Mach Learn (2009) 75: 129–165
DOI 10.1007/s10994-008-5097-z

An efficient algorithm for learning to rank from preference graphs

Tapio Pahikkala · Evgeni Tsivtsivadze · Antti Airola · Jouni Järvinen · Jorma Boberg

Received: 9 March 2007 / Revised: 5 December 2008 / Accepted: 10 December 2008 / Published online: 9 January 2009

Abstract In this paper, we introduce a framework for regularized least-squares (RLS) type of ranking cost functions and we propose three such cost functions. Further, we propose a kernel-based preference learning algorithm, which we call RankRLS, for minimizing these functions. It is shown that RankRLS has many computational advantages compared to the ranking algorithms that are based on minimizing other types of costs, such as the hinge cost. In particular, we present efficient algorithms for training, parameter selection, multiple output learning, cross-validation, and large-scale learning. Circumstances under which these computational benefits make RankRLS preferable to RankSVM are considered. We evaluate RankRLS on four different types of ranking tasks using RankSVM and the standard RLS regression as the baselines. RankRLS outperforms the standard RLS regression and its performance is very similar to that of RankSVM, while RankRLS has several computational benefits over RankSVM.

Keywords Ranking · Preference learning · Preference graph · Regularized least-squares · Kernel methods

Editors: Thomas Gärtner and Gemma C. Garriga.

T. Pahikkala (✉) · E. Tsivtsivadze · A. Airola · J. Järvinen · J. Boberg
Turku Centre for Computer Science (TUCS), Department of Information Technology, University of Turku, Joukahaisenkatu 3-5 B, 20520 Turku, Finland
e-mail: tapio.pahikkala@utu.fi

E. Tsivtsivadze
e-mail: evgeni.tsivtsivadze@utu.fi

A. Airola
e-mail: antti.airola@utu.fi

J. Järvinen
e-mail: jouni.jarvinen@utu.fi

J. Boberg
e-mail: jorma.boberg@utu.fi

1 Introduction

Preference learning has recently received a lot of attention in the machine learning literature—we refer to Fürnkranz and Hüllermeier (2005) for a compact and illuminating summary of the preference learning tasks. Preference learning can be considered as a task in which the aim is to learn a function capable of arranging data points according to a given preference relation. When comparing two data points, the function is able to evaluate whether the first point is preferred over the second one.

We assume that we are given a training data consisting of input data points and their pairwise preferences that are used to train a supervised learning algorithm for the prediction of the preference relations among unseen data points. We also consider the scoring setting in which we are given a training data consisting of scored data points, that is, each input data point is assigned a real valued score indicating its goodness. The pairwise preference between these data points are then determined by the differences of the scores. This type of preference learning tasks are often cast into classification tasks so that each pair of data points, in which one point is preferred over the other, is used as a training data point whose class indicates the direction of the preference (for recent in depth theoretical analysis of ranking algorithms see, e.g. Clémençon et al. 2005; Agarwal 2006; Cortes et al. 2007b). For example, Herbrich et al. (1999) used this approach together with support vector machines (SVM) for ordinal regression tasks. This method is often referred to as RankSVM. A similar SVM adaptation was made by Joachims (2002) to rerank the results obtained from a search engine.

Recently, it has been shown that the RLS classifiers (see e.g. Rifkin 2002), also known as the least-squares SVMs (Suykens and Vandewalle 1999), have a classification performance similar to the regular SVMs (see e.g. Rifkin 2002; Gestel et al. 2004; Zhang and Peng 2004). In Pahikkala et al. (2007b), we proposed RankRLS, an algorithm that learns preferences from scored data, in which for each input data point x we have a real value score s attached. If an input data point x is preferred over x', the difference to be regressed is $s - s'$, where s and s' are the scores of x and x', respectively. It was shown that the computational complexity of training RankRLS is equal to the complexity of training an RLS regressor for the same data set. Namely, the computational complexity of training RankRLS was shown to be $O(m^3)$ in the dual form, where m is the number of input data points in the training data, and $O(mn^2 + n^3)$ in the primal form, where n is the dimensionality of the feature space. A similar algorithm with equal computational times was at the same time independently proposed by Cortes et al. (2007a, 2007b) and called MPRank. They also provide a thorough theoretical analysis of the generalization error of the MPRank algorithm using stability bounds. The difference between RankRLS and MPRank is that the former includes in the training process only such input data point pairs that are relevant to the task in question, while the latter is defined to include every possible input pair. For example, in many document retrieval tasks, each input data point consists of a query and a document, and hence there should be no preferences between such inputs that are associated to different queries. Otherwise, the objective functions of RankRLS and MPRank are the same, but the closed form solution are derived in a slightly different way.

