Journal of Machine Learning Research 1 (2017) 1-48 Submitted 12/17; Published ?

Support vector comparison machines

David Venuto DAVID.VENUTO@MAIL.MCGILL.CA
Department of Human Genetics
McGill University
Montreal, QC, H3A 0G4, Canada

Toby Dylan Hocking TOBY.HOCKING@MCGILL.CA
Department of Human Genetics
McGill University
Montreal, QC, H3A 0G4, Canada

Supaporn Spanurattana ?@K.U-TOKYO.AC.JP
Department of Computer Science
Tokyo Institute of Technology,
Tokyo 152-8552, Japan

Masashi Sugiyama SUGI@K.U-TOKYO.AC.JP
RIKEN Center for Advanced Intelligence Project
and The University of Tokyo,
Tokyo 103-0027, Japan

Editor: ?

Abstract

In ranking problems, the goal is to learn a ranking function $r(\mathbf{x}) \in \mathbb{R}$ from labeled pairs $\mathbf{x}, \mathbf{x}'$ of input points. In this paper, we consider the related comparison problem, where the label $y \in \{-1, 0, 1\}$ indicates which element of the pair is better ($y = -1$ or 1), or if there is no significant difference ($y = 0$). We cast the learning problem as a margin maximization, and show that it can be solved by converting it to a standard SVM. We use simulated nonlinear patterns and a real learning to rank sushi data set to show that our proposed SVMcompare algorithm outperforms SVMrank when there are equality $y = 0$ pairs. In addition, we show that SVMcompare outperforms the ELO rating system when predicting the outcome of chess matches.

1. Introduction

In the supervised learning to rank problem (Li, 2011), we are given labeled pairs of items $\mathbf{x}, \mathbf{x}'$, where the label $y \in \{-1, 1\}$ indicates which item in the pair should be ranked higher. The goal is to learn a ranking function $r(\mathbf{x}) \in \mathbb{R}$ which outputs a real-valued rank for each item. In this paper we consider a related problem in which the expanded label space $y \in \{-1, 0, 1\}$ includes the $y = 0$ label which indicates that there should be no rank difference. In this context the goal is to learn a comparison function $c(\mathbf{x}, \mathbf{x}') \in \{-1, 0, 1\}$.

Comparison data naturally arise from competitive two-player games in which the space of possible outcomes includes a draw (neither player wins). In games such as chess, draws are a common result between highly skilled players. To accurately predict the outcome of such games, it is thus important to learn a model that can predict a draw.

Comparison data also result when considering subjective human evaluations of pairs of items. For example, if each item is a movie, a person might say that *Les Misérables* is better than *Star Wars*, and *The Empire Strikes Back* is as good as *Star Wars*. Another example is rating food items such as wine, in which a person may prefer one wine to another, but not be able to perceive a difference between two other wines. In this context, it is important to use a model which can predict no difference between two items.

More formally, assume that we have a training sample of n labeled pairs. For each pair $i \in \{1, \ldots, n\}$ we have input features $\mathbf{x}_i, \mathbf{x}'_i \in \mathbb{R}^p$, where p is a positive integer, and a label $y_i \in \{-1, 0, 1\}$ that indicates which element is better:

$$y_i = \begin{cases} -1 & r(\mathbf{x}_i) > r(\mathbf{x}'_i), \ \mathbf{x}_i \text{ is better than } \mathbf{x}'_i, \\ 0 & r(\mathbf{x}_i) = r(\mathbf{x}'_i), \ \mathbf{x}_i \text{ is as good as } \mathbf{x}'_i, \\ 1 & r(\mathbf{x}_i) < r(\mathbf{x}'_i), \ \mathbf{x}'_i \text{ is better than } \mathbf{x}_i. \end{cases} \tag{1}$$

These data are geometrically represented in the top panel of Figure 1. Pairs with equality labels $y_i = 0$ are represented as line segments, and pairs with inequality labels $y_i = \{-1, 1\}$ are represented as arrows pointing to the item with the higher rank.

The goal of learning is to find a comparison function $c : \mathbb{R}^p \times \mathbb{R}^p \to \{-1, 0, 1\}$ which generalizes to a test set of data, as measured by the zero-one loss:

$$\underset{c}{\text{minimize}} \sum_{i \in \text{test}} I\left[c(\mathbf{x}_i, \mathbf{x}'_i) \neq y_i\right], \tag{2}$$

where I is the indicator function. If there are no equality $y_i = 0$ pairs, then this problem is equivalent to learning to rank with a pairwise zero-one loss function (Li, 2011). Learning to rank has been extensively studied, resulting in state-of-the-art algorithms such as SVMrank (Joachims, 2002). However, we are interested in learning to compare with equality $y_i = 0$ pairs, which to our knowledge has only been studied by Zhou et al. (2008). In this article we propose SVMcompare, a support vector algorithm for these data.

The notation and organization of this article is as follows. We use bold uppercase letters for matrices such as $\mathbf{X}, \mathbf{K}$, and bold lowercase letters for their row vectors $\mathbf{x}_i, \mathbf{k}_i$. In Section 2 we discuss links with related work on classification and ranking, then in Section 3 we propose a new algorithm: SVMcompare. We show results on 3 illustrative simulated data sets and two real learning to rank sushi data set and chess datasets in Section 4, then discuss future work in Section 5.

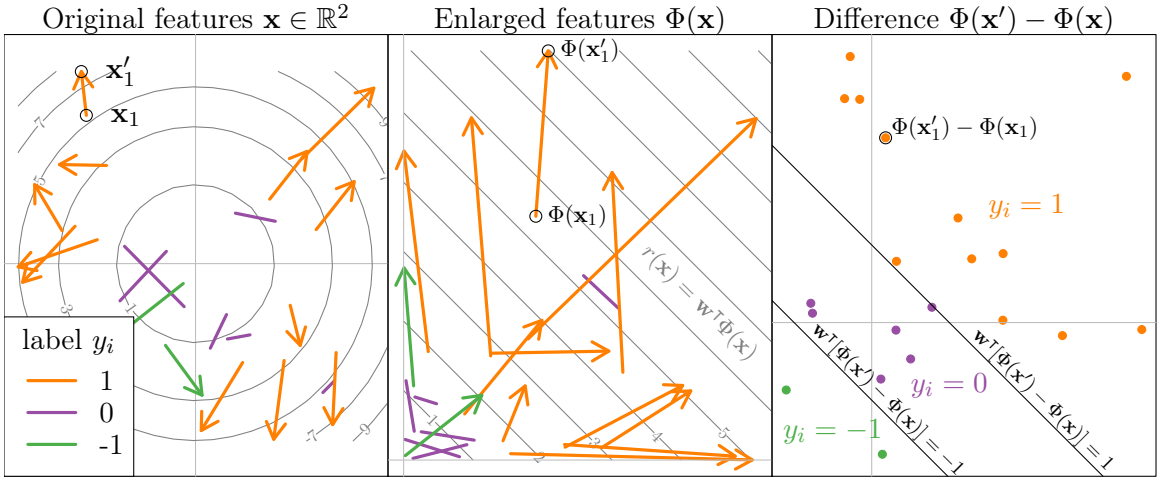

Figure 1: Geometric interpretation. **Left**: input feature pairs $\mathbf{x}_i, \mathbf{x}'_i \in \mathbb{R}^p$ are segments for $y_i = 0$ and arrows for $y_i \in \{-1, 1\}$. The level curves of the ranking function $r(\mathbf{x}) = ||\mathbf{x}||_2^2$ are grey, and differences $|r(\mathbf{x}') - r(\mathbf{x})| \leq 1$ are considered insignificant ($y_i = 0$). **Middle**: in the enlarged feature space, the ranking function is linear: $r(\mathbf{x}) = \mathbf{w}^\intercal \Phi(\mathbf{x})$. **Right**: two symmetric hyperplanes $\mathbf{w}^\intercal[\Phi(\mathbf{x}'_i) - \Phi(\mathbf{x}_i)] \in \{-1, 1\}$ are used to classify the difference vectors.

