A support vector method for ranking minimizing the number of swapped pairs

ARTICLE · JULY 2014

3 AUTHORS, INCLUDING:

Jorge Díez
University of Oviedo
49 PUBLICATIONS **335** CITATIONS
SEE PROFILE

Juan José del Coz
University of Oviedo
51 PUBLICATIONS **297** CITATIONS
SEE PROFILE

A support vector method for ranking minimizing the number of swapped pairs

J. Díez, J.J. del Coz, A. Bahamonde*
Artificial Intelligence Center
University of Oviedo at Gijón
E33271 – Spain
{jdiez,juanjo,antonio}@aic.uniovi.es

Abstract

Learning tasks where the set Y of classes has an ordering relation arise in a number of important application fields. In this context, the loss function may be defined in different ways, ranging from multiclass classification to ordinal or metric regression. However, to consider only the ordered structure of Y, a measure of goodness of a hypothesis h has to be related to the number of pairs whose relative ordering is swapped by h. In this paper, we present a method, based on the use of a multivariate version of Support Vector Machines (SVM) that learns to order minimizing the number of swapped pairs. Finally, using benchmark datasets, we compare the scores so achieved with those found by other alternative approaches.

1 Introduction

In this paper we deal with the following learning problem: a training set $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, where each example is described by a vector $\boldsymbol{x}_i$ of an input space X and a class y_i of an output space Y endowed with an ordering relation. Moreover, the order plays a central role in the problem, so much so that the aim is to learn a hypothesis $h : X \to \mathbb{R}$ such that the relative ordering of $(h(\boldsymbol{x}_1), \ldots, h(\boldsymbol{x}_n))$ be as coherent as possible with the relative ordering of $(y_1, \ldots, y_n)$. There are several closely related learning problems that deal with orderings; sometimes they receive different names in the literature like ranking, preferences, ordinal regression, or simply ordering. Throughout this paper we will refer to this learning task as the ranking problem.

The order appears explicitly in a great number of learning applications, see [1], that include information retrieval [2, 3], recommender systems [4], and beef cattle selection [5].

When Y is included in a metric space like $\mathbb{R}$, an obvious candidate to learn from $\mathcal{S}$ is a metric regression algorithm; however, sometimes the accuracy is so poor that we must relax the strong metric requirements of regression in order to achieve some useful hypothesis appealing only to the ordering structure of classes. This is the case, for instance, when the aim is to make sensory analysis of food from data provided by a panel of consumers [6, 7, 8, 9].

It is important to emphasize here that the ordering of Y can be only a partial ordering; that is, there may exist pairs of Y elements not comparable. So, as was pointed out in [10], when the data collected in $\mathcal{S}$ comes from rankings given by different people, it is necessary to avoid assuming that, for instance, a rating of 7 means the same thing to every user: the scales of different users are typically incomparable.

An important issue when we are learning to order is to fix the way in which we are going to measure the quality of the result. Depending of the approach used, there is a variety of options ranging from

*http://www.aic.uniovi.es/MLGroup

multiclassification to ordinal or metric regression. But multiclassification is clearly inadequate, since it does not consider the ordering of classes. On the other hand, regression approaches are closer to fitting the ranking setting, although the optimization searched by regression is not exactly a measure of the coherence of observed and predicted rankings [11].

In this paper we are going to use the loss function defined by [12] that for a hypothesis h returns the number of pairs $(h(\boldsymbol{x}_i), h(\boldsymbol{x}_j))$ whose relative ordering is swapped with respect to the relative ordering of (y_i, y_j). Formally, given $\mathcal{S}$ we look for hypothesis h (from a given hypothesis space) that minimize the average loss extended over the set of independently identically distributed (i.i.d.) test sets $\mathcal{S}'$, where the loss is given by the probability of swapped pairs:

$$\Delta_{SP}(h, \mathcal{S}') = \Pr(h(\boldsymbol{x}'_i) \leq h(\boldsymbol{x}'_j) | y'_i > y'_j) = \frac{\sum_{i,j:y'_i > y'_j} I_{h(\boldsymbol{x}'_i) \leq h(\boldsymbol{x}'_j)}}{\sum_{i,j} I_{y'_i > y'_j}}. \quad (1)$$

This function is analogous to the loss used by [10]. Additionally, it is noteworthy that when the number of classes is two, to minimize Eq. (1) is equivalent to maximize the area under the ROC curve (AUC) for binary classification datasets.