In this paper, we extend our consideration of RankRLS so that it can be used to learn not only from scored data, but also from a given sequence of pairwise preferences and their magnitudes when the scores are not given. Moreover, we introduce a general framework for RLS based ranking cost functions and propose three different specializations for it. Note that we only consider the case in which we learn so-called scoring function that maps each possible input to a real value. The function then induces a total order for the inputs. The direction of preference between two inputs is obtained by comparing their predicted scores.

Cost functions that are variations of the least-squares cost have certain computational advantages compared to the other types of costs, such as the hinge cost used with SVM. For example, the least-squares-based learning methods can be expressed using matrix calculus which makes them simple to implement and analyze. Moreover, RankRLS can be trained so that its computational complexity depends on the number of data points instead of the number of observed pairwise preferences. This is an important advantage, because the number of preferences is usually proportional to the square of the number of individual data points. Furthermore, there often exists efficient shortcut methods for calculating the cross-validation performance for the least-squares based learners and for parameter selection (see e.g. Pahikkala et al. 2006a, 2007a, 2008a; Rifkin and Lippert 2007a). RLS-based learning algorithms can also be extended for large-scale learning using the subset of regressors method (see e.g. Quiñonero-Candela et al. 2007; Tsivtsivadze et al. 2008). Further advantages include the possibility to learn several functions in parallel as considered by Rifkin and Klautau (2004). In this paper, we present efficient algorithms for training RankRLS both in small and large scale as well as for both linear and nonlinear learning tasks. We also present fast algorithms for cross-validation, parameter selection, and multiple output learning.

We also make a thorough consideration of the circumstances under which the fast cross-validation, parameter selection, and multiple output learning algorithms make RankRLS preferable to RankSVM from computational complexity point of view. Namely, the benefits and drawbacks of RankRLS and RankSVM in both small-scale and large-scale learning tasks are investigated and so are both the linear and nonlinear learning problems. For example, training a single instance of a RankSVM may be faster than training a single instance of RankRLS in the linear learning tasks, but the efficient cross-validation, parameter selection, and multiple output learning algorithms make RankRLS in many situations much faster method to use than RankSVM. This is especially the case if nonlinear kernel functions are used and if cross-validation is used for performance estimation.

In our experiments, we test our ranking algorithms with different tasks. The pairwise preferences in all of the tasks are induced by a scoring of the data points. Two of the considered tasks are the ranking of dependency parses and document retrieval. Comparing parses generated for different sentences or documents returned for different queries would, of course, make little sense. This kind of preference structure is typical for label ranking problems in which the aim is to rank for each object a set of labels, and this is the approach we use for these tasks. In the former task, the objects and labels are sentences and their parse candidates, and in the latter they are queries and documents retrieved by them. The other two tasks considered are document classification and collaborative filtering which we consider as object ranking problems in which the aim is to learn a given preference structure over all data points. Further, the document retrieval and binary classification tasks are bipartite ranking problems, that is, there are only two possible score values for the data points. In contrast, the parse ranking and collaborative filtering tasks have real-valued scores.

In the label ranking tasks, we test whether some of the input pairs that are not relevant to the learning problem to be solved would be beneficial if included in the training process. Our results suggest that this is not the case. Moreover, we compare the three proposed cost functions on the ranking tasks. In all experiments, we use as baselines the RankSVM method and the ordinary RLS regressor trained to regress the scores of the data points. The ranking for the data points is then obtained from the regressed scores. We observe that performances of RankRLS and RankSVM are very similar in all considered tasks, with no statistically significant differences observed, although RankRLS has many computational benefits over RankSVM as discussed in the paper. RankRLS and RankSVM perform better or as well as the RLS regressor.