2. Related work

First we discuss connections with several existing methods, and then we discuss how ranking algorithms can be applied to the comparison problem.

2.1 Comparison and ranking problems

In ranking problems, each training example is a pair of inputs/features $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$, and a corresponding label/output $y_i n \{-1, 1\}$, which indicates which of the two inputs should be ranked higher (Table 1). In the comparison problem that we study in this paper, outputs $y \in \{-1, 0, 1\}$ include the $y = 0$ equality pairs or ties, which indicate that the two inputs should be ranked equally. The statistics literature contains many probabilistic models for paired comparison experiments, some of which directly model ties (Davidson, 1970). Such

Outputs \ Inputs	single items $\mathbf{x}$	pairs $\mathbf{x}, \mathbf{x}'$
$y \in \{-1, 1\}$	SVM	SVMrank
$y \in \{-1, 0, 1\}$	-	**This work: SVMcompare**

Table 1: Our proposed SVM for comparison is similar to previous SVM algorithms for ranking and binary classification.

models are concerned with accurately ranking a finite number of inputs $x \in \{1, \ldots, t\}$, so are not directly applicable to the real-valued inputs $\mathbf{x} \in \mathbb{R}^p$ we consider in this paper.

The supervised learning to rank problem has been extensively studied in the machine learning literature (Kamishima et al., 2010; Li, 2011), and is similar to the supervised comparison problem we consider in this paper. There are several Bayesian models which can be applied to learning to rank, such as TrueSkill (Herbrich et al., 2006) and Glicko (Glickman, 1999), which are generalizations of the Elo chess rating system. The SVMrank algorithm was proposed for learning to rank (Joachims, 2002), and the large-margin learning formulation we propose in this article is similar. The difference is that we also consider the case where both inputs are judged to be equally good ($y_i = 0$). A boosting algorithm for this "ranking with ties" problem was proposed by Zhou et al. (2008), who observed that modeling ties is more effective when there are more output values.

Ranking data sets are often described not in terms of labeled pairs of inputs $(\mathbf{x}_i, \mathbf{x}'_i, y_i)$ but instead single inputs $\mathbf{x}_i$ with ordinal labels $y_i \in \{1, \ldots, k\}$, where k is the number of integral output values. Support Vector Ordinal Regression (Chu and Keerthi, 2005) has a large-margin learning formulation specifically designed for these data. Another approach is to first convert the inputs to a database of labeled pairs, and then learn a ranking model such as the SVMcompare model we propose in this paper. Van Belle et al. (2011) observed that directly using a regression model gives better performance than ranking models for survival data. However, in this paper we limit our study to models for labeled pairs of inputs, and we focus on answering the question, "how can we adapt the Support Vector Machine to exploit the structure of the equality $y_i = 0$ pairs when they are present?"

2.2 SVMrank for comparing

In this subsection we explain how to apply the existing SVMrank algorithm to a comparison data set. The goal of SVMrank is to learn a ranking function $r : \mathbb{R}^p \to \mathbb{R}$. When $r(\mathbf{x}) = \mathbf{w}^\intercal \mathbf{x}$ (where $^\top$ denotes the transpose) is linear, the primal problem for some cost parameter $C \in \mathbb{R}^+$ (where $\mathbb{R}^+$ is a set of all non-negative real numbers) is the following quadratic program (QP):

$$\begin{aligned} \underset{\mathbf{w},\xi}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}^\intercal\mathbf{w} + C \sum_{i \in \mathcal{I}_1 \cup \mathcal{I}_{-1}} \xi_i \\ \text{subject to} \quad & \forall i \in \mathcal{I}_1 \cup \mathcal{I}_{-1},\ \xi_i \geq 0, \\ & \text{and } \xi_i \geq 1 - \mathbf{w}^\intercal(\mathbf{x}'_i - \mathbf{x}_i)y_i, \end{aligned} \tag{3}$$

where $\mathcal{I}_y = \{i \mid y_i = y\}$ are the sets of indices for the different labels. Note that (3) is the same as Optimization Problem 1 (Ranking SVM), in the paper of Joachims (2002). Note also that the equality $y_i = 0$ pairs are not used in the optimization problem.

After obtaining a weight vector $\mathbf{w} \in \mathbb{R}^p$ by solving SVMrank (3), we get a ranking function $r(\mathbf{x}) \in \mathbb{R}$, but we are not yet able to predict equality $y_i = 0$ pairs. To do so, we extend SVMrank by defining a threshold $\tau \in \mathbb{R}^+$ and a thresholding function $t_\tau : \mathbb{R} \to \{-1, 0, 1\}$

$$t_\tau(x) = \begin{cases} -1 & \text{if } x < -\tau, \\ 0 & \text{if } |x| \leq \tau, \\ 1 & \text{if } x > \tau. \end{cases} \tag{4}$$

A comparison function $c_\tau : \mathbb{R}^p \times \mathbb{R}^p \to \{-1, 0, 1\}$ is defined as the thresholded difference of predicted ranks

$$c_\tau(\mathbf{x}, \mathbf{x}') = t_\tau\big(r(\mathbf{x}') - r(\mathbf{x})\big). \tag{5}$$

We can then use grid search to estimate an optimal threshold $\hat{\tau}$, by minimizing the zero-one loss with respect to all the training pairs:

$$\hat{\tau} = \arg\min_\tau \sum_{i=1}^{n} I\left[c_\tau(\mathbf{x}_i, \mathbf{x}_i') \neq y_i\right]. \tag{6}$$

However, there are two potential problems with the learned comparison function $c_{\hat{\tau}}$. First, the equality pairs $i \in \mathcal{I}_0$ are not used to learn the weight vector $\mathbf{w}$ in (3). Second, the threshold $\hat{\tau}$ is learned in a separate optimization step, which may be suboptimal. In the next section, we propose a new algorithm that fixes these issues by directly using all the training pairs in a single learning problem.