Following [12] we will review a direct implementation that solves the ranking problem. Unfortunately, this approach leads us to deal with datasets of size n^2 when the original size of $\mathcal{S}$ is only n. This mean that some applications become intractable, although other times this approach was successfully used [5, 6, 7, 8, 3, 9]. To alleviate the difficulties caused by the size of datasets, the main problem is that (as happens with the AUC) Herbrich's loss function can not be expressed as a sum of disagreements or errors produced by each input $\boldsymbol{x}_i \in X$. Therefore, a reasonable option to tackle the ranking problem would consist in using the multivariate SVM presented by [13]. In fact, we are going to introduce an extension of the approach given by Joachims to maximize the AUC such that we will be able to handle output spaces with an arbitrary number of elements.

Additionally, we discuss the relation between ordinal regression and the approach followed in this paper. In the literature on the subject, we underline the work started by [14], and continued by [15]. Here the hypothesis projects inputs into contiguous real intervals, one for each rank or class. The optimization problem attached aims to maximize the margin of the decision borders between consecutive ranks penalizing each transgression taking into account the ordered semantics of the output space Y. In all cases, the size of the optimization problems is linear in the number of training examples. This approach constitutes an interesting heuristic to find a hypothesis that minimizes the number of swapping pairs, but the explicit objective is not the same. The main difference between the approach proposed here and ordinal regression is analogous to the difference between maximizing the AUC and minimizing the error rate in binary classification tasks. A detailed statistical analysis of this situation is given in [16].

The paper is organized as follows. First, we review the related work available in the literature. In the third section we present the method proposed to deal with the ranking problem. Finally, we report some experimental results.

2 Learning to order

In this section we briefly review two alternative methods for learning to order using support vectors. The first one optimizes the loss function of Eq. (1), but it is difficult to use in practice because the size of the optimization program. The second method optimizes a loss function closely related to regression, so it is not clear that it works well on ranking tasks.

2.1 An approach that uses pairs of inputs

In [12] the authors present a support vector method for learning to order that reformulates the original task. The core idea is that if a hypothesis $h : X \to \mathbb{R}$ is linear and has to fulfill that $h(\boldsymbol{x}_i) > h(\boldsymbol{x}_j)$ since $y_i > y_j$, then this constraint is equivalent to

$$h(\boldsymbol{x}_i) > h(\boldsymbol{x}_j) \Leftrightarrow h(\boldsymbol{x}_i - \boldsymbol{x}_j) > 0. \quad (2)$$

Notice that Eq. (2) converts ordering constraints into classification (with one class) constraints, but now the input space is $X \times X$ and each pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is represented by the difference $\boldsymbol{x}_i - \boldsymbol{x}_j$.

According to this approach, the goal is to find a hypothesis h such that solves the following convex optimization program:

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \frac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w}\rangle + C \sum_{i,j:y_i>y_j} \xi_{ij} \\ s.t. \quad & \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle - \langle \boldsymbol{w}, \boldsymbol{x}_j\rangle \geq +1 - \xi_{ij}, \\ & \xi_{ij} \geq 0, \quad \forall i,j : y_i > y_j. \end{aligned} \tag{3}$$

This method can be easily extended to nonlinear kernels [12]. Therefore, we have available an elegant support vector method to solve the ranking problem when it is possible to deal with training sets of size n^2. However, this approach is hard to be readdressed to one with a training set of size n again. The difficulty stems from the fact that the loss function Eq. (1) is not rewritable as a linear combination of functions that depend only of each input $\boldsymbol{x}_i \in X$. Therefore, it is not straightforward to write a convex optimization program in such conditions. A practical approach can be implemented using only a small part of the whole set of constraints.

2.2 Ordinal Regression

In [15], the authors follow the work of [14] and present one algorithm for ordinal regression based on support vectors that outperforms other ordinal regression methods. This algorithm tries to find a hyperplane with director vector $\boldsymbol{w}$ and a set of thresholds $\{b_1, \ldots, b_{r-1}\}$ such that $\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle$ are separated by the thresholds in classes. So b_i defines the border between classes i and $i+1$. In other words, $\boldsymbol{w}$ ranks inputs and each rank is determined by a set of contiguous intervals: $\{(-\infty, b_1), [b_1, b_2), \ldots, [b_{r-1}, +\infty)\}$.