This paper is organized as follows. In Sect. 2, we present a formal introduction on the preference learning tasks under consideration, and define the proposed method RankRLS. Section 3 considers computationally efficient algorithms for training and validation. We summarize the computational benefits of RankRLS and compare them to those of RankSVM in Sect. 4. RankRLS is experimentally evaluated in Sect. 5. We conclude the paper in Sect. 6.

2 The RankRLS algorithm

First, we give a formulation of preference learning problems in Sect. 2.1. Section 2.2 concerns the framework of regularized kernel methods. In Sect. 2.3, we introduce the RankRLS algorithm. Finally, we consider three different variations of the least-squares ranking cost function in Sect. 2.4.

2.1 Preference learning

Let $\mathcal{X}$, called the input space, denote the set of input data points which we call in the following shortly as inputs. We assume that we are given a sequence

$$X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$$

of inputs. Moreover, let

$$E = (e_1, \ldots, e_l)^{\mathrm{T}} \in (\mathcal{X} \times \mathcal{X} \times \mathbb{R}^+)^l$$

be a sequence of observed preferences between the inputs, that is, $e_i = (x_h, x_j, y_i)$, where $1 \leq h, j \leq m$ and $h \neq j$, indicating that x_h is preferred over x_j with magnitude y_i. The magnitudes may be supplied in the training data, or in case such information is not available, magnitude 1 can be given for each pairwise preference. Altogether, we define the training data to be the tuple

$$G = (X, E).$$

G can be considered as a preference graph in which the inputs x_h, are the vertices and e_i are the edges. By the definition of E, there may be multiple observed preferences with possibly differing magnitudes between two inputs. Thus, G is a multigraph. Finally, we define $\mathcal{G}$ to be the set of all possible graphs of the above defined type.

We also consider a special case of preference learning setting in which E is not given but so-called scoring information of the inputs is available and the preferences are determined by this scoring. Thus, in the scoring setting we assume that we are given a sequence

$$S = (s_1, \ldots, s_m)^{\mathrm{T}} \in \mathbb{R}^m$$

of real valued scores corresponding to the input sequence $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$. In this case, we can obtain a sequence of observed preferences so that for each pair of inputs x_h and x_j, where $1 \leq h, j \leq m$ and $h \neq j$, there exists an edge $e_i = (x_h, x_j, y_i)$, where $y_i = s_h - s_j$, if and only if $s_h > s_j$. Unless stated otherwise, we do not assume that the observed preferences are determined by a scoring information.

Fürnkranz and Hüllermeier (2005) divided the preference learning tasks into two categories, namely, to the tasks of learning object preferences and learning label preferences. Here, we consider a similar type of division. The cases in which the aim is to learn a given

preference structure over all inputs are considered as object ranking problems. For example, any binary classification task can be considered as an object ranking task in which the aim is to rank all inputs belonging to the positive class above the ones belonging to the negative class. We define label ranking tasks to be such in which the inputs are comprised of an object and its label. In this case, the observed preferences make sense, that is, are relevant only between such inputs that are associated with the same object. In many document retrieval tasks, for example, each input consists of a query and a document. In this case, the documents can be considered as the labels of the queries. Clearly, there should be no preferences between such inputs that are associated to different queries. If the preferences of a label ranking task are induced by a scoring of the inputs, the preferences between inputs associated to different objects are considered to be irrelevant to the task in question and they are not included in the preference graph. Formally, the sequence of preferences is obtained from the scoring so that for each pair of inputs x_h and x_j, where $1 \leq h, j \leq m$ and $h \neq j$, there exists an edge $e_i = (x_h, x_j, y_i)$, where $y_i = s_h - s_j$, if and only if $s_h > s_j$ and the preference is relevant to the task under consideration. In our notation, we make no difference between the object and label ranking settings, and we assume that the sequence of observed preferences is formed according to the setting in question.

