthi (2007) can be approximated by using a large γ. As we shall see in Subsection 4.3.1, even a simple assignment of $\gamma = 1$ performs similarly to the approaches of Chu and Keerthi (2007) in practice.

In addition to the algorithmic benefits described above, the reduction framework can also be used theoretically. For instance, we demonstrated how we can derive novel bounds of some common cost functions in Section 3.1. Next, we extend the bounds to SVM-based formulations and to a wider class of cost functions. While Shashua and Levin (2003) derived one such bound with a specific cost function, their bound is not data dependent and hence does not fully explain the out-of-sample performance of SVM-based rankers in reality (for more discussions on data-dependent bounds, see the work of, for example, Bartlett and Shawe-Taylor (1998)). Our bound, on the other hand, is not only more general, but also data dependent.

Theorem 4.8 (Large-margin bounds for SVM-based rankers). *Consider a collection*

$$\mathcal{F} = \left\{ f_{\mathbf{v},b,\theta}\left(\mathbf{X}^{(k)}\right) = \langle \mathbf{v}, \phi(\mathbf{x})\rangle + b - \theta_k, \text{ where } \|\mathbf{v}\|^2 + \|b-\theta\|^2 \leq 1, \|\phi(\mathbf{x})\|^2 + 1 \leq R^2 \right\}.$$

Let $B_{\max} = \max_{\mathbf{c}\in\mathcal{C}} (\mathbf{c}[1] + \mathbf{c}[K])$, $B_{\min} = \min_{\mathbf{c}\in\mathcal{C}} (\mathbf{c}[1] + \mathbf{c}[K])$, *and* $\beta = B_{\max}/B_{\min}$. *If* $\theta_1 \leq \theta_2 \leq \ldots \leq \theta_{K-1}$, *or if every* $\mathbf{c}$ *is convex, for any* $\Delta > 0$, *with probability at least* $1-\delta$, *and for every* f *in* $\mathcal{F}$, *the associated ranker*

$$r(\mathbf{x}) = 1 + \sum_{k=1}^{K-1} [\![f\left(\mathbf{X}^{(k)}\right) > 0]\!],$$

satisfies

$$\pi(r) \leq \frac{\beta}{N \cdot (K-1)} \sum_{n=1}^{N} \sum_{k=1}^{K-1} W_n^{(k)} \left[\!\left[Y_n^{(k)} f\left(\mathbf{X}_n^{(k)}\right) \leq \Delta \right]\!\right] + O\left(\frac{\log N}{\sqrt{N}}, \frac{R}{\Delta}, \sqrt{\log \frac{1}{\delta}} \right).$$

Proof. For every example $(\mathbf{x}, y, \mathbf{c})$, by the same derivation as Theorem 4.1,

$$\begin{aligned} & (K-1) \cdot \mathbf{c}[r(\mathbf{x})] \\ \leq & \sum_{k=1}^{K-1} W^{(k)} \left[\!\left[Y^{(k)} f\left(\mathbf{X}^{(k)}\right) \leq 0 \right]\!\right] \\ \leq & (K-1) \cdot (\mathbf{c}[1] + \mathbf{c}[K]) \cdot \sum_{k=1}^{K-1} \frac{W^{(k)}}{(K-1) \cdot (\mathbf{c}[1] + \mathbf{c}[K])} \left[\!\left[Y^{(k)} f\left(\mathbf{X}^{(k)}\right) \leq 0 \right]\!\right]. \end{aligned}$$

Note that

$$P^{(k)} = \frac{W^{(k)}}{(K-1) \cdot (\mathbf{c}[1] + \mathbf{c}[K])}$$

sums to 1. Then, for each example $(\mathbf{x}, y, \mathbf{c})$ obtained from $dF(\mathbf{x}, y, \mathbf{c})$, we can randomly choose k according to $P^{(k)}$ and form an unweighted binary example $\left(\mathbf{X}^{(k)}, Y^{(k)}\right)$. The procedure above defines a probability measure $dF_u\left(\mathbf{X}^{(k)}, Y^{(k)}\right)$. Integrating over all $(\mathbf{x}, y, \mathbf{c})$, we get

$$\pi(r) \leq B_{\max} \int_{\mathbf{X}^{(k)}, Y^{(k)}} \left[\!\left[Y^{(k)} f\left(\mathbf{X}^{(k)}\right) \leq 0 \right]\!\right] \, dF_u\left(\mathbf{X}^{(k)}, Y^{(k)}\right).$$

When each k_n is chosen independently according to $P_n^{(k)}$, we can generate N independent examples $\left(\mathbf{X}_n^{(k_n)}, Y_n^{(k_n)}\right)$ from $dF_u\left(\mathbf{X}^{(k)}, Y^{(k)}\right)$ from Z. Then, using a cost bound for SVM in binary classification (Bartlett and Shawe-Taylor 1998), with probability at least $(1 - \frac{\delta}{2})$ over the choice of $\left\{ \left(\mathbf{X}_n^{(k_n)}, Y_n^{(k_n)}\right) \right\}_{n=1}^{N}$,

$$\begin{aligned} & \int_{\mathbf{X}^{(k)}, Y^{(k)}} \left[\!\left[Y^{(k)} f\left(\mathbf{X}^{(k)}\right) \leq 0 \right]\!\right] \, dF_u\left(\mathbf{X}^{(k)}, Y^{(k)}\right) \\ \leq & \frac{1}{N} \sum_{n=1}^{N} \left[\!\left[Y_n^{(k_n)} f\left(\mathbf{X}_n^{(k_n)}\right) \leq \Delta \right]\!\right] + O\left(\frac{\log N}{\sqrt{N}}, \frac{R}{\Delta}, \sqrt{\log \frac{1}{\delta}} \right). \end{aligned}$$

Using the same technique as the proof of Theorem 3.2 with $b_n = \left[\!\left[Y_n^{(k_n)} f\left(\mathbf{X}_n^{(k_n)}\right) \leq \Delta \right]\!\right]$ and a union bound, with probability $> 1 - \delta$,

$$
\begin{aligned}
& \pi(r) \\
\leq\ & \frac{B_{\max}}{N} \sum_{n=1}^{N} \left[\!\left[Y_n^{(k_n)} f\left(\mathbf{X}_n^{(k_n)}\right) \leq \Delta \right]\!\right] + O\left(\frac{\log N}{\sqrt{N}}, \frac{R}{\Delta}, \sqrt{\log \frac{1}{\delta}} \right) \\
\leq\ & \frac{B_{\max}}{N} \sum_{n=1}^{N} \frac{1}{(K-1) \cdot (\mathbf{c}_n[1] + \mathbf{c}_n[K])} \sum_{k=1}^{K-1} W_n^{(k)} \cdot \left[\!\left[Y_n^{(k)} f\left(\mathbf{X}_n^{(k)}\right) \leq \Delta \right]\!\right] \\
& + O\left(\frac{\log N}{\sqrt{N}}, \frac{R}{\Delta}, \sqrt{\log \frac{1}{\delta}} \right) + O\left(\frac{1}{\sqrt{N}}, \sqrt{\log \frac{1}{\delta}} \right) \\
\leq\ & \frac{\beta}{N \cdot (K-1)} \sum_{n=1}^{N} \sum_{k=1}^{K-1} W_n^{(k)} \cdot \left[\!\left[Y_n^{(k)} f\left(\mathbf{X}_n^{(k)}\right) \leq \Delta \right]\!\right] + O\left(\frac{\log N}{\sqrt{N}}, \frac{R}{\Delta}, \sqrt{\log \frac{1}{\delta}} \right). \quad \square
\end{aligned}
$$

Thus, if f achieves large margins ($\geq \Delta$) on most of the extended training examples $\left(\mathbf{X}_n^{(k)}, Y_n^{(k)}, W_n^{(k)}\right)$, $\pi(r)$ is guaranteed to be small. A similar proof can be used to extend Theorem 3.2 and Corollary 3.4 to a more general class of cost functions.

4.2.2 AdaBoost for Ordinal Ranking

In Section 3.2, we introduced the ORBoost algorithm, which aggregates a set of binary classifiers (confidence functions) to perform ordinal ranking. Given the wide availability of ordinal ranking algorithms, can we use a boosting approach to aggregate rankers directly? Next, we use reduction and reverse reduction to design such an approach. We first introduce the ideas behind our approach. In the preprocessing stage, we apply the reduction method, and in the training stage, we take AdaBoost as the core binary classification algorithm. AdaBoost would then call a base algorithm to get a base binary classifier g_t with weighted binary examples in its t-th iteration. We use the reverse-reduction technique to replace g_t with g_{r_t} and let our approach train a ranker r_t with ordinal examples instead.

After the steps above, our approach would return an ensemble of rankers $U = \{(r_t, v_t)\}_{t=1}^{T}$, where $v_t \geq 0$ is the weight associated with the ranker r_t. In the prediction

stage, we first apply the reverse-reduction technique in (4.6) to cast each ranker r_t as a binary classifier $g_t = g_{r_t}$. The weighted votes from all the binary classifiers in the ensemble are gathered to form binary predictions. Then, the reduction method comes into play and constructs an ordinal prediction from the binary ones by (4.1). Combining the steps above, we get the following prediction rule for an ordinal ranking ensemble U:

$$r_U(\mathbf{x}) \equiv 1 + \sum_{k=1}^{K-1} \left[\!\!\left[\sum_{t=1}^{T} v_t \left[\!\left[k < r_t(\mathbf{x}) \right]\!\right] \geq \frac{1}{2} \sum_{t=1}^{T} v_t \right]\!\!\right]. \tag{4.9}$$

The steps of going back and forth between reduction and reverse reduction may seem complicated. Nevertheless, we can simplify many of them with careful derivations, which are illustrated below. We start with the prediction steps and derive a simplified form of (4.9) as follows.

Theorem 4.9. *For any ordinal ranking ensemble $U = \{(r_t, v_t)\}_{t=1}^{T}$ such that $v_t \geq 0$ and $\sum_{t=1}^{T} v_t = 1$,*

$$r_U(\mathbf{x}) \;\; = \;\; \min \left\{ k \colon \sum_{t=1}^{T} v_t \left[\!\left[k \geq r_t(\mathbf{x}) \right]\!\right] > \frac{1}{2} \right\}. \tag{4.10}$$

Proof. Let $k^* = \min \left\{ k \colon \sum_{t=1}^{T} v_t \left[\!\left[k \geq r_t(\mathbf{x}) \right]\!\right] > \frac{1}{2} \right\}$. Then, $\sum_{t=1}^{T} v_t \left[\!\left[k \geq r_t(\mathbf{x}) \right]\!\right] > \frac{1}{2}$ if and only if $k^* \leq k$. That is, $\sum_{t=1}^{T} v_t \left[\!\left[k < r_t(\mathbf{x}) \right]\!\right] \geq \frac{1}{2}$ if and only if $k < k^*$. Therefore, $r_U(\mathbf{x}) = 1 + k^* - 1 = k^*$. □

Thus, the seemly complicated prediction rule (4.9) can be equivalently performed in (4.10) by computing a simple and intuitive statistic: the weighted median. Note that the rule in (4.10) is not specific for our approach. It can be applied to ordinal ranking ensembles produced by any ensemble learning approaches, such as bagging (Breiman 1996).

We now look at the training steps. First, we list the steps of the original AdaBoost.

Algorithm 4.2 (AdaBoost, Freund and Schapire 1997).

1. *For a given training set $\tilde{Z} = \{(\tilde{\mathbf{x}}_m, \tilde{y}_m, \tilde{w}_m)\}_{m=1}^{M}$, initialize $\tilde{w}_m^{(1)} = \tilde{w}_m$ for all m.*

2. *For* $t = 1, 2, \ldots, T$,

   (a) *Obtain* $\tilde{g}_t$ *from the base binary classification algorithm* $\mathcal{A}_b$.