To solve this problem, the authors set a convex optimization program trying to maximizing the margins between ranks, penalizing margin violations with two groups of slack variables: ξ_i^j computes the deviation of the $\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle$ from the lower margin $(b_j - 1)$ of its rank and all upper ranks $(\forall j \geq y_i)$; and ξ_i^{*j} computes the violation of the $\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle$ from the upper margin $(b_j + 1)$ of its rank and all lower ranks $(\forall j \leq y_i)$. Formally, the primal problem is the following:

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{b},\boldsymbol{\xi},\boldsymbol{\xi^*}} \; & \frac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w}\rangle + C \sum_{i=1}^{n} \left(\sum_{y:y\geq y_i} \xi_i^y + \sum_{y:y_i>y} \xi_i^{*y} \right) \\ s.t. \; & \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle - b_y \leq -1 + \xi_i^y, \;\; \forall i, y : r > y \geq y_i, \xi_i^y \geq 0, \\ & \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle - b_y \geq 1 - \xi_i^{*y}, \;\; \forall i, y : y_i > y \geq 1, \xi_i^{*y} \geq 0. \end{aligned} \tag{4}$$

Notice that if $r = n$, the number of constrains is also $O(n^2)$. The sum of slack variables $\sum_{i=1}^{n} \left(\sum_{y:y\geq y_i} \xi_i^y + \sum_{y:y_i>y} \xi_i^{*y} \right)$ is an upper bound of the absolute deviation; see [15].

3 Proposed Method

In this section we are going to present the method proposed in this paper to tackle the ranking problem settled in the Introduction. The method is a generalization of Joachims' approach to optimize the AUC in binary classification [13] that uses a structural SVM described in [17].

3.1 Optimizing non-linear performance measures with SVM

In [17], the authors present a kind of support vector machines, called *structural* SVM, which are able to learn problems where the output space contains complex objects like trees or a sequences. One of the advantages of this method is that it allows to find hypothesis that optimize measures of performance that can not be expressed as a linear combination of loss functions defined over individual instances. In fact, the most striking novelty is that structural SVM formulates learning problems considering not the individual predictions for each input, but the prediction of the whole set of inputs simultaneously. In [13] it is described an adaptation of this approach specially devised for optimizing performance measures that can be computed from contingency tables as the F_1-score.

Algorithm 1 Algorithm for solving quadratic program of multivariate $\text{SVM}^{\Delta}_{multi}$

Input: $\bar{\boldsymbol{x}} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and $\bar{y} = (y_1, \ldots, y_n)$, ϵ
$CS \leftarrow \varnothing$
repeat
  $\bar{y}' \leftarrow \text{argmax}_{\bar{y}^* \in \bar{Y}} \{\Delta(\bar{y}^*, \bar{y}) + \boldsymbol{w}^T \Psi(\bar{\boldsymbol{x}}, \bar{y}^*)\}$
  $\xi' \leftarrow \Delta(\bar{y}', \bar{y}) - \boldsymbol{w}^T [\Psi(\bar{\boldsymbol{x}}, \bar{y}) - \Psi(\bar{\boldsymbol{x}}, \bar{y}')]$
  $\xi \leftarrow \max_{\bar{y}'' \in CS} \{0, \Delta(\bar{y}'', \bar{y}) - \boldsymbol{w}^T [\Psi(\bar{\boldsymbol{x}}, \bar{y}) - \Psi(\bar{\boldsymbol{x}}, \bar{y}'')]\}$
  if $\xi' \geq \xi + \epsilon$ **then**
    $CS \leftarrow CS \cup \{\bar{y}'\}$
    $\boldsymbol{w} \leftarrow$ optimize $\text{SVM}^{\Delta}_{multi}$ objective over CS
  end if
until CS has not changed during last iteration
Output: $\boldsymbol{w}$

Joachims includes a section to describe an algorithm to optimize directly the area under the ROC curve (the AUC) in binary classification.