3. Support vector comparison machines

In this section we propose new learning algorithms for comparison problems. In all cases, we will first learn a ranking function $r : \mathbb{R}^p \to \mathbb{R}$ and then a comparison function c_1 :

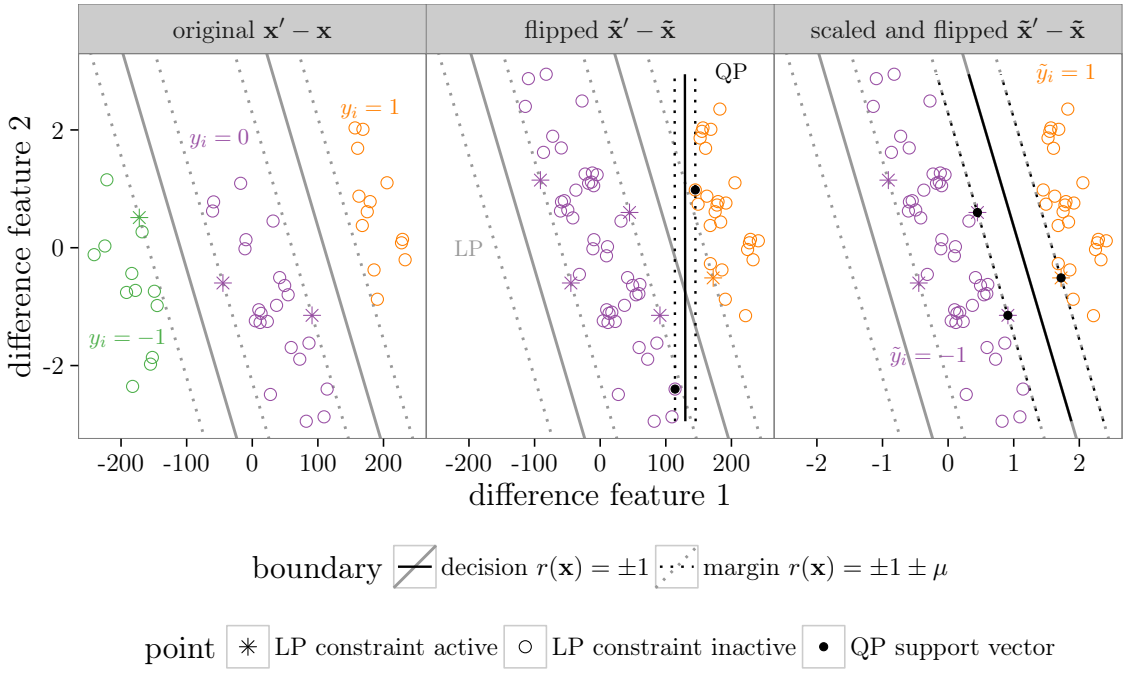

Figure 2: The separable LP and QP comparison problems. **Left**: the difference vectors $\mathbf{x}' - \mathbf{x}$ of the original data and the optimal solution to the LP (7). **Middle**: for the unscaled flipped data $\tilde{\mathbf{x}}' - \tilde{\mathbf{x}}$ (8), the LP is not the same as the QP (9). **Right**: in these scaled data, the QP is equivalent to the LP.

$\mathbb{R}^p \times \mathbb{R}^p \to \{-1, 0, 1\}$ (5). In other words, a small rank difference $|r(\mathbf{x}') - r(\mathbf{x})| \leq 1$ is considered insignificant, and there are two decision boundaries $r(\mathbf{x}') - r(\mathbf{x}) \in \{-1, 1\}$.

3.1 LP and QP for separable data

In our learning setup, the best comparison function is the one with maximum margin. We will define the margin in two different ways, which correspond to the linear program (LP) and quadratic program (QP) discussed below. To illustrate the differences between these max-margin comparison problems, in this subsection we assume that the training data are linearly separable. Later in Section 3.2, we propose an algorithm for learning a nonlinear function from non linearly-separable data.

In the following linear program, we learn a linear ranking function $r(\mathbf{x}) = \mathbf{w}^\intercal \mathbf{x}$ that maximizes the margin μ, defined in terms of ranking function values. The margin μ is the smallest rank difference between a decision boundary $r(\mathbf{x}) \in \{-1, 1\}$ and a difference vector $r(\mathbf{x}'_i - \mathbf{x}_i)$. The max margin LP is

$$\begin{array}{ll} \underset{\mu \in \mathbb{R}^+, \mathbf{w} \in \mathbb{R}^p}{\text{maximize}} & \mu \\ \text{subject to} & \mu \leq 1 - |\mathbf{w}^\intercal(\mathbf{x}'_i - \mathbf{x}_i)|, \ \forall\ i \in \mathcal{I}_0, \\ & \mu \leq -1 + \mathbf{w}^\intercal(\mathbf{x}'_i - \mathbf{x}_i) y_i, \ \forall\ i \in \mathcal{I}_1 \cup \mathcal{I}_{-1}. \end{array} \tag{7}$$

The optimal decision boundaries $r(\mathbf{x}) \in \{-1, 1\}$ and margin boundaries $r(\mathbf{x}) \in \{-1 \pm \mu, 1 \pm \mu\}$ are drawn in Figure 2. Note that finding a feasible point for this LP is a test of linear separability. If there are no feasible points then the data are not linearly separable.

Another way to formulate the comparison problem is by first performing a change of variables, and then solving a binary SVM QP. The idea is to maximize the margin between significant differences $y_i \in \{-1, 1\}$ and equality pairs $y_i = 0$. Let $\mathbf{X}_y, \mathbf{X}'_y$ be the $|\mathcal{I}_y| \times p$ matrices formed by all the pairs $i \in \mathcal{I}_y$. We define a new "flipped" data set with $m = |\mathcal{I}_1| + |\mathcal{I}_{-1}| + 2|\mathcal{I}_0|$ pairs suitable for training a binary SVM:

$$\tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}'_{-1} \\ \mathbf{X}_0 \\ \mathbf{X}'_0 \end{bmatrix}, \ \tilde{\mathbf{X}}' = \begin{bmatrix} \mathbf{X}'_1 \\ \mathbf{X}_{-1} \\ \mathbf{X}'_0 \\ \mathbf{X}_0 \end{bmatrix}, \ \tilde{\mathbf{y}} = \begin{bmatrix} \mathbf{1}_{|\mathcal{I}_1|} \\ \mathbf{1}_{|\mathcal{I}_{-1}|} \\ -\mathbf{1}_{|\mathcal{I}_0|} \\ -\mathbf{1}_{|\mathcal{I}_0|} \end{bmatrix}, \tag{8}$$

where $\mathbf{1}_n$ is an n-vector of ones, $\tilde{\mathbf{X}}, \tilde{\mathbf{X}}' \in \mathbb{R}^{m \times p}$ and $\tilde{\mathbf{y}} \in \{-1, 1\}^m$. Note that $\tilde{y}_i = -1$ implies no significant difference between $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}'_i$, and $\tilde{y}_i = 1$ implies that $\tilde{\mathbf{x}}'_i$ is better than $\tilde{\mathbf{x}}_i$. We then learn an affine function $f(\mathbf{x}) = \beta + \mathbf{u}^\intercal \mathbf{x}$ using a binary SVM QP:

$$\begin{array}{ll} \underset{\mathbf{u} \in \mathbb{R}^p, \beta \in \mathbb{R}}{\text{minimize}} & \mathbf{u}^\intercal \mathbf{u} \\ \text{subject to} & \tilde{y}_i(\beta + \mathbf{u}^\intercal(\tilde{\mathbf{x}}'_i - \tilde{\mathbf{x}}_i)) \geq 1, \ \forall i \in \{1, \ldots, m\}. \end{array} \tag{9}$$

This SVM QP learns a separator $f(\mathbf{x}) = 0$ between significant difference pairs $\tilde{y}_i = 1$ and insignificant difference pairs $\tilde{y}_i = -1$ (middle and right panels of Figure 2). However, we

want a comparison function that predicts $c(\mathbf{x}, \mathbf{x}') \in \{-1, 0, 1\}$. So we use the next lemma to construct a ranking function $r(\mathbf{x}) = \hat{\mathbf{w}}^\intercal \mathbf{x}$ that is feasible for the original max margin comparison LP (7), and can be used with the comparison function c_1 (5).