   (b) *Compute the weighted training error* $\tilde{\epsilon}_t$.

$$\tilde{\epsilon}_t = \left(\sum_{m=1}^{M} \tilde{w}_m^{(t)} \cdot [\![\tilde{y}_m \neq \tilde{g}_t(\tilde{\mathbf{x}}_m)]\!] \right) \Big/ \left(\sum_{m=1}^{M} \tilde{w}_m^{(t)} \right)$$

   If $\tilde{\epsilon}_t > \frac{1}{2}$, *set* $T = t - 1$ *and abort loop.*

   (c) *Let* $\tilde{v}_t = \frac{1}{2} \log \frac{\tilde{\epsilon}_t}{1 - \tilde{\epsilon}_t}$.

   (d) *Let* $\tilde{\Lambda}_t = \exp(2\tilde{v}_t) - 1$, *and*

$$\tilde{w}_m^{(t+1)} = \begin{cases} \tilde{w}_m^{(t)}, & \tilde{y}_m = \tilde{g}_t(\tilde{\mathbf{x}}_m) \ ; \\ \tilde{w}_m^{(t)} + \tilde{\Lambda}_t \tilde{w}_m^{(t)}, & \tilde{y}_m \neq \tilde{g}_t(\tilde{\mathbf{x}}_m) \ . \end{cases}$$

After plugging AdaBoost into reduction and a base ordinal ranking algorithm into reverse reduction, we can equivalently obtain the *AdaBoost for ordinal ranking* (AdaBoost.OR) algorithm below.

Algorithm 4.3 (AdaBoost.OR: AdaBoost for ordinal ranking).

1. *For a given training set* $Z = \{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^{N}$, *initialize* $\mathbf{c}_n^{(1)}[k] = \mathbf{c}_n[k]$ *for all* n *and* k.

2. *For* $t = 1, 2, \ldots, T$,

   (a) *Obtain* r_t *from the base ordinal ranking algorithm* $\mathcal{A}_r$.

   (b) *Compute the weighted training error* ϵ_t.

$$\epsilon_t = \left(\sum_{n=1}^{N} \mathbf{c}_n^{(t)}[r_t(\mathbf{x})] \right) \Big/ \left(\sum_{n=1}^{N} \mathbf{c}_n^{(t)}[1] + \mathbf{c}_n^{(t)}[K] \right)$$

   If $\epsilon_t > \frac{1}{2}$, *set* $T = t - 1$ *and abort loop.*

   (c) *Let* $v_t = \frac{1}{2} \log \frac{\epsilon_t}{1 - \epsilon_t}$.

(d) Let $\Lambda_t = \exp(2v_t) - 1$. *If* $r_t(\mathbf{x}_n) \geq y_n$, *then*

$$\mathbf{c}_n^{(t+1)}[k] = \begin{cases} \mathbf{c}_n^{(t)}[k]\,, & k \leq y_n\ ; \\ \mathbf{c}_n^{(t)}[k] + \Lambda_t \cdot \mathbf{c}_n^{(t)}[k]\,, & y_n < k \leq r_t(\mathbf{x}_n)\ ; \\ \mathbf{c}_n^{(t)}[k] + \Lambda_t \cdot \mathbf{c}_n^{(t)}[r_t(\mathbf{x}_n)]\,, & k > r_t(\mathbf{x}_n)\ . \end{cases}$$

Otherwise switch $>$ *to* $<$ *and vice versa.*

The connection between Algorithms 4.2 and 4.3 is based on maintaining the following invariance in each iteration.

Lemma 4.10. *Substitute the indices* m *in Algorithm 4.2 with* (n, k). *That is,*

$$\tilde{\mathbf{x}}_m = \mathbf{X}_n^{(k)}, \tilde{y}_m = \tilde{y}_{nk} = Y_n^{(k)}, \text{ and } \tilde{w}_m = \tilde{w}_{nk} = W_n^{(k)}.$$

Take $\tilde{g}_t(\mathbf{x}, k) = g_{r_t}(\mathbf{x}, k)$ *and assume that in Algorithms 4.2 and 4.3,*

$$\mathbf{c}_n^{(\tau)}[k] = \sum_{\ell=1}^{K-1} \frac{\tilde{w}_{n\ell}^{(\tau)}}{K-1} \cdot [\![y_n \leq \ell < k \text{ or } k < \ell \leq y_n]\!] \tag{4.11}$$

is satisfied for $\tau = t$ *with* $\tilde{w}_{n\ell}^{(\tau)} \geq 0$. *Then, equation* (4.11) *is satisfied for* $\tau = t + 1$ *with* $\tilde{w}_{n\ell}^{(\tau)} \geq 0$.

Proof. Because (4.11) is satisfied for $\tau = t$ and $\tilde{w}_{n\ell}^{(t)} \geq 0$, the cost vector $\mathbf{c}_n^{(t)}$ is V-shaped with respect to y_n and $\mathbf{c}_n^{(t)}[y_n] = 0$. Thus,

$$\sum_{n=1}^{N} \left(\mathbf{c}_n^{(t)}[1] + \mathbf{c}_n^{(t)}[K] \right) = \sum_{n=1}^{N} \sum_{k=1}^{K-1} \tilde{w}_{nk}^{(t)}.$$

In addition, since $\tilde{g}_t(\mathbf{x}, k) = g_{r_t}(\mathbf{x}, k)$, by a proof similar to Lemma 4.2,

$$\sum_{n=1}^{N} \mathbf{c}_n^{(t)}[r_t(\mathbf{x})] = \sum_{n=1}^{N} \sum_{k=1}^{K-1} \tilde{w}_{nk}^{(t)} \cdot [\![y_{nk} \neq \tilde{g}_t(\mathbf{x}_n, k)]\!]\,.$$

Therefore, $\tilde{\epsilon}_t = \epsilon_t$, $\tilde{v}_t = v_t$, and $\tilde{\Lambda}_t = \Lambda_t$.

Because $\tilde{g}_t(\mathbf{x}_n, k) \neq y_{nk}$ if and only if $r_t(\mathbf{x}_n) \leq k < y_n$ or $y_n < k \leq r_t(\mathbf{x}_n)$,

$$\tilde{w}_{nk}^{(t+1)} = \begin{cases} \tilde{w}_{nk}^{(t)} + \tilde{\Lambda}_t \tilde{w}_{nk}^{(t+1)}, & y_n < k \leq r_t(\mathbf{x}_n) \text{ or } y_n < k \leq r_t(\mathbf{x}_n) \ ; \\ \tilde{w}_{nk}^{(t)}, & \text{otherwise} \ . \end{cases} \tag{4.12}$$

It is easy to check that $\tilde{w}_{nk}^{(t+1)}$ are nonnegative. Furthermore, we see that the update rule in Algorithm 4.3 is equivalent to combining (4.12) and (4.11) with $\tau = t + 1$. Thus, equation (4.11) is satisfied for $\tau = t + 1$. □

Then, by mathematical induction from $\tau = 1$ up to T with Lemma 4.10, plugging AdaBoost into reduction and a base ordinal ranking algorithm into reverse reduction is equivalent to running AdaBoost.OR with the base algorithm. AdaBoost.OR takes AdaBoost as a special case of $K = 2$. It can use any base ordinal ranking algorithm that produces individual rankers r_t with errors $\epsilon_t \leq \frac{1}{2}$. In binary classification, the $\frac{1}{2}$ error bound can be naturally achieved by a constant classifier or a fair coin flip. For ordinal ranking, is $\frac{1}{2}$ still easy to achieve? The short answer is yes. In the following theorem, we demonstrate that there always exists a constant ranker that satisfies the error bound.

Theorem 4.11. *Define constant rankers $r^{(k)}$ by $r^{(k)}(\mathbf{x}) \equiv k$ for all $\mathbf{x}$. For any set $\{\mathbf{c}_n\}_{n=1}^N$, there exists a constant ranker with $1 \leq k \leq K$ such that*

$$\epsilon^{(k)} = \left(\sum_{n=1}^{N} \mathbf{c}_n \left[r^{(k)}(\mathbf{x}) \right] \right) \Big/ \left(\sum_{n=1}^{N} \mathbf{c}_n[1] + \mathbf{c}_n[K] \right) \leq \frac{1}{2}.$$

Proof. Either $r^{(1)}$ or $r^{(K)}$ achieves error $\leq \frac{1}{2}$ because by definition $\epsilon^{(1)} + \epsilon^{(K)} = 1$. □

Therefore, even the simplest deterministic rankers can always achieve the desired error bound.[4] If the base ordinal ranking algorithm produces better rankers, the following theorem bounds the normalized training cost of the final ensemble U.

[4]Similarly, the error bound can be achieved by a randomized ordinal ranker that returns either 1 or K with equal probability.

Theorem 4.12. *Suppose the base ordinal ranking algorithm produces rankers with errors $\epsilon_1, \ldots, \epsilon_T$, where each $\epsilon_t \leq \frac{1}{2}$. Let $\gamma_t = \frac{1}{2} - \epsilon_t$, the final ensemble r_U satisfies the following error bound:*

$$\frac{N}{\sum_{n=1}^{N} \mathbf{c}_n[1] + \mathbf{c}_n[K]} \cdot \nu(r_U) \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq \exp\Big(-2 \sum_{t=1}^{T} \gamma_t^2\Big).$$

Proof. Similar to the proof for Lemma 4.10, the left-hand side of the bound equals

$$\left(\sum_{n=1}^{N} \sum_{k=1}^{K-1} w_{nk} \left[\!\left[y_{nk} \neq g_{\tilde{U}}(\mathbf{x}_n, k) \right]\!\right] \right) \Big/ \left(\sum_{n=1}^{N} \sum_{k=1}^{K-1} w_{nk} \right) ,$$

where $\tilde{U}$ is a binary classification ensemble $\{(\tilde{g}_t, v_t)\}_{t=1}^{T}$ with $\tilde{g}_t = g_{r_t}$. Then, the bound is a simple consequence of the well-known AdaBoost bound (Freund and Schapire 1997). □

Theorem 4.12 indicates that if the base algorithm always produces a ranker with $\epsilon_t \leq \frac{1}{2} - \gamma$ for $\gamma > 0$, the training cost of U would decrease exponentially with T. That is, AdaBoost.OR can rapidly boost the training performance of such a base algorithm.

Similar to Theorems 3.2 and 4.8, we can also use reduction to extend the out-of-sample cost bounds of AdaBoost to AdaBoost.OR, including the iteration-based bound (Freund and Schapire 1997) and the margin-based ones (Schapire et al. 1998).

4.3 Experiments

In this section, we first compare SVM-based ordinal ranking algorithms. We also compare them with the cost-sensitive classification algorithms discussed in Chapter 2. Then, we demonstrate the ability of AdaBoost.OR to boost the training and test performance of base ordinal ranking algorithms.

Table 4.2: Test absolute cost of SVM-based ordinal ranking algorithms

data set	RED-SVM perceptron	SVOR-IMC Gaussian
pyrimdines	**1.304±0.040**	**1.294±0.046**
machine	**0.842±0.022**	0.990±0.026
boston	**0.732±0.013**	0.747±0.011
abalone	1.383±0.004	**1.361±0.003**
bank	1.404±0.002	**1.393±0.002**
computer	**0.565±0.002**	0.596±0.002
california	**0.940±0.001**	1.008±0.001
census	**1.143±0.002**	1.205±0.002

(those within one standard error of the lowest one are marked in bold)

Table 4.3: Test classification cost of SVM-based ordinal ranking algorithms

data set	RED-SVM perceptron	SVOR-EXC Gaussian
pyrimdines	**0.762±0.021**	**0.752±0.014**
machine	**0.572±0.013**	0.661±0.012
boston	**0.541±0.009**	0.569±0.006
abalone	**0.721±0.002**	0.736±0.002
bank	0.751±0.001	**0.744±0.001**
computer	**0.451±0.002**	0.462±0.001
california	**0.613±0.001**	0.640±0.001
census	**0.688±0.001**	0.699±0.000

(those within one standard error of the lowest one are marked in bold)

4.3.1 SVM for Ordinal Ranking

We perform experiments on our proposed RED-SVM algorithms with the same eight benchmark data sets from Chu and Keerthi (2007) and the same setup as we did in Subsections 2.4.2 and 3.3.2. The γ parameter in (4.8) is fixed to 1. Similar to the SVM-based approaches in Subsection 2.4.2, the κ parameter is chosen within $\{2^{-17}, 2^{-15}, \ldots, 2^{3}\}$ using a 5-fold CV procedure on the training set (Hsu, Chang and Lin 2003).