The general setting of structural SVM is the following. Let us assume that a learning task tries to find a hypothesis $h_{\boldsymbol{w}} : X \to Y = \{-1, +1\}$ with

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T \phi(\boldsymbol{x})), \tag{5}$$

such that $h_{\boldsymbol{w}}$ has to minimize a nonlinear loss function Δ. The approach of the structural SVM consists in facing this learning task as a multivariate prediction of the whole dataset. Hence, we will consider, instead of $h_{\boldsymbol{w}}$, hypothesis $\bar{h}_{\boldsymbol{w}}$ that map a tuple of inputs $\bar{\boldsymbol{x}} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ to a tuple of outputs $\bar{y} = (y_1, \ldots, y_n)$. To link this approach to Eq. (5), we will use discriminant functions of the form:

$$\bar{h}_{\boldsymbol{w}}(\bar{\boldsymbol{x}}) = \underset{\bar{y}^* \in \bar{Y}}{\text{argmax}} \{\boldsymbol{w}^T \Psi(\bar{\boldsymbol{x}}, \bar{y}^*)\}, \tag{6}$$

where $\boldsymbol{w}$ is the weight vector, and $\Psi(\bar{\boldsymbol{x}}, \bar{y}^*)$ computes a combined feature representation of inputs $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and outputs $(y_1^*, \ldots, y_n^*)$. The argmax of the expression in Eq. (5) can be computed efficiently when the definition of Ψ is given by

$$\Psi(\bar{\boldsymbol{x}}, \bar{y}^*) = \sum_{i=1}^{n} y_i^* \phi(\boldsymbol{x}_i). \tag{7}$$

In fact, in this case,

$$\bar{h}_{\boldsymbol{w}}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \text{sign}(\boldsymbol{w}^T \phi(\boldsymbol{x}_1), \ldots, \boldsymbol{w}^T \phi(\boldsymbol{x}_n)). \tag{8}$$

In this context, the multivariate prediction problem entails solving the following optimization:

$$\begin{aligned} \min_{\boldsymbol{w}, \xi} \quad & \frac{1}{2} \langle \boldsymbol{w}, \boldsymbol{w} \rangle + C\xi \\ s.t. \quad & \boldsymbol{w}^T [\Psi(\bar{\boldsymbol{x}}, \bar{y}) - \Psi(\bar{\boldsymbol{x}}, \bar{y}')] \geq \Delta(\bar{y}', \bar{y}) - \xi, \ \forall \bar{y}' \in \bar{Y} \setminus \bar{y}. \end{aligned} \tag{9}$$

Despite the apparent intractability of this problem due to the exponential number of constraints, following the sparse approach of [13, 17], the Algorithm 1 solves it in polynomial time. For this purpose, it is necessary that the argmax of the first step of each iteration must be computed in polynomial time. The core idea of the algorithm is to increase iteratively the set of constraints CS. In each iteration, the most violated constraint $\bar{y}'$ is added to CS, and a new weight vector is computed. The process ends when no new constraint has to be added. The weight vector returned by the Algorithm 1 solves the original problem, see Eq. (5).

3.2 Learning to rank minimizing the number of swapping pairs

Let us recall that we are dealing with a dataset $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} \subset X \times Y$. In order to simplify the explanations of this section, we are going to assume that $Y = \{1, \ldots, r\}$; moreover, we will use a linear kernel, although the extension to other kernels is straightforward. Following [13] we are going to describe the adaptation of $\text{SVM}^{\Delta}_{multi}$ that minimizes the number of swapping pairs, the resulting algorithm will be denoted by $\text{SVM}^{\Delta_{SP}}_{multi}$.

Algorithm 2 Algorithm for computing argmax

Input: $\bar{\boldsymbol{x}} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and $\bar{y} = (y_1, \ldots, y_n)$
for $i = 1, \ldots, n$ **do**
  if $y_i > 1$ **then** $v_i^+ \leftarrow -0.25 + \boldsymbol{w}^T \boldsymbol{x}_i$
  end if
  if $y_i < r$ **then** $v_i^- \leftarrow 0.25 + \boldsymbol{w}^T \boldsymbol{x}_i$
  end if
end for
$V \leftarrow$ sort together all v_i^+ and v_i^- values
for $k = 1, \ldots, r$ **do**
  $s_n[k] \leftarrow 0$
  $s_p[k] \leftarrow n_{k+1} + \ldots + n_r$
  $\#neg[k] \leftarrow n_1 + \ldots + n_{k-1}$
  $\#pos[k] \leftarrow n_{k+1} + \ldots + n_r$
end for
for each v in V **do**
  if v is v_{index}^+ **then**
    $c_{index} \leftarrow \#neg[class(v)] - 2s_n[class(v)]$
    for $k = 1, \ldots, class(v) - 1$ **do** $s_p[k] \leftarrow s_p[k] - 1$
    end for
  else /*v is v_{index}^-*/
    $d_{index} \leftarrow \#pos[class(v)] - 2s_p[class(v)]$
    for $k = class(v) + 1, \ldots, r$ **do** $s_n[k] \leftarrow s_n[k] + 1$
    end for
  end if
end for
Output: $(c_1, \ldots, c_n)$ and $(d_1, \ldots, d_n)$