Lemma 1 *Let $\mathbf{u} \in \mathbb{R}^p, \beta \in \mathbb{R}$ be a solution of (9). Then $\hat{\mu} = -1/\beta$ and $\hat{\mathbf{w}} = -\mathbf{u}/\beta$ are feasible for (7).*

Proof Begin by assuming that we want to find a ranking function $r(\mathbf{x}) = \hat{\mathbf{w}}^\intercal \mathbf{x} = \gamma \mathbf{u}^\intercal \mathbf{x}$, where $\gamma \in \mathbb{R}$ is a scaling constant. Then consider that for all $\mathbf{x}$ on the decision boundary, we have

$$r(\mathbf{x}) = \hat{\mathbf{w}}^\intercal \mathbf{x} = 1 \text{ and } f(\mathbf{x}) = \mathbf{u}^\intercal \mathbf{x} + \beta = 0. \tag{10}$$

Taken together, it is clear that $\gamma = -1/\beta$ and thus $\hat{\mathbf{w}} = -\mathbf{u}/\beta$. Consider for all $\mathbf{x}$ on the margin we have

$$r(\mathbf{x}) = \hat{\mathbf{w}}^\intercal \mathbf{x} = 1 + \hat{\mu} \text{ and } f(\mathbf{x}) = \mathbf{u}^\intercal \mathbf{x} + \beta = 1. \tag{11}$$

Taken together, these imply $\hat{\mu} = -1/\beta$. Now, by definition of the flipped data (8), we can re-write the max margin QP (9) as

$$\begin{aligned} \underset{\mathbf{u} \in \mathbb{R}^p, \beta \in \mathbb{R}}{\text{minimize}} \quad & \mathbf{u}^\intercal \mathbf{u} \\ \text{subject to} \quad & \beta + |\mathbf{u}^\intercal (\mathbf{x}'_i - \mathbf{x}_i)| \leq -1, \ \forall \ i \in \mathcal{I}_0, \\ & \beta + \mathbf{u}^\intercal (\mathbf{x}'_i - \mathbf{x}_i) y_i \geq 1, \ \forall \ i \in \mathcal{I}_1 \cup \mathcal{I}_{-1}. \end{aligned} \tag{12}$$

By re-writing the constraints of (12) in terms of $\hat{\mu}$ and $\hat{\mathbf{w}}$, we recover the same constraints as the max margin comparison LP (7). Thus $\hat{\mu}, \hat{\mathbf{w}}$ are feasible for (7). ■

One may also wonder: are $\hat{\mu}, \hat{\mathbf{w}}$ optimal for the max margin comparison LP? In general, the answer is no, and we give one counterexample in the middle panel of Figure 2. This is because the LP defines the margin in terms of ranking function values $r(\mathbf{x}) = \mathbf{w}^\intercal \mathbf{x}$, but the QP defines the margin in terms of the size of the normal vector $||\mathbf{u}||$, which depends on the scale of the inputs $\mathbf{x}, \mathbf{x}'$. However, when the input variables are scaled in a pre-processing step, we have observed that the solutions to the LP and QP are equivalent (right panel of Figure 2).

Lemma 1 establishes the fact that one can learn a ranking function r and a corresponding comparison function c_1 (5) by solving either the LP (7) or the QP (12). To make corresponding learning problems for non linearly-separable data, one can add slack variables to either the QP or the LP. In the next subsection, we pursue only the QP, since it leads to a dual problem with a sparse solution that can be solved by any standard SVM solver such as libsvm (Chang and Lin, 2011).

3.2 Kernelized QP for non-separable data

In this subsection, we assume the data are not linearly separable, and want to learn a nonlinear ranking function. We define a positive definite kernel $\kappa : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$, which implicitly defines an enlarged set of features $\Phi(\mathbf{x})$ (middle panel of Figure 1). As in (9), we learn a function $f(\mathbf{x}) = \beta + \mathbf{u}^\intercal \Phi(\mathbf{x})$ which is affine in the feature space. Let $\alpha, \alpha' \in$

$\mathbb{R}^m$ be coefficients such that $\mathbf{u} = \sum_{i=1}^m \alpha_i \Phi(\tilde{\mathbf{x}}_i) + \alpha'_i \Phi(\tilde{\mathbf{x}}'_i)$, and so we have $f(\mathbf{x}) = \beta + \sum_{i=1}^m (\alpha_i \kappa(\tilde{\mathbf{x}}_i, \mathbf{x}) + \alpha'_i \kappa(\tilde{\mathbf{x}}'_i, \mathbf{x}))$. We then use Lemma 1 to define the ranking function

$$r(\mathbf{x}) = \frac{\mathbf{u}^\intercal \Phi(\mathbf{x})}{-\beta} = \sum_{i=1}^m \frac{\alpha_i \kappa(\tilde{\mathbf{x}}_i, \mathbf{x}) + \alpha'_i \kappa(\tilde{\mathbf{x}}'_i, \mathbf{x})}{-\beta}. \tag{13}$$

Let $\mathbf{K} = [\mathbf{k}_1 \cdots \mathbf{k}_m \ \mathbf{k}'_1 \cdots \mathbf{k}'_m] \in \mathbb{R}^{2m \times 2m}$ be the kernel matrix, where for all pairs $i \in \{1, \ldots, m\}$, the kernel vectors $\mathbf{k}_i, \mathbf{k}'_i \in \mathbb{R}^{2m}$ are defined as

$$\mathbf{k}_i = \begin{bmatrix} \kappa(\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_i) \\ \vdots \\ \kappa(\tilde{\mathbf{x}}_m, \tilde{\mathbf{x}}_i) \\ \kappa(\tilde{\mathbf{x}}'_1, \tilde{\mathbf{x}}_i) \\ \vdots \\ \kappa(\tilde{\mathbf{x}}'_m, \tilde{\mathbf{x}}_i) \end{bmatrix}, \ \mathbf{k}'_i = \begin{bmatrix} \kappa(\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}'_i) \\ \vdots \\ \kappa(\tilde{\mathbf{x}}_m, \tilde{\mathbf{x}}'_i) \\ \kappa(\tilde{\mathbf{x}}'_1, \tilde{\mathbf{x}}'_i) \\ \vdots \\ \kappa(\tilde{\mathbf{x}}'_m, \tilde{\mathbf{x}}'_i) \end{bmatrix}. \tag{14}$$

Letting $\mathbf{a} = [\alpha^\intercal \ \alpha'^\intercal]^\intercal \in \mathbb{R}^{2m}$, the norm of the affine function f in the feature space is $\mathbf{u}^\intercal \mathbf{u} = \mathbf{a}^\intercal \mathbf{K} \mathbf{a}$, and we can write the primal soft-margin comparison QP for some $C \in \mathbb{R}^+$ as

$$\begin{aligned} \underset{\mathbf{a} \in \mathbb{R}^{2m}, \xi \in \mathbb{R}^m, \beta \in \mathbb{R}}{\text{minimize}} \quad & \frac{1}{2} \mathbf{a}^\intercal \mathbf{K} \mathbf{a} + C \sum_{i=1}^m \xi_i \\ \text{subject to} \quad & \text{for all } i \in \{1, \ldots, m\}, \ \xi_i \geq 0, \\ & \text{and } \xi_i \geq 1 - \tilde{y}_i (\beta + \mathbf{a}^\intercal (\mathbf{k}'_i - \mathbf{k}_i)). \end{aligned} \tag{15}$$