Table 4.2 compares our proposed RED-SVM algorithm with the perceptron kernel with the SVOR-IMC results listed by Chu and Keerthi (2007) using the mean absolute cost (and the standard error), and Table 4.3 compares our algorithm with their SVOR-

EXC results using the mean classification cost. We can see that our proposed RED-SVM can often perform significantly better than the SVOR algorithms in both tables.

Note, however, that Chu and Keerthi (2007) uses the Gaussian kernel rather than the perceptron kernel in their experiments. For a fair comparison, we implemented their SVOR-IMC algorithm with the perceptron kernel by modifying LIBSVM (Chang and Lin 2001) and conduct experiments with the same parameter selection procedure.[5] With the same perceptron kernel, we compare RED-SVM with SVOR-IMC in Table 4.4. We see that our direct reduction to the standard SVM (RED-SVM) performs similarly to SVOR-IMC. In other words, the change from the Gaussian kernel to the perceptron kernel explains most of the performance differences between the columns of Tables 4.2 and 4.3. Our RED-SVM, nevertheless, is much easier to implement. In addition, RED-SVM is significantly faster than SVOR-IMC in training, which is illustrated in Figure 4.2 using the four largest data sets.[6] The main cause to the time difference is the speed-up heuristics. While, to the best of our knowledge, not much has been done to improve the original SVOR-IMC algorithm, plenty of heuristics, such as shrinking and advanced working selection in LIBSVM, can be seamlessly adopted by RED-SVM because of the reduction framework. The difference demonstrate an important advantage of the reduction framework: Any improvements to the binary classification approaches can be immediately inherited by reduction-based ordinal ranking algorithms.

Tables 4.5 and 4.6 compares RED-SVM to CSOVA and CSOVO using SVM with perceptron kernel as the underlying binary classification algorithm. We conduct experiments on both the eight data sets for ordinal ranking and the six data sets for classification. We can see a clear difference between the proposed cost-sensitive classification algorithms and the proposed ordinal ranking algorithms. For ordinal ranking data sets, RED-SVM enjoys an advantage, even in the classification cost setup. Such a result justifies our final arguments in Section 4.1: When ordinal ranking allows

[5] We only focus on SVOR-IMC because it is more difficult to implement SVOR-EXC with LIBSVM and to compare it fairly to RED-SVM.

[6] We gathered the CPU time on a 1.7G Dual Intel Xeon machine with 1GB RAM.

Table 4.4: Test absolute cost of SVM-based ordinal ranking algorithms with the perceptron kernel

data set	RED-SVM	SVOR-IMC
pyrimdines	**1.304±0.040**	**1.315±0.039**
machine	0.842±0.022	**0.814±0.019**
boston	**0.732±0.013**	**0.729±0.013**
abalone	**1.383±0.004**	**1.386±0.005**
bank	**1.404±0.002**	**1.404±0.002**
computer	**0.565±0.002**	**0.565±0.002**
california	**0.940±0.001**	**0.939±0.001**
census	**1.143±0.002**	**1.143±0.002**

(those within one standard error of the lowest one are marked in bold)

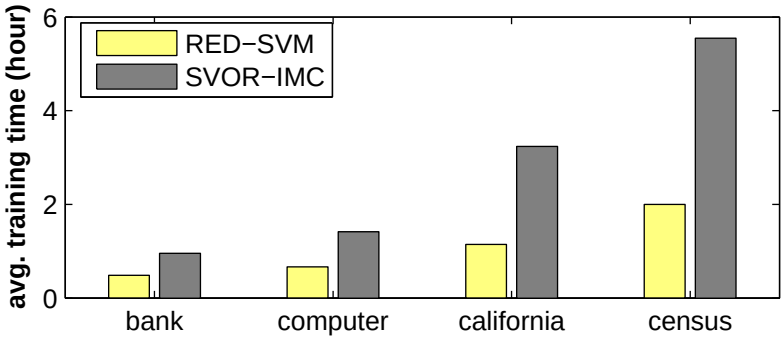


Figure 4.2: Training time (including automatic parameter selection) of SVM-based ordinal ranking algorithms with the perceptron kernel

natural ordinal comparison between different ranks, it can be easily solved via our reduction framework to binary classification. On the other hand, when using the classification data sets, in which there is no natural comparison between different ranks, the reduction framework would encounter hard binary classification problems. Then, cost-sensitive classification algorithms like CSOVA and CSOVO can perform better than RED-SVM.

4.3.2 AdaBoost for Ordinal Ranking

We now demonstrate the validity of AdaBoost.OR. We will first illustrate its behavior on an artificial data set. Then, we test its training and test performance on the eight ordinal ranking data sets.

Table 4.5: Test absolute cost of all SVM-based algorithms

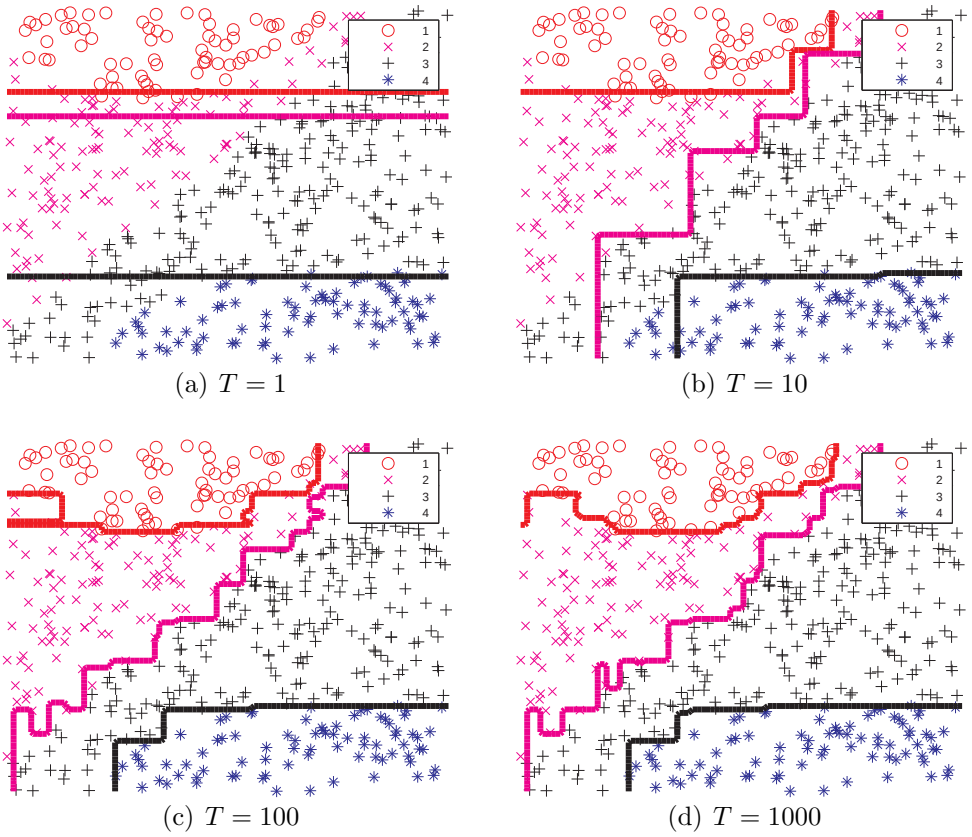
data set	CSOVA	CSOVO	RED-SVM
pyrimdines	1.627±0.055	**1.337±0.054**	**1.304±0.040**
machineCPU	0.975±0.024	**0.842±0.023**	**0.842±0.022**
boston	0.946±0.017	0.789±0.015	**0.732±0.013**
abalone	1.674±0.007	1.422±0.006	**1.383±0.004**
bank	1.801±0.004	1.414±0.003	**1.404±0.002**
computer	0.644±0.003	0.575±0.002	**0.565±0.002**
california	1.121±0.002	0.951±0.002	**0.940±0.001**
census	1.329±0.003	**1.135±0.001**	1.143±0.002
vehicle	**0.226±0.007**	**0.225±0.007**	0.282±0.006
vowel	**0.030±0.005**	**0.030±0.005**	0.331±0.009
segment	**0.043±0.003**	**0.045±0.003**	0.082±0.003
dna	**0.054±0.002**	0.067±0.002	0.178±0.003
satimage	**0.123±0.003**	0.127±0.003	0.192±0.002
usps	**0.077±0.002**	0.089±0.002	0.294±0.003

(those within one standard error of the lowest one are marked in bold)

Table 4.6: Test classification cost of all SVM-based algorithms

data set	CSOVA (OVA)	CSOVO (OVO)	RED-SVM
pyrimdines	**0.750±0.015**	0.792±0.018	**0.762±0.021**
machine	0.608±0.012	0.612±0.012	**0.572±0.013**
boston	0.614±0.004	0.583±0.007	**0.541±0.009**
abalone	0.735±0.002	0.726±0.002	**0.721±0.002**
bank	0.767±0.001	**0.750±0.001**	**0.751±0.001**
computer	0.502±0.001	0.468±0.001	**0.451±0.002**
california	0.631±0.001	**0.611±0.001**	0.613±0.001
census	0.692±0.001	**0.674±0.001**	0.688±0.001
vehicle	0.191±0.005	**0.185±0.005**	0.265±0.006
vowel	0.015±0.002	**0.011±0.002**	0.225±0.006
segment	**0.024±0.001**	**0.024±0.001**	0.070±0.002
dna	**0.040±0.002**	0.043±0.002	0.168±0.002
satimage	**0.071±0.002**	**0.072±0.002**	0.161±0.001
usps	**0.022±0.000**	0.023±0.000	0.218±0.002

(those within one standard error of the lowest one are marked in bold)



(a) $T = 1$ (b) $T = 10$

(c) $T = 100$ (d) $T = 1000$

Figure 4.3: Decision boundaries produced by AdaBoost.OR on an artificial data set

We first generate 500 input vectors $\mathbf{x}_n \in [0, 1] \times [0, 1]$ uniformly and rank them with $\{1, 2, 3, 4\}$ based on three quadratic boundaries. Then, we apply AdaBoost.OR on these examples with the absolute cost setup.

We use a simple base ordinal ranking algorithm called ORStump, which solves the following optimization problem efficiently with essentially the same dynamic programming technique used in RankBoost-OR (Subsection 3.2.1):

$$\begin{aligned} \min_{\theta_k, d, q} \quad & \sum_{n=1}^{N} \mathbf{c}_n[r(\mathbf{x}_n, \theta, d, q)] \ , \\ \text{subject to} \quad & \theta_1 \leq \theta_2 \leq \ldots \leq \theta_{K-1}, \\ \text{where} \quad & r(\mathbf{x}, \theta, d, q) \equiv \max\{k \colon q \cdot \mathbf{x}[d] < \theta_k\} \, . \end{aligned}$$

The ordinal ranking decision stump $r(\cdot, \theta, d, q)$ is a natural extension of the binary

decision stump (Holte 1993). Note that the set of all possible ordinal ranking decision stumps includes constant rankers. Therefore, ORStump can always achieve $\epsilon_t \leq \frac{1}{2}$.