We will use the approach of Section 2.1, and hence we reduce the ranking problem to a classification problem where inputs are all pairs $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with $y_i > y_j$. In this way, we are building a new dataset $\bar{\bar{S}} \subset \bar{\bar{X}} \times \bar{\bar{Y}}$ where each input $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ has class $y_{ij} = +1$ and it is described by $\boldsymbol{x}_{ij} = \boldsymbol{x}_i - \boldsymbol{x}_j$. Notice that if $n_1, n_2, \ldots, n_r$ stand for the number of inputs of $\mathcal{S}$ of classes $1, 2, \ldots, r$ respectively, then total number of pairs of $\bar{\bar{S}}$ is given by:

$$N = \sum_{k,l:k>l} n_k n_l. \tag{10}$$

In this case Eq. (6), using Eq. (7), for a given $\boldsymbol{w}$ and $\bar{\bar{\boldsymbol{x}}}$, returns

$$\bar{\bar{y}}' = \bar{\bar{h}}_{\boldsymbol{w}}(\bar{\bar{\boldsymbol{x}}}) = \underset{\bar{\bar{y}}^* \in \bar{\bar{Y}}}{\text{argmax}} \{\boldsymbol{w}^T \Psi\left(\bar{\bar{\boldsymbol{x}}}, \bar{\bar{y}}^*\right)\} \tag{11}$$

where we have that for each pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$, y'_{ij} is 1 if and only if $\boldsymbol{w}^T \boldsymbol{x}_i > \boldsymbol{w}^T \boldsymbol{x}_j$, and the class is -1 otherwise; see Eq. (2). Additionally, the number of disagreements in classification between $\bar{\bar{y}}'$ and $\bar{\bar{y}}$ is the number of swapped pairs:

$$SP = \Delta_{SP}(\bar{\bar{y}}', \bar{\bar{y}}) = \sum_{i,j:y_i>y_j} \frac{1}{2}\left(y_{ij} - y'_{ij}\right) = \sum_{i,j:y_i>y_j} \frac{1}{2}\left(1 - y'_{ij}\right). \tag{12}$$

The advantage of this approach is that now we can avoid the explicit construction of the set of all pairs $\bar{\bar{X}}$. The reason is that we will see that any $\bar{\bar{y}}' \in \bar{\bar{Y}}$ can be represented, in the Algorithm 1, by a couple of vectors of size $n = |\mathcal{S}|$. These vectors are defined by

$$\begin{aligned} c'_i &= \sum_{j:y_i>y_j} y'_{ij}, \\ d'_i &= \sum_{j:y_i<y_j} y'_{ji}. \end{aligned} \tag{13}$$

Notice that these definitions imply that

$$\begin{aligned} c'_i &= 0 \qquad \forall i : y_i = 1, \\ d'_i &= 0 \qquad \forall i : y_i = r. \end{aligned} \tag{14}$$

Using c' and d' it is possible, in fact, to obtain the number of swapped pairs, since

$$\Delta_{SP}(\bar{\bar{y}}', \bar{\bar{y}}) = \sum_{i,j:y_i>y_j} \frac{1}{2}\left(y_{ij} - y'_{ij}\right) = \frac{1}{2}\left[\sum_{i,j:y_i>y_j} y_{ij} - \sum_{i,j:y_i>y_j} y'_{ij}\right] = \frac{1}{2}\left[\sum_{i:y_i>1} c_i - \sum_{i:y_i>1} c'_i\right]. \tag{15}$$

Given the first expression of Eq. (14), we can extend the sums to the whole dataset, then we have that

$$\Delta_{SP}(\bar{\bar{y}}', \bar{\bar{y}}) = \frac{1}{2}\left[\sum_{i=1}^{n} c_i - \sum_{i=1}^{n} c'_i\right] = \frac{1}{2}\sum_{i=1}^{n}(c_i - c'_i). \tag{16}$$