Let $\mathbf{z}, \mathbf{v} \in \mathbb{R}^m$ be the dual variables, and $\mathbf{Y} = \text{Diag}(\tilde{\mathbf{y}})$ be the diagonal matrix of m labels. Then the Lagrangian can be written as

$$\mathcal{L} = \frac{1}{2} \mathbf{a}^\intercal \mathbf{K} \mathbf{a} + C \xi^\intercal \mathbf{1}_m - \mathbf{z}^\intercal \xi + \mathbf{v}^\intercal (\mathbf{1}_m - \tilde{\mathbf{y}} \beta - \mathbf{Y} \mathbf{M}^\intercal \mathbf{K} \mathbf{a} - \xi), \tag{16}$$

Algorithm 1 SVMcompare

Input: cost $C \in \mathbb{R}^+$, kernel $\kappa : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$, features $\mathbf{X}, \mathbf{X}' \in \mathbb{R}^{n \times p}$, labels $\mathbf{y} \in \{-1, 0, 1\}^n$.
$\tilde{\mathbf{X}} \leftarrow [\ \mathbf{X}_1^\intercal \quad \mathbf{X}'^\intercal_{-1} \quad \mathbf{X}_0^\intercal \quad \mathbf{X}'^\intercal_0 \ ]^\intercal$.
$\tilde{\mathbf{X}}' \leftarrow [\ \mathbf{X}'^\intercal_1 \quad \mathbf{X}_{-1}^\intercal \quad \mathbf{X}'^\intercal_0 \quad \mathbf{X}_0^\intercal \ ]^\intercal$.
$\tilde{\mathbf{y}} \leftarrow [\ \mathbf{1}^\intercal_{|\mathcal{I}_1|} \quad \mathbf{1}^\intercal_{|\mathcal{I}_{-1}|} \quad -\mathbf{1}^\intercal_{|\mathcal{I}_0|} \quad -\mathbf{1}^\intercal_{|\mathcal{I}_0|} \ ]^\intercal$.
$\mathbf{K} \leftarrow \textsc{KernelMatrix}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}', \kappa)$.
$\mathbf{M} \leftarrow [-\mathbf{I}_m \ \mathbf{I}_m]^\intercal$.
$\tilde{\mathbf{K}} \leftarrow \mathbf{M}^\intercal \mathbf{K} \mathbf{M}$.
$\mathbf{v}, \beta \leftarrow \text{SVM}\textsc{dual}(\tilde{\mathbf{K}}, \tilde{\mathbf{y}}, C)$.
$\textsc{sv} \leftarrow \{i : v_i > 0\}$.
Output: Support vectors $\tilde{\mathbf{X}}_{\text{sv}}, \tilde{\mathbf{X}}'_{\text{sv}}$, labels $\tilde{\mathbf{y}}_{\text{sv}}$, bias β, dual variables $\mathbf{v}$.

where $\mathbf{M} = [-\mathbf{I}_m \, \mathbf{I}_m]^\intercal \in \{-1, 0, 1\}^{2m \times m}$ and $\mathbf{I}_m$ is the identity matrix. Solving $\nabla_{\mathbf{a}} \mathcal{L} = 0$ results in the following stationary condition:

$$\mathbf{a} = \mathbf{MYv}. \tag{17}$$

The rest of the derivation of the dual comparison problem is the same as for the standard binary SVM. The resulting dual QP is

$$\begin{aligned} \underset{\mathbf{v} \in \mathbb{R}^m}{\text{minimize}} \quad & \frac{1}{2} \mathbf{v}^\intercal \mathbf{YM}^\intercal \mathbf{KMYv} - \mathbf{v}^\intercal \mathbf{1}_m \\ \text{subject to} \quad & \sum_{i=1}^{m} v_i \tilde{y}_i = 0, \\ & \text{for all } i \in \{1, \ldots, m\}, \, 0 \leq v_i \leq C, \end{aligned} \tag{18}$$

which is equivalent to the dual problem of a standard binary SVM with kernel $\tilde{\mathbf{K}} = \mathbf{M}^\intercal \mathbf{KM} \in \mathbb{R}^{m \times m}$ and labels $\tilde{\mathbf{y}} \in \{-1, 1\}^m$.

So we can solve the dual comparison problem (18) using any efficient SVM solver, such as libsvm (Chang and Lin, 2011). We used the R interface in the `kernlab` package (Karatzoglou et al., 2004), and our code is available in the `rankSVMcompare` package on Github.

After obtaining optimal dual variables $\mathbf{v} \in \mathbb{R}^m$ as the solution of (18), the SVM solver also gives us the optimal bias β by analyzing the complementary slackness conditions. The learned ranking function can be quickly evaluated since the optimal $\mathbf{v}$ is sparse. Let $\text{SV} = \{i : v_i > 0\}$ be the indices of the support vectors. Since we need only $2|\text{SV}|$ kernel evaluations, the ranking function (13) becomes

$$r(\mathbf{x}) = \sum_{i \in \text{SV}} \tilde{y}_i v_i \left[\kappa(\tilde{\mathbf{x}}_i, \mathbf{x}) - \kappa(\tilde{\mathbf{x}}'_i, \mathbf{x}) \right] / \beta. \tag{19}$$

Note that for all $i \in \{1, \ldots, m\}$, the optimal primal variables $\alpha_i = -\tilde{y}_i v_i$ and $\alpha'_i = \tilde{y}_i v_i$ are recovered using the stationary condition (17). The learned comparison function c_1 remains the same (5).

The training procedure is summarized as Algorithm 1, SVMcompare. There are two sub-routines: KernelMatrix computes the $2m \times 2m$ kernel matrix, and SVMdual solves the SVM dual QP (18). There are two hyper-parameters to tune: the cost C and the kernel κ. As with standard SVM for binary classification, these parameters can be tuned by minimizing the prediction error on a held-out validation set.

4. Results

The goal of learning to compare is to accurately predict a test set of labeled pairs (2), which includes equality $y_i = 0$ pairs. We test the SVMcompare algorithm alongside two baseline models that use SVMrank (Joachims, 2002). We chose SVMrank as a baseline because of its similar large-margin learning formulation, to demonstrate the importance of directly modeling the equality $y_i = 0$ pairs. SVMrank does not directly model the equality $y_i = 0$ pairs, so we expect that the proposed SVMcompare algorithm makes better predictions when these data are present. The differences between the algorithms are summarized in Table 2:

rank is described in Section 2.2: first we use $|\mathcal{I}_1|+|\mathcal{I}_{-1}|$ inequality pairs to learn SVMrank, then we use all n pairs to learn a threshold $\hat{\tau}$ for when to predict $c(\mathbf{x},\mathbf{x}')=0$.

rank2 is another variant of SVMrank that treats each input pair as 2 inequality pairs. Since SVMrank can only use inequality pairs, we transform each equality pair $(\mathbf{x}_i,\mathbf{x}'_i,0)$ into two opposite-facing inequality pairs $(\mathbf{x}'_i,\mathbf{x}_i,1)$ and $(\mathbf{x}_i,\mathbf{x}'_i,1)$. To ensure equal weight for all input pairs in the cost function, we also duplicate each inequality pair, resulting in $2n$ pairs used to train SVMrank.

compare is the SVMcompare model proposed in this paper, which uses $m=n+|\mathcal{I}_0|$ input pairs.