The decision boundaries generated by AdaBoost.OR with ORStump using $T = 1, 10, 100$, and 1000 are shown in Figure 4.3. The case of $T = 1$ is the same as applying ORStump directly on the artificial set, and we can see that its resulting decision boundary cannot capture the full characteristic of the data. As T gets larger, however, AdaBoost.OR is able to boost up ORStump to form more sophisticated boundaries that approximate the underlying quadratic curves better.

Next, we run AdaBoost.OR on the eight benchmark data sets. We couple AdaBoost.OR with two base ordinal ranking algorithms: ORStump and PRank (Crammer and Singer 2005). For PRank, we adopt the SiPrank variant and make it cost-sensitive by presenting random examples $(\mathbf{x}_n, y_n)$ with probability proportional to $\max_{1 \leq k \leq K} \mathbf{c}_n[k]$. In addition, we apply the pocket technique with ratchet (Gallant 1990) for 2000 epochs to get a decent training cost minimizer.

We run AdaBoost.OR for $T = 1000$ iterations for ORStump, and 100 for PRank. Such a setup is intended to compensate the computational complexity of each individual base ordinal ranking algorithm. Nevertheless, a more sophisticated choice of T should further improve the performance of AdaBoost.OR.

For each algorithm, the mean training absolute cost as well as its standard error is reported in Table 4.7; the mean test absolute cost and its standard error is reported in Table 4.8. For each pair of single and AdaBoost.OR entries, we mark the one with the lowest cost in bold.

From the tables, we see that AdaBoost.OR almost always boosts both the training and test performance of the base ordinal ranking algorithm significantly. Note, however, that it is harder for AdaBoost.OR to improve the performance of PRank, because PRank sometimes cannot produce a good r_t in terms of minimizing the training cost.

Table 4.7: Training absolute cost of base and AdaBoost.OR algorithms

data set	ORStump		PRank	
	single	AdaBoost.OR	single	AdaBoost.OR
pyrimdines	1.757±0.017	**0.024±0.007**	0.457±0.029	**0.268±0.048**
machine	1.118±0.015	**0.122±0.009**	0.880±0.011	**0.864±0.010**
boston	1.049±0.010	**0.000±0.000**	0.845±0.009	**0.831±0.008**
abalone	1.528±0.008	**1.048±0.008**	1.439±0.010	**1.437±0.010**
bank	1.975±0.005	**1.141±0.004**	1.514±0.003	**1.467±0.003**
computer	1.178±0.003	**0.499±0.002**	0.659±0.003	**0.658±0.002**
california	1.615±0.004	**0.883±0.004**	**1.205±0.004**	1.205±0.004
census	1.826±0.002	**1.113±0.004**	1.582±0.008	**1.562±0.006**

(the lowest one among the two using the same base algorithm is marked in bold)

Table 4.8: Test absolute cost of base and AdaBoost.OR algorithms

data set	ORStump		PRank	
	single	AdaBoost.OR	single	AdaBoost.OR
pyrimdines	1.913±0.087	**1.244±0.051**	1.569±0.070	**1.417±0.066**
machine	1.286±0.040	**0.842±0.020**	0.969±0.012	**0.932±0.022**
boston	1.172±0.013	**0.887±0.014**	0.906±0.012	**0.892±0.011**
abalone	1.592±0.003	**1.475±0.004**	1.477±0.009	**1.475±0.008**
bank	2.000±0.003	**1.530±0.003**	1.540±0.004	**1.502±0.004**
computer	1.200±0.003	**0.627±0.002**	0.661±0.003	**0.660±0.003**
california	1.636±0.001	**0.995±0.003**	**1.206±0.002**	1.206±0.003
census	1.851±0.001	**1.253±0.002**	1.598±0.005	**1.578±0.003**

(the lowest one among the two using the same base algorithm is marked in bold)

Chapter 5

Studies on Binary Classification

In Chapter 4, we proved that ordinal ranking is PAC-learnable if and only if binary classification is PAC-learnable. In other words, under the PAC setup, if we want to have a good learning algorithm (and learning model) for ordinal ranking, it is necessary and sufficient to design a good learning algorithm for binary classification. In this chapter, we discuss two projects that aim at understanding and improving binary classification in the context of ensemble learning. The first one identifies some restrictions of AdaBoost (Algorithm 4.2) and resolves them with the help of SVM. The second one, on the other hand, focuses on a particular advantage of AdaBoost, and uses the advantage to improve other learning algorithms. The findings in the projects reveal the relative strength and weakness of AdaBoost and SVM, two of the most important binary classification algorithms.

5.1 SVM for Infinite Ensemble Learning

Recall that we proposed the threshold ensemble model for ordinal ranking in Chapter 3. The model originates from the ensemble model for binary classification (Meir and Rätsch 2003), which is accompanied by many successful algorithms such as bagging (Breiman 1996) and AdaBoost (Freund and Schapire 1997). The algorithms construct a classifier that averages over some base hypotheses in a set $\mathcal{H}$. While the size of $\mathcal{H}$ can be infinite, most existing algorithms use only a finite subset of $\mathcal{H}$, and the classifier is effectively a finite ensemble of hypotheses. Some theories show

that the finiteness places a restriction on the capacity of the ensemble (Freund and Schapire 1997), and some theories suggest that the performance of AdaBoost can be linked to its asymptotic behavior when the ensemble is allowed to be of an infinite size (Rätsch, Onoda and Müller 2001). Thus, it is possible that an infinite ensemble is superior for learning. Nevertheless, the possibility has not been fully explored because constructing such an ensemble is a challenging task (Vapnik 1998).

Next, we discuss how we conquer the task of infinite ensemble learning and demonstrate that better performance can be achieved by going from finite ensembles to infinite ones. In particular, we formulate a framework for infinite ensemble learning using SVM (Lin 2005; Lin and Li 2008). The key of the framework is to embed an infinite number of hypotheses into an SVM kernel. Such a framework can be applied both to construct new kernels for SVM and to interpret some existing ones (Lin 2005; Lin and Li 2008). Furthermore, the framework allows us to compare SVM and AdaBoost in a fair manner using the same base hypothesis set. Experimental results show that SVM with these kernels is superior to AdaBoost with the same base hypothesis set and help understand both SVM and AdaBoost better.

5.1.1 SVM and Ensemble Learning

Before introducing our framework, we first review the connections between SVM and ensemble learning in literature. Consider a binary classification problem where the labels $y \in \{-1, +1\}$, SVM (Vapnik 1995) obtains a decision function

$$\hat{g}(\mathbf{x}) = \operatorname{sign}\Big(\langle \mathbf{v}, \phi(\mathbf{x}) \rangle + b\Big)$$

from the optimal solution to the following problem:

$$\begin{aligned} \min_{\mathbf{v},b,\xi_n} \quad & \frac{1}{2}\langle \mathbf{v}, \mathbf{v} \rangle + \kappa \sum_{n=1}^{N} \xi_n \,, & & (5.1) \\ \text{subject to} \quad & y_n\big(\langle \mathbf{v}, \phi(\mathbf{x}_n) \rangle + b\big) \geq 1 - \xi_n, & & \text{for } n = 1, 2, \ldots, N, \\ & \xi_n \geq 0, & & \text{for } n = 1, 2, \ldots, N. \end{aligned}$$

Here $\kappa > 0$ is the regularization parameter, and ϕ is a feature mapping from $\mathcal{X}$ to a Hilbert space $\mathcal{F}$ (Schölkopf and Smola 2002). Because $\mathcal{F}$ can be of an infinite number of dimensions, SVM solvers usually work on the dual problem:

$$\begin{aligned} \min_{\lambda_n} \quad & \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N}\lambda_m\lambda_n y_m y_n \cdot \mathcal{K}(\mathbf{x}_m, \mathbf{x}_n) - \sum_{n=1}^{N}\lambda_n \,, \\ \text{subject to} \quad & 0 \le \lambda_n \le \kappa, \qquad \text{for } n = 1, 2, \ldots, N, \\ & \sum_{n=1}^{N} y_n \lambda_n = 0. \end{aligned} \tag{5.2}$$

Here $\mathcal{K}$ is the kernel function defined by $\mathcal{K}(\mathbf{x}, \mathbf{x}') \equiv \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$. Then, the optimal classifier becomes

$$\hat{g}(\mathbf{x}) = \operatorname{sign}\left(\sum_{n=1}^{N} y_n \lambda_n \mathcal{K}(\mathbf{x}_n, \mathbf{x}) + b \right), \tag{5.3}$$

where b can be computed through the primal-dual relationship (Schölkopf and Smola 2002; Vapnik 1998).

It is known that SVM is connected to AdaBoost (Demiriz, Bennett and Shawe-Taylor 2002; Freund and Schapire 1999; Rätsch et al. 2002; Rätsch, Onoda and Müller 2001). Recall that AdaBoost (Algorithm 4.2) iteratively selects T hypotheses $h_t \in \mathcal{H}$ and weights $v_t \ge 0$ to construct an ensemble classifier $H_T(\mathbf{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} v_t h_t(\mathbf{x})\right)$. Under some assumptions (Rätsch, Onoda and Müller 2001), it is shown that when $T \to \infty$, AdaBoost asymptotically approximates an infinite ensemble classifier $H_\infty(\mathbf{x})$ such that $\{(v_t, h_t)\}_{t=1}^{\infty}$ is an optimal solution to

$$\begin{aligned} \min_{v_t, h_t} \quad & \sum_{t=1}^{\infty} v_t \,, \\ \text{subject to} \quad & y_n \left(\sum_{t=1}^{\infty} v_t h_t(\mathbf{x}_n) \right) \ge 1, \qquad \text{for } n = 1, 2, \ldots, N, \\ & v_t \ge 0, \qquad \text{for } t = 1, 2, \ldots, \infty. \end{aligned} \tag{5.4}$$

Comparing (5.4) with (5.1) plus the feature mapping

$$\phi(\mathbf{x}) = \big(h_1(\mathbf{x}), h_2(\mathbf{x}), \dots\big), \tag{5.5}$$

we see that the elements of $\phi(\mathbf{x})$ in SVM are similar to the hypotheses $h_t(\mathbf{x})$ in AdaBoost. They both work on linear combinations of these elements, though SVM deals with an additional intercept term b. SVM minimizes the ℓ_2-norm of the weights while AdaBoost works on the ℓ_1-norm. SVM introduces slack variables ξ_n and use the parameter κ for regularization, while AdaBoost relies on the choice of the parameter T (Rosset, Zhu and Hastie 2004). Note that AdaBoost requires $v_t \geq 0$ for ensemble learning.

Let us take a deeper look at (5.4), which contains infinitely many variables. In order to approximate the optimal solution well with a fixed and finite T, AdaBoost resorts to two related properties of some of the optimal solutions for (5.4): finiteness and sparsity.

Finiteness: When two hypotheses share the same prediction patterns on the training input vectors, they can be used interchangeably during the training time and are thus *ambiguous*. Since there are at most 2^N prediction patterns on N training input vectors, we can partition $\mathcal{H}$ into at most 2^N groups, each of which contains mutually ambiguous hypotheses. Some optimal solutions of (5.4) only assign one or a few nonzero weights within each group (Demiriz, Bennett and Shawe-Taylor 2002). Thus, it is possible to work on a finite data-dependent subset of $\mathcal{H}$ instead of $\mathcal{H}$ itself without losing optimality.

Sparsity: Minimizing the ℓ_1-norm $\|\mathbf{v}\|_1 = \sum_{t=1}^{\infty} |v_t|$ often leads to sparse solutions (Meir and Rätsch 2003; Rosset et al. 2007). That is, for hypotheses in the finite (but possibly still large) subset of $\mathcal{H}$, only a small number of weights needs to be nonzero. AdaBoost can be viewed as a greedy search algorithm that approximates such a finite and sparse ensemble (Rosset, Zhu and Hastie 2004).