It is easy to show that $\sum_{i=1}^{n} c'_i = \sum_{i=1}^{n} d'_i$; therefore, Δ_{SP} can also be expressed in terms of c' y d':

$$\Delta_{SP}(\bar{\bar{y}}', \bar{\bar{y}}) = \frac{1}{4}\sum_{i=1}^{n}\left[(c_i + d_i) - (c'_i + d'_i)\right]. \tag{17}$$

To compute Ψ, see Eq. (7), we proceed in a similar way:

$$\Psi(\bar{\bar{\boldsymbol{x}}}, \bar{\bar{y}}') = \sum_{i,j:y_i>y_j} y'_{ij}\boldsymbol{x}_{ij} = \sum_{i,j:y_i>y_j} y'_{ij}\boldsymbol{x}_i - \sum_{i,j:y_i>y_j} y'_{ij}\boldsymbol{x}_j = \sum_{i:y_i>1} c'_i\boldsymbol{x}_i - \sum_{j:y_j<r} d'_j\boldsymbol{x}_j. \tag{18}$$

Again, taking into account Eq. (14) we can rewrite these expressions as follows:

$$\Psi(\bar{\bar{\boldsymbol{x}}}, \bar{\bar{y}}') = \sum_{i=1}^{n} c'_i\boldsymbol{x}_i - \sum_{j=1}^{n} d'_j\boldsymbol{x}_j = \sum_{i=1}^{n}(c'_i - d'_i)\boldsymbol{x}_i. \tag{19}$$

Using the representation of $\bar{\bar{Y}}$ vectors just described, the function argmax, used in Algorithm 1, can be computed by the Algorithm 2 according to the following result that it is a generalization of the Lemma 2 of [13].

Lemma 1. *The Algorithm 2 computes the vectors c'_i and d'_i that represent*

$$\bar{\bar{y}}' = \underset{\bar{\bar{y}}^* \in \bar{\bar{Y}}}{argmax}\{\Delta_{SP}(\bar{\bar{y}}^*, \bar{\bar{y}}) + \boldsymbol{w}^T\Psi\left(\bar{\bar{\boldsymbol{x}}}, \bar{\bar{y}}^*\right)\}, \tag{20}$$

in time $\max\{O(n\log n), O(nr)\}$.

Proof. If $\bar{\bar{y}}'$ is defined by Eq. (20), then it is equal to:

$$\underset{\bar{\bar{y}}^* \in \bar{\bar{Y}}}{\text{argmax}} \sum_{i,j:y_i>y_j} \left(\frac{1}{2}(1 - y^*_{ij}) + y^*_{ij}\boldsymbol{w}^T(\boldsymbol{x}_i - \boldsymbol{x}_j)\right) = \underset{\bar{\bar{y}}^* \in \bar{\bar{Y}}}{\text{argmax}} \sum_{i,j:y_i>y_j} y^*_{ij}\left(-\frac{1}{2} + \boldsymbol{w}^T(\boldsymbol{x}_i - \boldsymbol{x}_j)\right) =$$

$$\underset{\bar{\bar{y}}^* \in \bar{\bar{Y}}}{\text{argmax}} \sum_{i,j:y_i>y_j} y^*_{ij}\left((\boldsymbol{w}^T\boldsymbol{x}_i - \frac{1}{4}) - (\boldsymbol{w}^T\boldsymbol{x}_j + \frac{1}{4})\right).$$

Hence, given that $y'_{ij} \in \{-1, +1\}$,

$$y'_{ij} = \text{sign}\left((\boldsymbol{w}^T\boldsymbol{x}_i - \frac{1}{4}) - (\boldsymbol{w}^T\boldsymbol{x}_j + \frac{1}{4})\right), \tag{21}$$

when $y_i > y_j$. For this reason, the Algorithm 2 defines the vectors v_i^+ and v_i^-, then they are ordered, and finally the algorithm counts the number of times that v_i^+ is greater than (and lower than) v_j^- for every pair of indices where $y_i > y_j$. □

4 Experimental Results

In order to evaluate the benefits of our approach we conducted a battery of experiments. The datasets used were obtained from a website maintained by L. Torgo[1]. Their descriptions, and the sizes of these train/test sets are described in Table 1.