For each experiment, there are train, validation, and test sets each drawn from the same set of examples. We fit a 10×10 grid of models to the training set (cost parameter $C=10^{-3},\ldots,10^3$, Gaussian kernel width $2^{-7},\ldots,2^4$), and select the model using the validation set. We use two evaluation metrics to judge the performance of the models: zero-one loss and area under the ROC curve (AUC).

Note that the ROC curves are calculated by first evaluating the learned ranking function $r(\mathbf{x})$ at each test point $\mathbf{x}$, and then varying the threshold τ of the comparison function c_τ (5). For $\tau=0$ we have 100% false positive rate and for $\tau=\infty$ we have 100% false negative rate (Table 3).

4.1 Simulation: squared norms in 2D

We used a simulation to visualize the learned nonlinear ranking functions in a $p=2$ dimensional feature space. We generated pairs $\mathbf{x}_i,\mathbf{x}'_i\in[-3,3]^2$ and noisy labels $y_i=t_1[r(\mathbf{x}'_i)-r(\mathbf{x}_i)+\epsilon_i]$, where t_1 is the threshold function (4), r is the latent ranking function, $\epsilon_i\sim N(0,\sigma)$ is noise, and $\sigma=1/4$ is the standard deviation. We picked train, validation, and test sets, each with the same number of pairs n and the same proportion ρ of equality pairs. We selected the model with minimum zero-one loss on the validation set.

In Figure 3 we defined the true ranking function $r(\mathbf{x})=||\mathbf{x}||_1^2$, then picked $n=100$ pairs, with $\rho=1/2$ equality and inequality pairs. We show the training set and the level curves

Input:	equality pairs		inequality pairs	
	$\lvert\mathcal{I}_0\rvert$	—	$\lvert\mathcal{I}_1\rvert+\lvert\mathcal{I}_{-1}\rvert$	$\rightarrow$
rank	0		$\lvert\mathcal{I}_1\rvert+\lvert\mathcal{I}_{-1}\rvert$	$\rightarrow$
rank2	$2\lvert\mathcal{I}_0\rvert$	$\leftarrow\rightarrow$	$2(\lvert\mathcal{I}_1\rvert+\lvert\mathcal{I}_{-1}\rvert)$	$\rightarrow\rightarrow$
compare	$2\lvert\mathcal{I}_0\rvert$	— —	$\lvert\mathcal{I}_1\rvert+\lvert\mathcal{I}_{-1}\rvert$	$\rightarrow$

Table 2: Summary of how the different algorithms use the input pairs to learn the ranking function r. Equality $y_i=0$ pairs are shown as — segments and inequality $y_i\in\{-1,1\}$ pairs are shown as $\rightarrow$ arrows. For example, the rank2 algorithm converts each input equality pair to two opposite-facing inequality pairs.

of the ranking functions learned by the SVMrank and SVMcompare models. It is clear that the true ranking function r is not accurately recovered by the rank model, since it does not use the equality $y_i = 0$ pairs. In contrast, the compare and rank2 methods which exploit the equality $y_i = 0$ pairs are able to recover a ranking function that is closer to the true r.

We also used the simulations to demonstrate that our model can achieve lower test error than the baseline SVMrank model, by learning from the equality $y_i = 0$ pairs. In Figure 4 we fixed the proportion of equality pairs $\rho = 1/2$, varied the number of training pairs $n \in \{50, \ldots, 800\}$, and tested three simulated ranking functions $r(\mathbf{x}) = ||\mathbf{x}||_j^2$ for $j \in \{1, 2, \infty\}$. In general, the test error of all models decreases as training set size n increases. The model with the highest test error is the rank model, which does not use the equality $y_i = 0$ pairs. The next best model is the rank2 model, which converts the equality $y_i = 0$ pairs to inequality pairs and then trains SVMrank. The best model is the proposed SVMcompare model, which achieves test error as good as the true ranking function in the case of $r(\mathbf{x}) = ||\mathbf{x}||_2^2$.

$\hat{y}$ \ y	**-1**	**0**	**1**
-1	0	FP	FN
0	FN	0	FN
1	FN	FP	0

Table 3: We use area under the ROC curve to evaluate predictions $\hat{y}$ given the true label y. False positives (FP) occur when predicting a significant difference $\hat{y} \in \{-1, 1\}$ when there is none $y = 0$. False Negatives (FN) occur when a labeled difference $y \in \{-1, 1\}$ is incorrectly predicted.

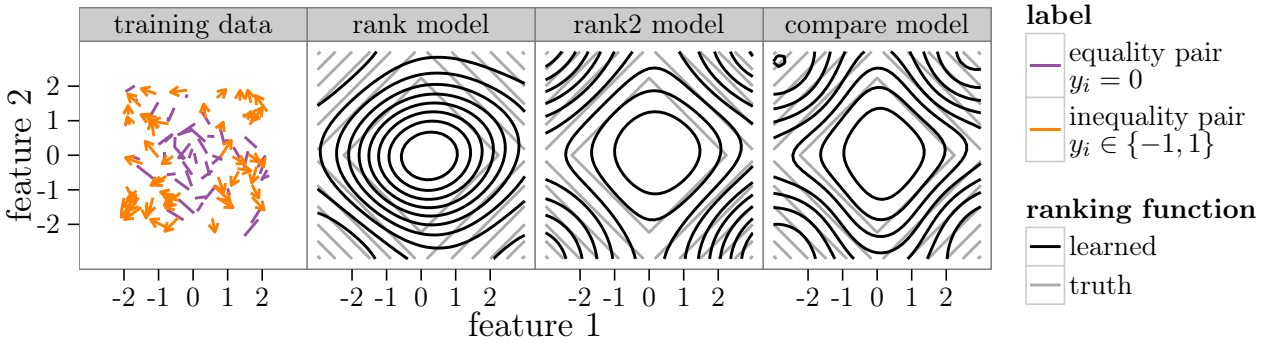

Figure 3: Application to a simulated pattern $r(\mathbf{x}) = ||\mathbf{x}||_1^2$ where $\mathbf{x} \in \mathbb{R}^2$. **Left**: the training data are $n = 100$ pairs, half equality (segments indicate two points of equal rank), and half inequality (arrows point to the higher rank). **Others**: level curves of the learned ranking functions. The rank model does not directly model the equality pairs, so the rank2 and compare models recover the true pattern better.

In Figure 5 we fixed the number of training pairs $n = 400$ and varied the proportion ρ of equality pairs for the three simulated squared norm ranking functions r. We select the model with maximum area under the validation set ROC curve, then use test set AUC to evaluate the learned models. All methods perform close to the optimal true ranking function when $r(\mathbf{x}) = ||\mathbf{x}||_2^2$. For the other patterns, it is clear that all the methods perform similarly when there are mostly inequality pairs ($\rho = 0.1$), since SVMrank was designed for this type of training data. In contrast, when there are mostly equality pairs ($\rho = 0.9$), the compare and rank2 methods clearly outperform the rank method, which ignores the equality pairs. It is also clear that the rank2 and compare methods perform similarly in terms of test AUC.

Overall from the simulations, it is clear that when the data contain equality pairs, it is advantageous to use a model such as the proposed SVMcompare method which learns from them directly as a part of the optimization problem.