Although there exist some good algorithms that can return an optimal solution

of (5.4) when $\mathcal{H}$ is infinitely large (Rätsch, Demiriz and Bennett 2002; Rosset et al. 2007), the resulting ensemble relies on the sparsity property and is effectively of only finite size. Thus, it is possible that the learning performance could be further improved if either or both the finiteness and the sparsity restrictions are removed. The possibility motivates us to study the task of infinite ensemble learning, as discussed next.

5.1.2 Infinite Ensemble Learning

Vapnik (1998) proposed the challenging task of designing an algorithm that is not limited by the finiteness restriction. In particular, the algorithm should be able to generate an infinite ensemble classifier (an ensemble classifier with infinitely many nonzero v_t). Traditional algorithms like AdaBoost cannot be directly generalized to solve the task, because they select the hypotheses in an iterative manner and only run for a finite number of iterations.

We conquer the challenge via another route: the connection between SVM and ensemble learning. We start by embedding the infinite number of hypotheses in $\mathcal{H}$ into an SVM kernel. We have shown in (5.5) that we could construct a feature mapping from $\mathcal{H}$. The idea is extended to a more general form for deriving a kernel in Definition 5.1.

Definition 5.1 (Lin and Li 2008). *Assume that $\mathcal{H} = \{h_\alpha \colon \alpha \in \mathcal{W}\}$, where $\mathcal{W}$ is a measure space. The kernel that embodies $\mathcal{H}$ is defined as*

$$\mathcal{K}_{\mathcal{H},\mu}(\mathbf{x}, \mathbf{x}') = \int_{\mathcal{W}} \phi_\alpha(\mathbf{x})\phi_\alpha(\mathbf{x}') \, d\alpha, \tag{5.6}$$

where $\phi_\alpha(\mathbf{x}) = \mu(\alpha)h_\alpha(\mathbf{x})$, and $\mu \colon \mathcal{W} \to \mathbb{R}^+$ is chosen such that the integral exists for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$.

Here α is the parameter of the hypothesis h_α. We shall denote $\mathcal{K}_{\mathcal{H},\mu}$ by $\mathcal{K}_{\mathcal{H}}$ when μ is clear from the context. The kernel $\mathcal{K}_{\mathcal{H}}$ embodies the predictions of all $h_\alpha \in \mathcal{H}$ with the integral and could handle the situation even when $\mathcal{H}$ is uncountable. When we

use $\mathcal{K}_{\mathcal{H}}$ in (5.2), the classifier obtained is equivalent to

$$\hat{g}(\mathbf{x}) = \operatorname{sign}\left(\int_{\mathcal{W}} v(\alpha)\mu(\alpha)h_\alpha(\mathbf{x})d\alpha + b\right). \tag{5.7}$$

Nevertheless, $\hat{g}$ is not an ensemble classifier yet, because we do not have the constraints $v(\alpha) \geq 0$, and we have an additional term b. Next, we would explain that such a classifier is equivalent to an ensemble classifier under some reasonable assumptions.

We start from the constraints $v(\alpha) \geq 0$, which cannot be directly considered in (5.1). Vapnik (1998) showed that even if we add a countably infinite number of constraints to (5.1), infinitely many variables and constraints would be introduced to (5.2). Then, the latter problem would still be difficult to solve.

One remedy is to assume that $\mathcal{H}$ is *negation complete*, that is,[1]

$$h \in \mathcal{H} \Leftrightarrow (-h) \in \mathcal{H}.$$

Then, every linear combination over $\mathcal{H}$ can be easily transformed to an equivalent linear combination with only nonnegative weights. Negation completeness is usually a mild assumption for a reasonable $\mathcal{H}$ (Rätsch et al. 2002). Following this assumption, the classifier (5.7) can be interpreted as an ensemble classifier over $\mathcal{H}$ with an intercept term b. Now b can be viewed as the weight on a constant hypothesis h_c, which always predicts $h_c(\mathbf{x}) = 1$ for all $\mathbf{x} \in \mathcal{X}$. We shall further add a mild assumption that $\mathcal{H}$ contains both h_c and $(-h_c)$. Then, the classifier (5.7) or (5.3) is indeed equivalent to an ensemble classifier, and we get the following framework for infinite ensemble learning.

Algorithm 5.1 (SVM-based framework for infinite ensemble learning).

1. *Consider a training set* $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ *and the hypothesis set* $\mathcal{H}$*, which is assumed to be negation complete and to contain a constant hypothesis.*

[1] We use $(-h)$ to denote the function $(-h)(\cdot) = -(h(\cdot))$.

2. *Construct a kernel $\mathcal{K}_{\mathcal{H}}$ according to Definition 5.1 with a proper embedding function μ.*

3. *Choose proper parameters, such as the soft-margin parameter κ.*

4. *Solve* (5.2) *with $\mathcal{K}_{\mathcal{H}}$ and obtain Lagrange multipliers λ_n and the intercept term b.*

5. *Output the classifier*

$$\hat{g}(\mathbf{x}) = \operatorname{sign}\left(\sum_{n=1}^{N} y_n \lambda_n \mathcal{K}_{\mathcal{H}}(\mathbf{x}_n, \mathbf{x}) + b\right),$$

which is equivalent to some ensemble classifier over $\mathcal{H}$.

The framework shall generally inherit the profound performance of SVM. Most of the steps in the framework can be done by existing SVM implementations, and the hard part is mostly in obtaining the kernel $\mathcal{K}_{\mathcal{H}}$. We have derived several kernels for the framework (Lin 2005; Lin and Li 2008). Next, we introduce two important ones: the stump kernel and the perceptron kernel.

Stump kernel: The stump kernel embodies infinitely many decision stumps of the form

$$s_{q,d,\alpha}(\mathbf{x}) = q \cdot \operatorname{sign}\big(\mathbf{x}[d] - \alpha\big).$$

The decision stump $s_{q,d,\alpha}$ works on the d-th element of $\mathbf{x}$ and classifies $\mathbf{x}$ according to $q \in \{-1, +1\}$ and the threshold α (Holte 1993). It is widely used for ensemble learning because of its simplicity (Freund and Schapire 1996).

To construct the stump kernel, we consider the following set of decision stumps

$$\mathcal{S} = \Big\{ s_{q,d,\alpha_d} \colon q \in \{-1, +1\}, d \in \{1, \ldots, D\}, \alpha_d \in [L_d, R_d] \Big\}.$$

We also assume $\mathcal{X} \subseteq (L_1, R_1) \times (L_2, R_2) \times \cdots \times (L_D, R_D)$. Thus, the set $\mathcal{S}$ is negation complete and contains $s_{+1,1,L_1}$ as a constant hypothesis. The stump kernel $\mathcal{K}_{\mathcal{S}}$ defined

below can then be used in Algorithm 5.1 to obtain an infinite ensemble of decision stumps.

Definition 5.2 (Lin and Li 2008). *The stump kernel is $\mathcal{K}_{\mathcal{S}}$ using Definition 5.1 and $\mu(q, d, \alpha_d) = \mu_{\mathcal{S}} = \frac{1}{2}$. In particular,*

$$\mathcal{K}_{\mathcal{S}}(\mathbf{x}, \mathbf{x}') = \Delta_{\mathcal{S}} - \|\mathbf{x} - \mathbf{x}'\|_1 \ ,$$

where $\Delta_{\mathcal{S}} = \frac{1}{2}\sum_{d=1}^{D}(R_d - L_d)$ is a constant.

Definition 5.2 is a concrete instance that follows Definition 5.1. Because scaling $\mu_{\mathcal{S}}$ is equivalent to scaling the parameter κ in SVM (Lin and Li 2008), we use $\mu_{\mathcal{S}} = \frac{1}{2}$ to obtain a cosmetically cleaner kernel function.

Given the ranges (L_d, R_d), the stump kernel is very simple to compute. Furthermore, the ranges are not even necessary in general, because dropping the constant $\Delta_{\mathcal{S}}$ does not affect the classifier obtained from SVM (Lin and Li 2008). That is, in (5.2), the *simplified stump kernel* $\tilde{\mathcal{K}}_{\mathcal{S}}(\mathbf{x}, \mathbf{x}') = -\|\mathbf{x}-\mathbf{x}'\|_1$ can be used instead of $\mathcal{K}_{\mathcal{S}}$ without changing the resulting classifier $\hat{g}$. The simplified stump kernel is simple to compute, yet useful in the sense of dichotomizing the training set—that is, fitting the training set perfectly.

Theorem 5.3 (Lin and Li 2008). *Consider training input vectors $\{\mathbf{x}_n\}_{n=1}^{N}$. If there is a dimension d such that $\mathbf{x}_m[d] \neq \mathbf{x}_n[d]$ for all $m \neq n$, then there exists some $\kappa^* > 0$ such that for all $\kappa \geq \kappa^*$, SVM with $\mathcal{K}_{\mathcal{S}}$ (or $\tilde{\mathcal{K}}_{\mathcal{S}}$) can always dichotomize the training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$.*

We shall make a remark here. Although Theorem 5.3 indicates how the stump kernel can be used with SVM to dichotomize the training set perfectly, the classifier obtained may suffer from overfitting (Keerthi and Lin 2003). Thus, SVM is usually coupled with a reasonable parameter selection procedure to achieve good test performance (Hsu, Chang and Lin 2003; Keerthi and Lin 2003).

Perceptron kernel: The perceptron kernel embodies infinitely many perceptrons, which are linear threshold classifiers of the form

$$p_{\mathbf{u},\alpha}(\mathbf{x}) = \operatorname{sign}\Big(\langle \mathbf{u}, \mathbf{x}\rangle - \alpha\Big).$$

It is a basic theoretical model for a neuron and is important for building neural networks (Haykin 1999).

To construct the perceptron kernel, we consider the following set of perceptrons

$$\mathcal{P} = \left\{p_{\mathbf{u},\alpha} \colon \mathbf{u} \in \mathbb{R}^D, \|\mathbf{u}\|_2 = 1, \alpha \in [-R, R]\right\}.$$

We assume that $\mathcal{X}$ is within the interior of a ball of radius R centered at the origin in $\mathbb{R}^D$. Then, the set $\mathcal{P}$ is negation complete and contains a constant hypothesis ($\mathbf{u} = \mathbf{1}_1$ and $\alpha = -R$). Thus, the perceptron kernel $\mathcal{K}_{\mathcal{P}}$ defined below can be used in Algorithm 5.1 to obtain an infinite ensemble of perceptrons.

Definition 5.4 (Lin and Li 2008). *Let*

$$\Theta_D = \int_{\|\mathbf{u}\|_2=1} d\mathbf{u}, \quad \Xi_D = \int_{\|\mathbf{u}\|_2=1} \Big|\cos\big(\operatorname{angle}(\mathbf{u}, \mathbf{1}_1)\big)\Big| \, d\mathbf{u},$$

where the operator $\operatorname{angle}(\cdot,\cdot)$ *is the angle between two vectors, and the integrals are calculated with uniform measure on the surface* $\|\mathbf{u}\|_2 = 1$. *The perceptron kernel is* $\mathcal{K}_{\mathcal{P}}$ *with* $\mu(\mathbf{u}, \alpha) = \mu_{\mathcal{P}}$. *In particular,*

$$\mathcal{K}_{\mathcal{P}}(\mathbf{x}, \mathbf{x}') = \Delta_{\mathcal{P}} - \|\mathbf{x} - \mathbf{x}'\|_2,$$

where the constants $\mu_{\mathcal{P}} = (2\Xi_D)^{-\frac{1}{2}}$ *and* $\Delta_{\mathcal{P}} = \Theta_D \Xi_D^{-1} R$.