[1] http://www.liacc.up.pt/∼ltorgo/Regression/DataSets.html

Table 1: Average over 20 trials of the percentage of swapped pairs achieved by $\text{SVM}^{\Delta_{SP}}_{multi}$, Herbrich's approach with 5 comparisons for each input (Herbrich$_5$) and Chu's algorithm using 5 equal frecuency bins (SVOR$_5$). We used bold face to indicate the lowest value among the three algorithms. The symbol † (and ††) is used to indicate that the difference observed between $\text{SVM}^{\Delta_{SP}}_{multi}$ and these other methods are significant using a T-test with a p-value threshold of 0.05 (0.01 respectively).

Dataset	#atr.(#nominal)	(train/ test)	SVOR$_5$	Herbrich$_5$	$\text{SVM}^{\Delta_{SP}}_{multi}$
Diabetes	2 (0)	(30 / 13)	31.50 ±7.79	**30.74** ±8.58	30.82 ±7.07
Pyrimidines	27 (0)	(50 / 24)	22.37†† ±5.12	20.42†† ±3.31	**18.86** ±3.43
Servo	4 (4)	(100 / 67)	20.61†† ±3.07	18.18†† ±2.21	**16.51** ±2.75
Triazines	60 (0)	(100 / 86)	**34.33** ±2.87	36.64†† ±2.96	34.84 ±2.93
Wpbc	32 (0)	(130 / 64)	**36.17** ±3.73	36.73 ±3.89	**36.17** ±3.93
MachineCPU	6 (0)	(150 / 59)	14.76†† ±2.04	**13.82** ±1.82	13.96 ±2.24
AutoMPG	7 (3)	(200 / 192)	9.44† ±0.63	9.37† ±0.73	**9.22** ±0.61
Boston	13 (1)	(200 / 306)	13.15†† ±0.74	12.42 ±0.85	**12.37** ±1.08
Stock	9 (0)	(200 / 750)	10.25†† ±0.69	6.12†† ±0.59	**5.36** ±0.64
Abalone	8 (1)	(200 /3977)	19.50 ±0.52	**19.41** ±0.60	19.55 ±0.69
Bank8	8 (0)	(200 /7992)	**6.92** ±0.25	7.04 ±0.24	7.00 ±0.51
Bank32	32 (0)	(200 /7992)	23.48†† ±0.96	23.63† ±1.36	**23.07** ±0.84

As was said in sections 2.1 and 2.2, the algorithms of Herbrich and Chu can not deal with ranking problems directly, because it implies to handle n^2 constraints. We made for those algorithms different training sets reducing constraints' number. For Herbirch's approach (in the following Herbrich$_5$) we created training sets where each input was paired with 5 randomly selected inputs with different ranks. For Chu's algorithm (in the following SVOR$_5$) the output space $Y \subseteq \mathbb{R}$ was split into a 5 intervals or bins in such a way that each bin includes the same number of training examples.

All algorithms were trained using a gaussian kernel. For tuning the parameter g and the regularization factor C, with each training set, we performed search grid with a 5-fold CV. The search was done on a 5×4 coarse grid linearly spaced in the region $\{\log_{10} C, \log_{10} g : -3 \leq \log_{10} C \leq 1, -3 \leq \log_{10} g \leq 0\}$. The parameters, so determined, with best scores (lower percentages of swapped pairs) were used to finally compute the train/test result. The scores shown in Table 1 are the average percentage of swapped pairs attained in 20 repetitions of a hold out experiment where test sets are always ranking sets.

Experiments were conducted to compare the results of $\text{SVM}^{\Delta_{SP}}_{multi}$ versus those achieved by algorithms discussed in Section 2: Herbrich$_5$ and SVOR$_5$. If we compare the results in pairs, we observe that $\text{SVM}^{\Delta_{SP}}_{multi}$ is "clearly" better than both of them: 8 out of 12 times the average of swapped pairs of $\text{SVM}^{\Delta_{SP}}_{multi}$ win those reported by SVOR$_5$ (1 tie), 7 of them are significant according to the T-test; and 9 out of 12 times $\text{SVM}^{\Delta_{SP}}_{multi}$ win scores reported by Herbrich$_5$ (no ties), 6 of them are significant. When our system loses, the differences are not significant.

If we compare the performance of the systems over results showed in Table 1 we appreciate significant differences in favour of our system using a T-test. The p-value thresholds are 0.02043 (with SVOR$_5$) and 0.01089 (with Herbrich$_5$). In other words, the experiments carried out seem to show that, as expected, when the performance measure is the number of swapped pairs, trying to optimize that measure returns better results than using an ordinal regression algorithm or a simplifiyed version of Herbrich's algorithm.