4.2 Learning to rank sushi data

We downloaded the sushi data set of Kamishima et al. (2010) from kamishima*. We used the `sushi3b.5000.10.score` data, which consist of 100 different sushis rated by 5000 different people. Each person rated 10 sushis on a 5 point scale, which we convert to 5 preference pairs, for a total of 17,832 equality $y_i = 0$ and 7,168 inequality $y_i \in \{-1, 1\}$ pairs. For each pair i we have features $\mathbf{x}_i, \mathbf{x}'_i \in \mathbb{R}^{14}$ consisting of 7 features of the sushi and 7 features of the person. Sushi features are style, major, minor, oily, eating frequency, price, and selling frequency. Person features are gender, age, time, birthplace and current home (we converted Japanese prefecture codes to latitude/longitude coordinates). As in the simulations of Section 4.1, we picked train, validation, and test sets, each with the same number of pairs n and the same proportion ρ of equality pairs. We fit a grid of models to the training set, select the model

∗. http://www.kamishima.net/sushi/

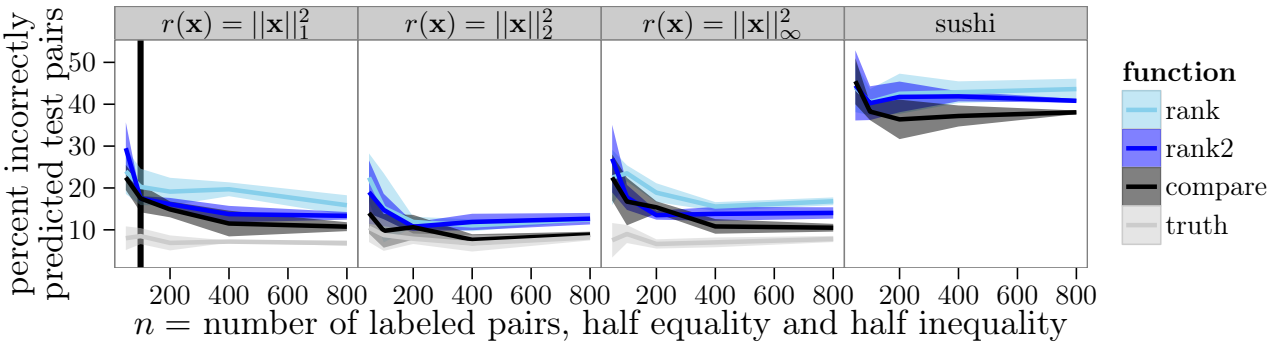

Figure 4: Test error for 3 different simulated patterns $r(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^2$ and one real sushi data set where $\mathbf{x} \in \mathbb{R}^{14}$. We randomly generated data sets with $\rho = 1/2$ equality and $1/2$ inequality pairs, then plotted test error as a function of data set size n (a vertical line shows the data set which was used in Figure 3). Lines show mean and shaded bands show standard deviation over 4 test sets.

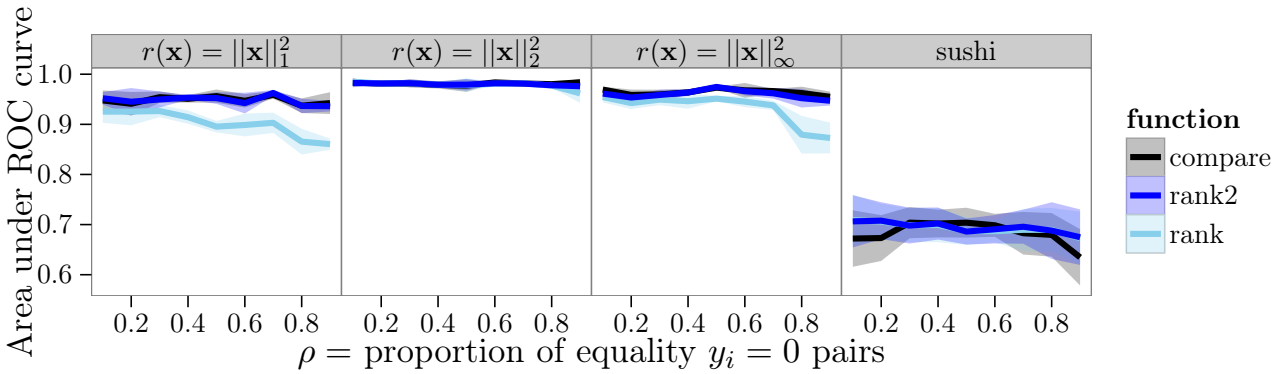

Figure 5: Area under the ROC curve (AUC) for 3 different simulated patterns $r(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^2$ and one real sushi data set where $\mathbf{x} \in \mathbb{R}^{14}$. For each data set we picked $n = 400$ pairs, varying the proportion ρ of equality pairs. We plot mean and standard deviation of AUC over 4 test sets.

with minimal zero-one loss on the validation set, and then use the test set to estimate the generalization ability of the selected model.

In Figure 4 we fixed the proportion of equality pairs $\rho = 1/2$, varied the number of training pairs $n \in \{50, \ldots, 800\}$, and calculated test error. The relative performance of the algorithms is the same as in the simulations: rank has the highest test error, rank2 does better, and the proposed SVMcompare algorithm has the lowest test error.

In Figure 5 we fixed the number of training pairs $n = 400$, varied the proportion ρ of equality pairs, and calculated test AUC. Like in the $r(\mathbf{x}) = ||\mathbf{x}||_2^2$ simulation, test AUC is about the same for each of the models considered. Perhaps this is because they are all able to learn a nearly optimal ranking function for this problem.

Overall from the sushi data, it is clear that the proposed SVMcompare model performs better than the SVMrank methods in terms of test error, and it performs as well as the SVMrank methods in terms of test AUC.

4.3 SVMcompare predict outcomes of chess games more accurately than ELO

In this ubsection, we show that our SVMcompare algorithm can be used for highly accurate prediction of the outcome of chess matches. Chess is a game between 2 players that results in a win, loss or a draw, with the most common outcome between highly ranked players in international tournaments being a draw. We wished to predict outcomes of tournament chess matches by learning a comparison function using features based on player statistics. The main statistic to quantify player rankings in FIDE (Fédération Internationale des Échecs) competitions is the ELO score. The ELO rating system is a method for calculating relative skill levels of players in competitor versus competitor games was intially proposed by Elo (1978). The Glicko rating system provides a more complex alternative (Glickman, 1999).

We downloaded the chess match dataset from Chessmetrics†, containing 1.8 million games played over the 11-year period from 1999–2009 by 54205 chess players. For each

†. http://www.chessmetrics.com/cm/

of the years 1999–2006, we consider the first four months (Jan–Apr) as a train set, and the last eight months as a test set (May–Dec). We removed all matches containing a player who had less than 10 matches against other players in the train set, to prevent our data set from containing players with very little information. We also removed all matches that contained a player's first match from the train set as we would have no information about this player. For each match i, we computed features $\mathbf{x}_i, \mathbf{x}'_i \in \mathbb{R}^{16}$ consisting of ELO scores, Glicko scores, if the player had the initial move, the percentage of instances where a player either lost to a lower ranked player, or won against a higher ranked player, the average score difference of opponents, win/loss/draw/games played raw values and percentages in addition to various other statistics. ELO scores were initially set at 1200 for all players and FIDE rules were applied to score calculations. ELO and Glicko scores were updated after every match using the PlayerRatings R package (Stephenson and Sonas, 2016).