With the perceptron kernel, we can construct an infinite ensemble of perceptrons. Such an ensemble is equivalent to a neural network with one hidden layer, infinitely many hidden neurons, and the hard-threshold activation functions. Williams (1998) built an infinite neural network with either the sigmoid or the Gaussian activation

function through computing the corresponding covariance function for Gaussian process models. Analogously, our approach returns an infinite neural network with hard-threshold activation functions (ensemble of perceptrons) through computing the perceptron kernel for SVM. Williams (1998) stated that "*Paradoxically, it may be easier to carry out Bayesian prediction with infinite networks rather than finite ones.*" Similar claims can be made with ensemble learning.

The perceptron kernel shares many similar properties to the stump kernel. First, the constant $\Delta_{\mathcal{P}}$ can also be dropped. That is, we can use the *simplified perceptron kernel* $\tilde{\mathcal{K}}_{\mathcal{P}}(\mathbf{x}, \mathbf{x}') = -\|\mathbf{x} - \mathbf{x}'\|_2$ instead of $\mathcal{K}_{\mathcal{P}}$. Second, SVM with the perceptron kernel can also dichotomize the training set perfectly, as formalized below.

Theorem 5.5 (Lin and Li 2008). *For the training set* $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$*, if* $\mathbf{x}_m \neq \mathbf{x}_n$ *for all* $m \neq n$*, there exists some* $\kappa^* > 0$ *such that for all* $\kappa \geq \kappa^*$*, SVM with* $\mathcal{K}_{\mathcal{P}}$ *(or* $\tilde{\mathcal{K}}_{\mathcal{P}}$*) can always dichotomize the training set.*

5.1.3 Experiments

Next, we compare our SVM-based framework for infinite ensemble learning with AdaBoost using the decision stumps or perceptrons as the base hypothesis set. The simplified stump kernel (SVM-Stump) and the simplified perceptron kernel (SVM-Perc) are plugged into Algorithm 5.1 respectively. For AdaBoost, the deterministic decision stump algorithm (Holte 1993) and the random coordinate descent perceptron algorithm (Li and Lin 2007a) are taken as the base algorithm for the corresponding base hypothesis set. For perceptrons, we use the RCD-bias variant with 200 epochs of training. All base algorithms above have been shown to work reasonably well with AdaBoost in literature (Freund and Schapire 1996; Li and Lin 2007a).

LIBSVM (Chang and Lin 2001) is adopted as the SVM solver, with a suggested procedure that selects a suitable parameter with a 5-fold CV on the training set (Hsu, Chang and Lin 2003). The parameter κ is searched within $\{2^{-17}, 2^{-15}, \ldots, 2^3\}$. After the parameter selection procedure, a new decision function is obtained from the whole training set, and the performance is evaluated on an unseen test set.

For AdaBoost, we conduct a similar parameter selection procedure and search T within $\{10, 20, \ldots, 1500\}$.

The three artificial data sets from Breiman (1999) (twonorm, threenorm, and ringnorm) are generated with training set size 300 and test set size 3000. We create three more data sets (twonorm-n, threenorm-n, ringnorm-n), which contain mislabeling noise on 10% of the training examples, to test the performance of the algorithms on noisy environments. We also use eight real-world data sets from the UCI repository (Hettich, Blake and Merz 1998): australian, breast, german, heart, ionosphere, pima, sonar, and votes84. Their feature elements are scaled to $[-1,\ 1]$. We randomly pick 60% of the examples for training, and the rest for testing. For the data sets above, we compute the means and the standard errors of the results over 100 different random runs. In addition, four larger real-world data sets are used to test the validity of the framework for large-scale learning. They are a1a (Hettich, Blake and Merz 1998; Platt 1998), splice (Hettich, Blake and Merz 1998), svmguide1 (Hsu, Chang and Lin 2003), and w1a (Platt 1998).[2] Each of them comes with a benchmark test set, on which we report the results. The information of the data sets used is summarized in Table 5.1.

Tables 5.2 and 5.3 show the test performance of ensemble learning algorithms on different base hypothesis sets. We can see that SVM-Stump and SVM-Perc are usually better than AdaBoost with the corresponding base hypothesis set. In noisy data sets, SVM-based framework for infinite ensemble learning always significantly outperforms AdaBoost. These results demonstrate that it is beneficial to go from a finite ensemble to an infinite one with suitable regularization.

Note that AdaBoost and our SVM-based framework differ in the concept of sparsity. As illustrated in Subsection 5.1.1, AdaBoost prefers sparse ensemble classifiers, that is, ensembles that include a small number of hypotheses. Our framework works with an infinite number of hypotheses, but results in a sparse classifier in the support vector domain. Both concepts can be justified with various bounds on the expected

[2]These data sets are downloadable on tools page of LIBSVM (Chang and Lin 2001).

Table 5.1: Binary classification data sets

data set	# training examples	# test examples	# features (D)
twonorm	300	3000	20
twonorm-n	300	3000	20
threenorm	300	3000	20
threenorm-n	300	3000	20
ringnnorm	300	3000	20
ringnorm-n	300	3000	20
australian	414	276	14
breast	409	274	10
german	600	400	24
heart	162	108	13
ionosphere	210	141	34
pima	460	308	8
sonar	124	84	60
votes84	261	174	16
a1a	1605	30956	123
splice	1000	2175	60
svmguide1	3089	4000	4
w1a	2477	47272	300

test cost (Freund and Schapire 1997; Graepel, Herbrich and Shawe-Taylor 2005). Nevertheless, our experimental results indicate that sparse ensemble classifiers are sometimes not sophisticated enough in practice, especially when the base hypothesis set is simple. For instance, when using the decision stumps, a general data set may require many of them to describe a suitable decision boundary. Thus, AdaBoost-Stump could be limited by the finiteness and sparsity restrictions (Lin and Li 2008). On the other hand, our framework (SVM-Stump), which suffers from neither restrictions, can perform better by averaging over an infinite number of hypotheses.

In our earlier work (Lin and Li 2008), we observed another advantage of the perceptron kernel (SVM-Perc). In particular, the perceptron kernel and the popular Gaussian kernel share almost indistinguishable performance in the experiments, but the former enjoys the benefit of faster parameter selection. For instance, determining a good parameter for the Gaussian kernel involves solving 550 optimization problems, but SVM-Perc deals with only 55. With the indistinguishable performance, SVM-Perc

Table 5.2: Test classification cost (%) of SVM-Stump and AdaBoost-Stump

data set	SVM-Stump	AdaBoost-Stump
twonorm	**2.858±0.038**	5.022±0.062
twonorm-n	**3.076±0.055**	12.748±0.165
threenorm	**17.745±0.100**	22.096±0.117
threenorm-n	**19.047±0.144**	26.136±0.167
ringnorm	**3.966±0.067**	10.082±0.140
ringnorm-n	**5.558±0.110**	19.620±0.200
australian	14.446±0.205	**14.232±0.179**
breast	**3.113±0.080**	4.409±0.103
german	**24.695±0.183**	25.363±0.193
heart	**16.352±0.274**	19.222±0.349
ionosphere	**8.128±0.173**	11.340±0.252
pima	**24.149±0.226**	24.802±0.225
sonar	**16.595±0.420**	19.441±0.383
votes84	4.759±0.139	**4.270±0.152**
a1a	16.194	**15.984**
splice	6.207	**5.747**
svmguide1	**2.925**	3.350
w1a	**2.090**	2.177

(for the last 4 rows, the best results are marked in bold; for the other rows, those within one standard error of the lowest one are marked in bold)

should be a preferable choice in practice.

Both advantages of SVM-Perc above were inherited by the REDSVM (and SVOR-IMC) algorithm for ordinal ranking via the reduction framework (Algorithm 4.1). First, we list the results of ORBoost-All with perceptron (Table 3.1) and REDSVM with the perceptron kernel (Table 4.2) in Table 5.4. Both algorithms can return a threshold ensemble of perceptrons. ORBoost-All roots from AdaBoost, while REDSVM roots from SVM. In the table, we see that REDSVM with the perceptron kernel is usually better than ORBoost-All with perceptron, just as SVM-Perc is usually better than AdaBoost-Perc.

Second, when using the perceptron kernel, SVOR-IMC (and REDSVM) also enjoys the benefit of faster parameter selection. In addition, in Tables 4.2 and 4.4, we see that SVOR-IMC performs decently with both the perceptron and the Gaussian kernels (actually, better with the perceptron kernel). Such a result makes the perceptron

Table 5.3: Test classification cost (%) of SVM-Perc and AdaBoost-Perc

data set	SVM-Perc	AdaBoost-Perc
twonorm	**2.548±0.033**	3.114±0.041
twonorm-n	**2.755±0.052**	4.529±0.101
threenorm	**14.643±0.084**	17.322±0.113
threenorm-n	**16.299±0.103**	20.018±0.182
ringnorm	**2.464±0.038**	36.278±0.141
ringnorm-n	**3.505±0.086**	37.812±0.196
australian	**14.482±0.170**	15.656±0.159
breast	**3.230±0.080**	3.493±0.101
german	**24.593±0.196**	25.027±0.184
heart	**17.556±0.307**	18.222±0.324
ionosphere	**6.404±0.198**	11.425±0.234
pima	**23.545±0.212**	24.825±0.197
sonar	**15.619±0.401**	19.774±0.427
votes84	**4.425±0.138**	**4.374±0.164**
a1a	**15.690**	19.986
splice	**10.391**	13.655
svmguide1	**3.100**	3.275
w1a	**1.915**	2.348

(for the last 4 rows, the best results are marked in bold; for the other rows, those within one standard error of the lowest one are marked in bold)

Table 5.4: Test absolute cost of algorithms for threshold perceptron ensembles

data	perceptron	
set	RED-SVM	ORBoost-All
pyrimdines	**1.304±0.040**	1.360±0.046
machine	**0.842±0.022**	0.889±0.019
boston	**0.732±0.013**	0.791±0.013
abalone	**1.383±0.004**	1.432±0.003
bank	**1.404±0.002**	1.490±0.002
computer	**0.565±0.002**	0.626±0.002
california	**0.940±0.001**	0.977±0.002
census	**1.143±0.002**	1.265±0.002

(those within one standard error of the lowest one are marked in bold)

kernel a preferable choice for ordinal ranking. These advantages clearly demonstrate how we can improve both binary classification and ordinal ranking simultaneously with the reduction framework.

5.2 AdaBoost with Seeding

We showed in the previous section that AdaBoost can suffer from the restrictions of finiteness and sparsity. Nevertheless, it is repeatedly observed in literature that AdaBoost enjoys one practical advantage: its resistance to overfitting (Breiman 1998; Mease and Wyner 2008; Schapire et al. 1998). More specifically, during the iterations of AdaBoost, it is often observed that even after the training cost has reached 0 (and hence cannot decrease further), the test cost keeps decreasing with T (Schapire et al. 1998). In other words, AdaBoost often does not overfit more after adding more hypotheses to the ensemble. The phenomenon is not coherent with some of the theoretical foundations of AdaBoost (Freund and Schapire 1997) and hence continues to attract much research attention (Mease and Wyner 2008; Schapire et al. 1998).

As mentioned in Section 1.1, preventing overfitting is one of the most important objective when designing learning models and algorithms. Next, we study whether the advantage of AdaBoost can be used to prevent other algorithms from overfitting. In particular, we propose a new boosting approach that takes AdaBoost as a machinery to correct the overfitting effect of other learning algorithms. The key of the approach is to decompose AdaBoost to two stages. Then, our new boosting approach, called *AdaBoost with Seeding* (SeedBoost), replaces the first stage by another learning algorithm. As we will see from the experimental results, if the learning algorithm on hand overfits during the first stage, SeedBoost can inherit the advantage of AdaBoost and regularizes the overfitting effect in the second stage.