5 Conclusions

We have presented a support vector method that directly optimizes the natural measure of goodness when we are dealing with ordering or ranking learning tasks: the number of swapped pairs. This measurement was used previously by [12] yielding to a classification problem that uses a dataset formed by all pairs of inputs with different classes. Therefore, the ranking problem may result

intractable when the size of the original dataset is large. The method proposed here, $\text{SVM}^{\Delta_{SP}}_{multi}$, solves this problem computing the result of all pairs of inputs with different classes but using only the original dataset. It was developed as an extension of the $\text{SVM}^{\Delta}_{multi}$ presented by [13] that uses a structural SVM presented in [17].

The ranking problem can be tackled using heuristic approaches, as do ordinal regression algorithms. Comparing experimentally the scores of both approaches, our algorithm has been shown to be a valid alternative. In fact, the main difference between our approach and ordinal regression is a generalization of the difference between optimizing the area under the ROC curve (AUC) and the error rate when the number of ordered classes is exactly two; see [16, 13].

Acknowledgements

The research reported in this paper has been supported in part under Spanish Ministerio de Ciencia y Tecnología (MCyT) and Feder grant TIN2005-08288. The authors would like to thank Thorsten Joachims for the source code of $\text{SVM}^{\Delta}_{multi}$ and for his useful comments.

References

[1] Shivani Agarwal, Corinna Cortés, and Ralf Herbrich. Workshop on learning to rank. In *Neural Information Processing Systems*, 2005.

[2] Susan Dumais, Krishna Bharat, Thorsten Joachims, and Andreas Weigend. Workshop on implicit measures of user interests and preferences. In *ACM SIGIR Conference*, 2003.

[3] Thorsten Joachims. Evaluating retrieval performance using clickthrough data. In *ACM Conference on Knowledge Discovery and Data Mining*, 2002.

[4] Paul Resnick and Hal Varian. Introduction to special section on recommender systems. *Communicatons of the ACM*, 40(3):56–58, 1997.

[5] Antonio Bahamonde, Gustavo F. Bayón, Jorge Díez, José Ramón Quevedo, Oscar Luaces, Juan José del Coz, Jaime Alonso, and Félix Goyache. Feature subset selection for learning preferences: a case study. In *International Conference on Machine Learning*, pages 49–56, 2004.

[6] Juan José del Coz, Gustavo F. Bayón, Jorge Díez, Oscar Luaces, Antonio Bahamonde, and Carlos Sañudo. Trait selection for assessing beef meat quality using non-linear svm. In *Neural Information Processing Systems*, pages 321–328, 2004.

[7] Jorge Díez, Gustavo F. Bayón, José Ramón Quevedo, Juan José del Coz, Oscar Luaces, Jaime Alonso, and Antonio Bahamonde. Discovering relevancies in very difficult regression problems: applications to sensory data analysis. In *European Conference on Artificial Intelligence*, pages 993–994, 2004.

[8] Jorge Díez, Juan José Del Coz, Carlos Sañudo, Pere Albertí, and Antonio Bahamonde. A kernel based method for discovering market segments in beef meat. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 462–469, 2005.

[9] Oscar Luaces, Gustavo F. Bayón, José Ramón Quevedo, Jorge Díez, Juan José Del Coz, and Antonio Bahamonde. Analyzing sensory data using non-linear preference learning with feature subset selection. In *European Conference of Machine Learning*, pages 286–297, 2004.

[10] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10(0):243–270, 1999.

[11] Shyamsundar Rajaram and Shivani Agarwal. Generalization bounds for k-partite ran. In *Workshop on Learning to Rank at NIPS*, pages 18–23, 2005.

[12] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in large margin classifiers*, pages 115–132, 2000.

[13] Thorsten Joachims. A support vector method for multivariate performance measures. In *International Conference on Machine Learning*, 2005.

[14] Amnon Shashua and Anat Levin. Ranking with large margin principle: Two approaches. In *Neural Information Processing Systems*, 2003.

[15] Wei Chu and S. Sathiya Keerthi. New approaches to support vector ordinal regression. In *International Conference on Machine Learning*, pages 145–152, 2005.

[16] Corinna Cortés and Mehryar Mohri. Auc optimization vs. error rate minimization. In *Neural Information Processing Systems*, 2003.

[17] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(0):1453–1484, 2005.