For hyper-parameter selection, we used the first 3 months of each 12-month period. This was done for computational speed reasons. We performed cross validation splits of the first $\{0.50, 0.75, 0.80, 0.85\}$ matches in the set as our training set and the remainder as the validation set. We fit a grid of models for each linear, polynomial and Gaussian kernel to the training set to select a model with the maximum AUC on the validation set. For all kernels, the grid of cost parameters was $C \in \{10^{-20}, 10^{-18}, \ldots, 10^{0}\}$. The Gaussian kernel width was $10^{-1}, 10^{0}, 10^{1}$ and the polynomial kernel degree was $1, 2, 3, 4$.

We then used the selected hyper-parameters to train a model using the first four months (Jan–Apr) of each year, and we used the learned model to predict on the test set for that year (May–Dec). We then computed the test AUC for each of the eight years. Since there are 3 labels $y \in \{-1, 0, 1\}$ corresponding to {win, draw, loss} respectively, the baseline AUC is non-standard (not 0.5). In our baseline calculation, we obtain a FPR and TPR of 0 by predicting every observation as a negative class (0). By predicting every observation as the most common positive class, we have obtained the maximum TPR without any learning. We then obtain the FPR given all observations are predicted to be the most common positive prediction. Since we have 2 distinct positive classes, it is highly unlikely that our TPR and FPR will be 1.0 through this method.

As shown in Figure 6, the linear and polynomial SVM kernels have higher test AUC than the ELO and Glicko scoring systems. Additionally, linear and polynomial SVM models trained using only ELO and Glicko features perform worse than models using all features. This suggests that the additional features computed in our model were relevant in learning models that performed better than ELO scores. Models using a combination of ELO and Glicko features obtain an AUC that appears to be the median of the Glicko and ELO AUC values. The model appears to be learning a combination of these features and therefore obtains an AUC intermediate of ELO and Glicko scores only. The Gaussian kernel also preforms poorly with respect to the linear and polynomial kernels. This is probably because the data appears to be linear separable since the linear kernel preforms best and the most common degree selected in the polynomial kernel is 1.

Overall, our analysis of the chess match data suggests that the proposed SVMcompare model performs better than the existing state-of-the-art ELO and Glicko scores.

5. Conclusions and future work

We discussed the learning to compare problem, which has not yet been extensively studied in the machine learning literature. In Section 3.1, we proposed two different formulations for max-margin comparison, and proved their relationship in Lemma 1. It justifies our proposed SVMcompare algorithm, which uses a binary SVM dual QP solver to learn a nonlinear comparison function. In future work it will be interesting to explore the learning capability of a kernelized version of the LP (7) with slack variables added to the objective function.

Our experimental results on simulated and real data clearly showed the importance of directly modeling the equality pairs, when they are present. We showed in Figure 5 that when there are few equality pairs, as is the usual setup in learning to rank problems, the baseline SVMrank algorithm performs as well as our proposed SVMcompare algorithm. However, when there are many equality pairs, it is clearly advantageous to use a model such as SVMcompare which directly learns from the equality pairs.

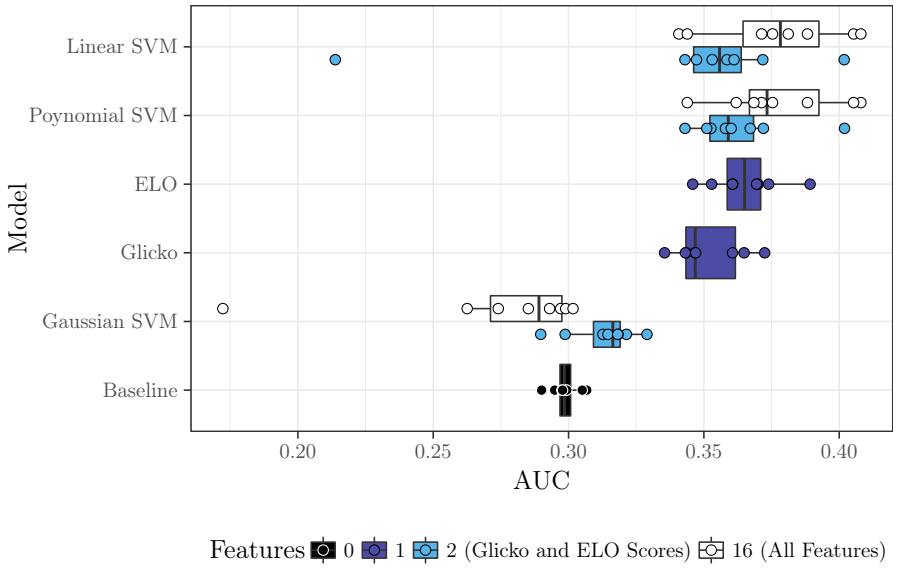

Figure 6: Test AUC for each model used after training on the first 4 months of match data in the 8 different 12-month periods. All SVM model AUCs are shown in addition to AUC of the ELO, Glicko scores and baseline. The plots in black show the baseline AUC calculated by predicting the most common positive label. The plots in dark blue are AUC values obtained from using Glicko or ELO scores only. The plots in light blue show the AUC distribution from models using only ELO and Glicko features and plots in white are AUC values from models using all computed features.

Out results also indicate that the proposed model performs better for predicting outcomes of chess matches than the current FIDE ELO player ranking system, and that incorporating additional features into out model gives an increase in accuracy.

For future work, it will be interesting to see if the same results are observed in learning to rank data from search engines. For scaling to these very large data sets, we would like to try algorithms based on smooth discriminative loss functions, such as stochastic gradient descent with a logistic loss.

Acknowledgements

DV was funded by an NSERC operating grant. TDH was funded by KAKENHI 23120004, SS by a MEXT scholarship, and MS by KAKENHI 17H01760. Thanks to Simon Lacoste-Julien and Hang Li for helpful discussions.

References

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

Wei Chu and S Sathiya Keerthi. New approaches to support vector ordinal regression. In *Proceedings of the 22nd international conference on Machine learning*, pages 145–152. ACM, 2005.

Roger R Davidson. On Extending the Bradley-Terry Model to Accommodate Ties in Paired Comparison Experiments. *Journal of the American Statistical Association*, 65(329):317–328, March 1970.

Arpad Elo. The rating of chess players, past and present. *Acro Publishing*, 1978.

Mark E Glickman. Parameter estimation in large dynamic paired comparison experiments. *Appl. Statist.*, 48(3):377–394, 1999.

Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill$^{\text{TM}}$: A Bayesian skill rating system. In *Advances in Neural Information Processing Systems*, pages 569–576, 2006.

T Joachims. Optimizing search engines using clickthrough data. In *KDD*, 2002.

T Kamishima, H Kazawa, and S Akaho. A survey and empirical comparison of object ranking methods. *Preference Learning*, pages 181–201, 2010.

Alexandros Karatzoglou, Alex Smola, Kurt Hornik, and Achim Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

Hang Li. A short introduction to learning to rank. *IEICE Transactions on Information and Systems*, E94-D(10), 2011.

Alec Stephenson and Jeff Sonas. PlayerRatings: Dynamic Updating Methods for Player Ratings Estimation (R package version 1.0-1). *CRAN*, 2016.

Vanya Van Belle, Kristiaan Pelckmans, Sabine Van Huffel, and Johan AK Suykens. Support vector methods for survival analysis: a comparison between ranking and regression approaches. *Artificial Intelligence in Medicine*, 53(2):107–118, 2011.

Ke Zhou, Gui-Rong Xue, Hongyuan Zha, and Yong Yu. Learning to rank with ties. In *Proc. ACM SIGIR 31*, SIGIR '08, pages 275–282, New York, NY, 2008.