Let $\mathbb{R}^{\mathcal{X}}$ denote the set of all functions from $\mathcal{X}$ to $\mathbb{R}$, and let $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}}$ be the hypothesis space. A natural way to measure how well a function $f \in \mathcal{H}$ agrees with preferences of E is to define a disagreement error:

$$D(f, G) = \frac{1}{2l} \sum_{i=1}^{l} (1 - \text{sign}(g(e_i))), \tag{1}$$

where $e_i = (x_h, x_j, y_i)$ for some $h \neq j$, $1 \leq h, j \leq m$,

$$g(e_i) = f(x_h) - f(x_j),$$

and sign is the signum function

$$\text{sign}(r) = \begin{cases} 1 & \text{if } r > 0, \\ -1 & \text{if } r \leq 0. \end{cases}$$

Note that in the disagreement error (1), we omit the magnitudes y_i. Nevertheless, we take advantage of the magnitude information when we design the learning algorithms.

Training can be considered as a process of selecting a function from the hypothesis space that best performs the learning task in question. Thus, learning can be viewed as an algorithm $\mathcal{A}$ that for a given preference graph G selects an appropriate function f from $\mathcal{H}$. Formally,

$$\mathcal{A} : \mathcal{G} \to \mathcal{H}. \tag{2}$$

2.2 Regularized kernel methods

Here, we consider the selection of the suitable function $f \in \mathcal{H}$. Let $\mathcal{X}$ denote the input space, which can be any set, and $\mathcal{F}$ denote an inner product space we call the feature space. For any mapping

$$\Phi : \mathcal{X} \to \mathcal{F},$$

the inner product

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

of the mapped inputs is called a kernel function. We define the symmetric kernel matrix $K \in \mathbb{R}^{m \times m}$, where $\mathbb{R}^{m \times m}$ denotes the set of real $m \times m$-matrices, as

$$K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \cdots & k(x_m, x_m) \end{pmatrix}$$

for the sequence $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$ of inputs. Unless stated otherwise, we assume that the kernel matrix is strictly positive definite, that is, $A^{\mathrm{T}} K A > 0$ for all $A \in \mathbb{R}^m$, $A \neq 0$. The strict positive definiteness of the kernel matrix K can be ensured, for example, by adding ϵI to K, where $I \in \mathbb{R}^{m \times m}$ is the identity matrix and ϵ is a small positive real number.

Following Schölkopf et al. (2001), we define the reproducing kernel Hilbert space (RKHS) determined by the input space $\mathcal{X}$ and the kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ as

$$\mathcal{H}_{k,\mathcal{X}} = \left\{ f \in \mathbb{R}^{\mathcal{X}} \;\middle|\; f(x) = \sum_{i=1}^{\infty} \beta_i k(x, x_i), \beta_i \in \mathbb{R}, x_i \in \mathcal{X}, \|f\|_k < \infty \right\},$$

where

$$\|f\|_k = \sqrt{\sum_{i,j=1}^{\infty} \beta_i \beta_j k(x_i, x_j)}$$

denotes the norm of the function f in the RKHS. Using $\mathcal{H}_{k,\mathcal{X}}$ as our hypothesis space, we next define the cost functions that we can use to measure how well the hypotheses fit our training data G. Overloading our notation, we denote

$$f(X) = (f(x_1), \ldots, f(x_m))^{\mathrm{T}}$$

for the sequence X of inputs and a hypothesis $f \in \mathcal{H}_{k,\mathcal{X}}$. We use cost functions of type

$$c : \mathbb{R}^m \times \mathcal{G} \to \mathbb{R}$$

to assign a value

$$c(f(X), G) \tag{3}$$

on the predictions $f(X)$, training data $G = (X, E)$, and a candidate hypothesis $f \in \mathcal{H}_{k,\mathcal{X}}$ that measures how well f fits G.