5.2.1 Algorithm

Before we introduce the SeedBoost algorithm, we take a closer look at AdaBoost (Algorithm 4.2). Following the gradient descent view of Mason et al. (2000), in the 1-st iteration, AdaBoost greedily chooses (h_1, v_1) to approximately minimize

$$\sum_{n=1}^{N} w_n \exp\big(-y_n v_1 h_1(\mathbf{x}_n)\big). \tag{5.8}$$

Then, in the t-th iteration, AdaBoost chooses (h_t, v_t) to approximately minimize

$$\sum_{n=1}^{N} w_n \exp\Big(-y_n\big(H_t(\mathbf{x}_n) + v_{t+1}h_{t+1}(\mathbf{x}_n)\big)\Big) = \sum_{n=1}^{N} w_n^{(t)} \exp\big(-y_n v_{t+1} h_{t+1}(\mathbf{x}_n)\big). \tag{5.9}$$

Comparing (5.8) and (5.9), we see that AdaBoost at the t-th iteration using the original training set $\{(\mathbf{x}_n, y_n, w_n)\}$ is equivalent to AdaBoost at the 1-st iteration using a modified training set $\Big\{(\mathbf{x}_n, y_n, w_n^{(t)})\Big\}$. Therefore, using (5.8) as a basic step, AdaBoost with $(t+T)$ iterations can be recursively defined as follows.

Algorithm 5.2 (A recursive view of AdaBoost with $(t+T)$ iterations).

1. *Run AdaBoost on* $\{(\mathbf{x}_n, y_n, w_n)\}_{n=1}^{N}$ *for* t *steps and get an ensemble classifier* $g^{(1)}(\mathbf{x}) = \text{sign}\Big(H_t^{(1)}(\mathbf{x})\Big)$.
2. *Run AdaBoost on* $\Big\{(\mathbf{x}_n, y_n, w_n^{(t)})\Big\}_{n=1}^{N}$ *for* T *steps and get an ensemble classifier* $g^{(2)}(\mathbf{x}) = \text{sign}\Big(H_T^{(2)}(\mathbf{x})\Big)$.
3. *Return* $H_{t+T}(\mathbf{x}) = \text{sign}\Big(H_t^{(1)}(\mathbf{x}) + H_T^{(2)}(\mathbf{x})\Big)$.

Our proposed SeedBoost algorithm simply generalizes the recursive steps above by replacing the first step with any learning algorithm, as listed below.

Algorithm 5.3 (SeedBoost: AdaBoost with seeding).

1. *Run some* $\mathcal{A}$ *on* $\{(\mathbf{x}_n, y_n, w_n)\}_{n=1}^{N}$ *and get a classifier* $g^{(1)}(\mathbf{x}) = \text{sign}\big(f^{(1)}(\mathbf{x})\big)$.

2. *Let* $w'_n = w_n \exp\big(-y_n f^{(1)}(\mathbf{x}_n)\big)$. *Run AdaBoost on* $\{(\mathbf{x}_n, y_n, w'_n)\}_{n=1}^N$ *for* T *steps and get an ensemble classifier* $g^{(2)}(\mathbf{x}) = \operatorname{sign}\Big(H_T^{(2)}(\mathbf{x})\Big)$.

3. *Return the combined decision function* $\hat{g}(\mathbf{x}) = \operatorname{sign}\Big(f^{(1)}(\mathbf{x}) + H_T^{(2)}(\mathbf{x})\Big)$.

We can see that the original AdaBoost with $(t + T)$ iterations is a special case of SeedBoost by using AdaBoost itself (with t iterations) as $\mathcal{A}$. Taking different learning algorithms as $\mathcal{A}$ in SeedBoost allows us to understand more about the properties of AdaBoost. For instance, as mentioned in the beginning of this section, AdaBoost has been observed to be resistant to overfitting even after the training cost has reached 0. Can SeedBoost inherit this property from AdaBoost to correct an overfitting algorithm $\mathcal{A}$? Also, can we obtain an improved performance by using some better algorithms (rather than the original AdaBoost) as $\mathcal{A}$? The answers to these questions help separate the effects of the two stages of AdaBoost. In the next section, we show some empirical study on SeedBoost to answer the questions above.

5.2.2 Experiments

We couple SeedBoost with two different algorithms as $\mathcal{A}$. The first one is a potentially overfitting algorithm. Recall that in Theorem 5.5, we proved that under a minor condition, when κ is large enough, SVM with the perceptron kernel can always dichotomize the training set. We set $\kappa = 2^{16}$ for this purpose and call the resulting algorithm the *separating SVM with the perceptron kernel* (SSVM-Perc). SSVM-Perc can reach training cost 0 in all our experiments, but may not lead to a good test cost because of overfitting. The second one is SVM-Perc (see Subsection 5.1.3), which is usually better than SSVM-Perc in terms of test performance because of it uses a parameter selection procedure to determine a suitable κ.

Similar to the procedures in Subsection 5.1.3, we use decision stumps and perceptrons within the AdaBoost component of SeedBoost. We perform experiment on the eight real-world data sets (see Table 5.1) from the UCI repository (Hettich, Blake and Merz 1998). For simplicity, we fix T to 100 in all the experiments, while the rest

of the setup is kept the same as the ones in Subsection 5.1.3.

Table 5.5 shows the results of SSVM-Perc both when used as a stand-alone learning algorithm and when coupled with SeedBoost. We see that SeedBoost can usually improve the test performance of SSVM-Perc. Such a result suggests that the second stage of AdaBoost exhibits some regularization properties that could correct overfitting.

Table 5.6 shows the results of SVM-Perc instead of SSVM-Perc. Unlike Table 5.5, SeedBoost usually cannot improve over stand-alone SVM-Perc significantly and often leads to worse test performance. That is, a decent binary classification algorithm like SVM-Perc cannot benefit by coupling with the second stage of AdaBoost.

If we compare Table 5.6 with AdaBoost-Perc in Table 5.3, we see that SeedBoost-Perc plus SVM-Perc is mostly comparable to AdaBoost-Perc. Thus, AdaBoost does not improve when we replace its first stage with a better learning algorithm. Such a result suggests that the second stage of AdaBoost plays a more important role and should be the focus of future research in explaining the success of AdaBoost.

Finally, in Table 5.7, we gather some columns from Tables 5.3 and 5.5 to show an interesting result: After SeedBoost corrects overfitting, on some of the data sets, SeedBoost-Stump plus SSVM-Perc can be significantly better than SVM-Perc. Note that because decision stumps are special cases of perceptrons, SeedBoost-Stump plus SSVM-Perc is also an algorithm that outputs an infinite ensemble of perceptrons. Since SSVM-Perc involves solving 1 optimization problem (5.2) while SVM-Perc needs to deal with 55, SeedBoost with SSVM-Perc is much faster than SVM-Perc in training. With the decent performance in Table 5.7 and faster training, SeedBoost-Stump with SSVM-Perc can be a useful alternative for infinite ensemble learning with perceptrons.

Table 5.5: Test classification cost (%) of SeedBoost with SSVM-Perc

data set	SSVM-Perc stand-alone	SeedBoost-Stump	SeedBoost-Perc
australian	16.696±0.141	**14.101±0.152**	15.438±0.149
breast	**3.299±0.087**	3.858±0.103	3.434±0.086
german	25.790±0.178	**24.093±0.182**	25.910±0.191
heart	19.722±0.323	**18.380±0.360**	**18.250±0.327**
ionosphere	**6.078±0.200**	9.830±0.247	11.298±0.254
pima	25.390±0.176	**24.396±0.195**	24.877±0.206
sonar	**15.012±0.368**	18.441±0.402	19.488±0.407
votes84	4.201±0.140	**3.994±0.135**	4.374±0.145

(those within one standard error of the lowest one are marked in bold)

Table 5.6: Test classification cost (%) of SeedBoost with SVM-Perc

data set	SVM-Perc stand-alone	SeedBoost-Stump	SeedBoost-Perc
australian	**14.482±0.170**	**14.525±0.176**	15.286±0.152
breast	**3.230±0.080**	4.051±0.104	3.453±0.089
german	**24.593±0.196**	**24.508±0.168**	26.122±0.192
heart	**17.556±0.307**	19.491±0.346	18.583±0.327
ionosphere	**6.404±0.198**	9.901±0.240	11.262±0.261
pima	**23.545±0.212**	24.302±0.192	25.195±0.201
sonar	**15.607±0.399**	18.012±0.386	19.786±0.417
votes84	4.425±0.138	**4.080±0.147**	4.351±0.142

(those within one standard error of the lowest one are marked in bold)

Table 5.7: Test classification cost (%) of SeedBoost with SSVM versus stand-alone SVM

data set	SSVM-Perc SeedBoost-Stump	SVM-Perc stand-alone
australian	**14.101±0.152**	14.482±0.170
breast	3.858±0.103	**3.230±0.080**
german	**24.093±0.182**	24.593±0.196
heart	18.380±0.360	**17.556±0.307**
ionosphere	9.830±0.247	**6.404±0.198**
pima	24.396±0.195	**23.545±0.212**
sonar	18.441±0.402	**15.607±0.399**
votes84	**3.994±0.135**	4.425±0.138

(those within one standard error of the lowest one are marked in bold)

Chapter 6

Conclusion

In Chapter 2, we proposed the cost-transformation technique from cost-sensitive classification to regular classification and proved its theoretical guarantees. Then, we designed two novel cost-sensitive classification algorithms, namely CSOVA and CSOVO, by applying the cost-transformation technique on their popular versions in regular classification. Experimental results showed that both algorithms worked well for cost-sensitive classification problems as well as for ordinal ranking problems.

In Chapter 3, we proposed the threshold ensemble model for ordinal ranking and defined margins for the model. Novel large-margin bounds of common cost functions were proved and were extended to threshold rankers for general threshold models. We studied two algorithms for obtaining threshold ensembles. The first algorithm, RankBoost-OR, combines RankBoost and a simple threshold algorithm. In addition, we designed a new boosting approach, ORBoost, which closely connects with the large-margin bounds. ORBoost is a direct extension of AdaBoost and inherits its theoretical and practical advantages. Experimental results demonstrated that both algorithms can perform decently on real-world data sets. In particular, ORBoost was comparable to SVM-based algorithms in terms of test cost, but enjoyed the advantage of faster training. These properties make ORBoost favorable over SVM-based algorithms on large data sets.

In Chapter 4, we presented the reduction framework from ordinal ranking to binary classification. The framework includes the reduction method and the reverse reduction technique. We showed the theoretical guarantees of the framework, includ-

ing the cost bound, the regret bound, and the equivalence between ordinal ranking and binary classification.

We used the reduction framework to extend SVM to ordinal ranking. We not only derived a general cost bound for SVM-based large-margin rankers, but also demonstrated that reducing to the standard SVM can readily yield superior performance in practice. We also used reduction to design a novel boosting approach, AdaBoost.OR, which can improve the performance of any cost-sensitive ordinal ranking algorithm. We showed the parallel between AdaBoost.OR and AdaBoost in algorithmic steps and in theoretical properties. Experimental results validated that AdaBoost.OR indeed improved both the training and test performance of existing ordinal ranking algorithms.

In Chapter 5, we first derived two novel kernels based on the SVM-based framework for infinite ensemble learning. The stump kernel embodies infinitely many decision stumps, and the perceptron kernel embodies infinitely many perceptrons. These kernels can be simply evaluated by the ℓ_1- or ℓ_2-norm distance between feature vectors. SVM equipped with the kernels generates infinite and nonsparse ensembles, which are usually more robust than finite and sparse ones. Experimental comparisons with AdaBoost showed that SVM with the kernels usually performed much better than AdaBoost with the same base hypothesis set. Therefore, existing applications that use AdaBoost with decision stumps or perceptrons may be improved by switching to SVM with the corresponding kernel. We also discussed how such an advantage propagates from binary classification to ordinal ranking.

Then, we proposed the SeedBoost algorithm, which takes AdaBoost as a machinery to regularize the overfitting effect of other learning algorithms. We conducted experimental studies on SeedBoost. The results demonstrated that SeedBoost not only improved some overfitting learning algorithms, but also achieved the best performance on some of the data sets.

Bibliography

Abe, N., B. Zadrozny, and J. Langford (2004). An iterative method for multi-class cost-sensitive learning. In W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel (Eds.), *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 3–11. ACM.

Abu-Mostafa, Y. S. (1989). The Vapnik-Chervonenkis dimension: Information versus complexity in learning. *Neural Computation 1*(3), 312–317.

Abu-Mostafa, Y. S., X. Song, A. Nicholson, and M. Magdon-Ismail (2004). The bin model. Technical Report CaltechCSTR:2004.002, California Institute of Technology.

Auer, P. and R. Meir (Eds.) (2005). *Learning Theory: 18th Annual Conference on Learning Theory*, Volume 3559 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory 44*(2), 525–536.

Bartlett, P. L. and J. Shawe-Taylor (1998). Generalization performance of support vector machines and other pattern classifiers. See (Schölkopf, Burges and Smola 1998), Chapter 4, pp. 43–54.

Becker, S., S. Thrun, and K. Obermayer (Eds.) (2003). *Advances in Neural Information Processing Systems: Proceedings of the 2002 Conference*, Volume 15. MIT Press.

Beygelzimer, A., V. Daniand, T. Hayes, J. Langford, and B. Zadrozny (2005). Error limiting reductions between classification tasks. In L. D. Raedt and S. Wrobel (Eds.), *Machine Learning: Proceedings of the 22rd International Conference*, pp. 49–56. ACM.

Beygelzimer, A., J. Langford, and P. Ravikumar (2007). Multiclass classification with filter trees. Downloaded from `http://hunch.net/~jl`.

Breiman, L. (1996). Bagging predictors. *Machine Learning 24*(2), 123–140.

Breiman, L. (1998). Arcing classifiers. *The Annals of Statistics 26*(3), 801–824.

Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Computation 11*(7), 1493–1517.

Cardoso, J. S. and J. F. P. da Costa (2007). Learning to classify ordinal data: The data replication method. *Journal of Machine Learning Research 8*, 1393–1429.

Chang, C.-C. and C.-J. Lin (2001). *LIBSVM: A Library for Support Vector Machines*. National Taiwan University. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chu, W. and Z. Ghahramani (2005). Gaussian processes for ordinal regression. *Journal of Machine Learning Research 6*, 1019–1041.

Chu, W. and S. S. Keerthi (2007). Support vector ordinal regression. *Neural Computation 19*(3), 792–815.

Crammer, K. and Y. Singer (2005). Online ranking by projecting. *Neural Computation 17*(1), 145–175.

Demiriz, A., K. P. Bennett, and J. Shawe-Taylor (2002). Linear programming boosting via column generation. *Machine Learning 46*(1-3), 225–254.

Domingos, P. (1999). MetaCost: A general method for making classifiers cost-sensitive. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 155–164. ACM.

Frank, E. and M. Hall (2001). A simple approach to ordinal classification. In L. D. Raedt and P. Flach (Eds.), *Machine Learning: Proceedings of the 12th European Conference on Machine Learning*, Volume 2167 of *Lecture Notes in Artificial Intelligence*, pp. 145–156. Springer-Verlag.

Freund, Y., R. Iyer, R. E. Shapire, and Y. Singer (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research 4*, 933–969.

Freund, Y. and R. E. Schapire (1996). Experiments with a new boosting algorithm. In L. Saitta (Ed.), *Machine Learning: Proceedings of the 13th International Conference*, pp. 148–156. Morgan Kaufmann.

Freund, Y. and R. E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55*(1), 119–139.

Freund, Y. and R. E. Schapire (1999). A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence 14*(5), 771–780. English version downloadable in `http://boosting.org`.

Gallant, S. I. (1990). Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks 1*(2), 179–191.

Graepel, T., R. Herbrich, and J. Shawe-Taylor (2005). PAC-Bayesian compression bounds on the prediction error of learning algorithms for classification. *Machine Learning 59*(1-2), 55–76.

Har-Peled, S., D. Roth, and D. Zimak (2003). Constraint classification: A new ap-

proach to multiclass classification and ranking. See (Becker, Thrun and Obermayer 2003), pp. 785–792.

Hastie, T., R. Tibshirani, and J. Friedman (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer-Verlag.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation* (Second ed.). Upper Saddle River, NJ: Prentice Hall.

Herbrich, R., T. Graepel, and K. Obermayer (2000). Large margin rank boundaries for ordinal regression. See (Smola et al. 2000), pp. 115–132.

Hettich, S., C. L. Blake, and C. J. Merz (1998). UCI repository of machine learning databases. Downloadable at `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association 58*(301), 13–30.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning 11*(1), 63–91.

Hsu, C.-W., C.-C. Chang, and C.-J. Lin (2003). A practical guide to support vector classification. Technical report, National Taiwan University.

Hsu, C.-W. and C.-J. Lin (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks 13*(2), 415–425.

Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*(5), 550–554.

International Neural Network Society (2007). *Proceedings of the 2007 International Joint Conference on Neural Networks (IJCNN 2007)*. International Neural Network Society: IEEE.

Joachims, T. (2005). Text categorization with support vector machines: Learning with many relevant features. In C. Nédellec and C. Rouveirol (Eds.), *Machine Learning: Proceedings of the 9th European Conference on Machine Learning*, Volume 1398 of *Lecture Notes in Computer Science*, pp. 137–142. Springer-Verlag.

Kearns, M. J. and U. V. Vazirani (1994). *An Introduction to Computational Learning Theory*. Cambridge, MA: MIT Press.

Keerthi, S. S. and C.-J. Lin (2003). Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Computation 15*(7), 1667–1689.

Langford, J. and A. Beygelzimer (2005). Sensitive error correcting output codes. See (Auer and Meir 2005), pp. 158–172.

Langford, J. and B. Zadrozny (2005). Estimating class membership probabilities using classifier learners. In R. G. Cowell and Z. Ghahramani (Eds.), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, Volume 13. Society for Artificial Intelligence and Statistics.

Li, L. and H.-T. Lin (2007a). Optimizing 0/1 loss for perceptrons by random coordinate descent. See (International Neural Network Society 2007), pp. 749–754.

Li, L. and H.-T. Lin (2007b). Ordinal regression by extended binary classification. In B. Schölkopf, J. C. Platt, and T. Hofmann (Eds.), *Advances in Neural Information Processing Systems: Proceedings of the 2006 Conference*, Volume 19, pp. 865–872. MIT Press.

Li, L., A. Pratap, H.-T. Lin, and Y. S. Abu-Mostafa (2005). Improving generalization by data categorization. In A. M. Jorge, L. Torgo, P. B. Brazdil, R. Camacho, and J. Gama (Eds.), *Knowledge Discovery in Databases: PKDD 2005*, Volume 3721 of *Lecture Notes in Computer Science*, pp. 157–168. Springer-Verlag.

Lin, H.-T. (2005). Infinite ensemble learning with support vector machines. Master's thesis, California Institute of Technology.

Lin, H.-T. and L. Li (2006). Large-margin thresholded ensembles for ordinal regression: Theory and practice. In J. L. Balcazár, P. M. Long, and F. Stephan (Eds.), *Algorithmic Learning Theory*, Volume 4264 of *Lecture Notes in Artificial Intelligence*, pp. 319–333. Springer-Verlag.

Lin, H.-T. and L. Li (2008). Support vector machinery for infinite ensemble learning. *Journal of Machine Learning Research 9*, 285–312.

Margineantu, D. D. (2001). *Methods for Cost-Sensitive Learning*. Ph. D. thesis, Oregon State University.

Mason, L., J. Baxter, P. L. Bartlett, and M. Frean (2000). Functional gradient techniques for combining hypotheses. See (Smola et al. 2000), pp. 221–246.

McCullagh, P. (1980). Regression models for ordinal data. *Journal of the Royal Statistical Society, Series B 42*(2), 109–142.

Mease, D. and A. Wyner (2008). Evidence contrary to the statistical view of boosting. *Journal of Machine Learning Research 9*, 131–156.

Meir, R. and G. Rätsch (2003). An introduction to boosting and leveraging. In S. Mendelson and A. J. Smola (Eds.), *Advanced Lectures on Machine Learning*, Volume 2600 of *Lecture Notes in Computer Science*, pp. 118–183. Springer-Verlag.

Platt, J. C. (1998). Fast training of support vector machines using sequential minimal optimization. See (Schölkopf, Burges and Smola 1998), Chapter 12, pp. 185–208.

Rajaram, S., A. Garg, X. S. Zhou, and T. S. Huang (2003). Classification approach towards ranking and sorting problems. In N. Lavrac, D. Gamberger, L. Todorovski, and H. Blockeel (Eds.), *Machine Learning: Proceedings of the 14th European Conference on Machine Learning*, Volume 2837 of *Lecture Notes in Computer Science*, pp. 301–312. Springer-Verlag.

Rätsch, G., A. Demiriz, and K. Bennett (2002). Sparse regression ensembles in infinite and finite hypothesis spaces. *Machine Learning 48*(1-3), 189–218.

Rätsch, G., S. Mika, B. Schölkopf, and K.-R. Müller (2002). Constructing boosting algorithms from SVMs: An application to one-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*(9), 1184–1199.

Rätsch, G., T. Onoda, and K. Müller (2001). Soft margins for AdaBoost. *Machine Learning 42*(3), 287–320.

Robertson, T., F. T. Wright, and R. L. Dykstra (1988). *Order Restricted Statistical Inference*. New York, NY: John Wiley & Sons.

Rosset, S., G. Swirszcz, N. Srebro, and J. Zhu (2007). ℓ_1 regularization in infinite dimensional feature spaces. In N. H. Bshouty and C. Gentile (Eds.), *Learning Theory: 20th Annual Conference on Learning Theory*, Volume 4539 of *Lecture Notes in Computer Science*, pp. 544–558. Springer-Verlag.

Rosset, S., J. Zhu, and T. Hastie (2004). Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research 5*, 941–973.

Rudin, C., C. Cortes, M. Mohri, and R. E. Schapire (2005). Margin-based ranking meets boosting in the middle. See (Auer and Meir 2005), pp. 63–78.

Schapire, R. E., Y. Freund, P. L. Bartlett, and W. S. Lee (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics 26*(5), 1651–1686.

Schölkopf, B., C. J. C. Burges, and A. J. Smola (Eds.) (1998). *Advances in Kernel Methods: Support Vector Learning*. MIT Press.

Schölkopf, B. and A. Smola (2002). *Learning with Kernels*. Cambridge, MA: MIT Press.

Shashua, A. and A. Levin (2003). Ranking with large margin principle: Two approaches. See (Becker, Thrun and Obermayer 2003), pp. 937–944.

Smola, A. J., P. L. Bartlett, B. Schökopf, and D. Schuurmans (Eds.) (2000). *Advances in Large Margin Classifiers*. MIT Press.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM 27*(11), 1134–1142.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. New York, NY: Springer-Verlag.

Vapnik, V. N. (1998). *Statistical Learning Theory*. New York, NY: Wiley.

Williams, C. K. I. (1998). Computation with infinite neural networks. *Neural Computation 10*(5), 1203–1216.

Xia, F., Q. Tao, J. Wang, and W. Zhang (2007). Recursive feature extraction for ordinal regression. See (International Neural Network Society 2007), pp. 78–83.

Xia, F., L. Zhou, Y. Yang, and W. Zhang (2007). Ordinal regression as multiclass classification. *International Journal of Intelligent Control and Systems 12*(3), 230–236.

Zadrozny, B., J. Langford, and N. Abe (2003). Cost sensitive learning by cost-proportionate example weighting. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pp. 435–442. IEEE Computer Society.