We now consider the following variational problem as a realization of algorithm (2) that we use to select an appropriate hypothesis f from $\mathcal{H}_{k,\mathcal{X}}$ for training data G. Namely, we consider an algorithm

$$\mathcal{A}(G) = \operatorname*{argmin}_{f \in \mathcal{H}_{k,\mathcal{X}}} J(f, G), \tag{4}$$

where

$$J(f, G) = c(f(X), G) + \lambda \|f\|_k^2 \tag{5}$$

and $\lambda > 0$ is a regularization parameter. The first term measures the performance of a candidate hypothesis on the training data and the second term, called the regularizer, measures the complexity of the hypothesis with the RKHS norm.

According to the representer theorem (Schölkopf et al. 2001), any minimizer $f \in \mathcal{H}_{k,\mathcal{X}}$ of (5) admits the representation of the following form:

$$f(x) = \sum_{i=1}^{m} a_i k(x, x_i), \tag{6}$$

where $a_i \in \mathbb{R}$ and k is the kernel function associated with the RKHS mentioned above. Let

$$A = (a_1, \ldots, a_m)^{\mathrm{T}} \in \mathbb{R}^m$$

be a vector consisting of the values that determine the solution (6). By overloading our notation, we write $k(x, X) = (k(x, x_1), \ldots, k(x, x_m)) \in (\mathbb{R}^m)^{\mathrm{T}}$, where $x \in \mathcal{X}$ and $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$. Using this type of matrix notation, we can write

$$f(x) = \sum_{i=1}^{m} a_i k(x, x_i) = k(x, X)A. \tag{7}$$

Similarly, the column vector $f(X) \in \mathbb{R}^m$, that contains the predictions for the inputs obtained with the function f, is

$$f(X) = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{pmatrix} = \begin{pmatrix} k(x_1, X)A \\ \vdots \\ k(x_m, X)A \end{pmatrix} = KA. \tag{8}$$

Further, according to (6) and to the definition of the RKHS norm, the regularizer can be written as

$$\lambda \|f\|_k^2 = \lambda \sum_{i,j=1}^{m} a_i a_j k(x_i, x_j) = \lambda A^{\mathrm{T}} K A. \tag{9}$$

Next, we consider realizations for the cost function.

2.3 Regularized least-squares regression of preferences

In order to construct a regularized kernel method that would learn the preferences defined on the training data $G = (X, E)$, we have to define an appropriate cost function. A natural way to encode the preference information into a cost function is to use the disagreement error (1) for the preferences:

$$c(f(X), G) = \sum_{i=1}^{l} (1 - \operatorname{sign}(g(e_i))), \tag{10}$$

where $e_i = (x_h, x_j, y_i)$ and $g(e_i) = f(x_h) - f(x_j)$. Note that in (1), $\frac{1}{2l}$ can be considered as a constant, and hence it can be omitted from (10). It is well-known that the use of this type of cost functions leads to intractable optimization problems. Therefore, instead of using (10),

we use functions approximating it. We adopt an approach similar to the regularized least-squares classification (Rifkin 2002) which has been shown to have a performance similar to that of the support vector machine classifiers. That is, we use the following type of square cost as an approximation of (10):

$$c(f(X), G) = \sum_{i=1}^{l} w_i^2 (z_i - g(e_i))^2, \tag{11}$$

where $z_i, w_i \geq 0$ are real-valued parameters. When defining the actual cost functions, we fix the parameters z_i and w_i to be constants or dependent on the magnitude y_i. We observe that the cost is a sum of parabolae whose zeros and widths are determined by z_i and w_i, respectively. Intuitively, the parameters w_i can be considered as importance weights of the edges e_i, since the cost function (11) is more sensitive to the predictions $g(e_i)$ of the edges having a large value of w_i than to those having a small value. In Sect. 2.4, we present three different specializations of the cost function by setting the parameters.

Before presenting the actual learning algorithm, we introduce some notation. Let $M \in \mathbb{R}^{m \times l}$ be a matrix whose rows and columns are indexed by the vertices and edges of the preference graph, respectively, and its entries are given by

$$M_{h,i} = \begin{cases} w_i & \text{if } e_i = (x_h, x_j, y_i) \text{ for some } j \neq h, \\ -w_i & \text{if } e_i = (x_j, x_h, y_i) \text{ for some } j \neq h, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

In the graph theory (see e.g. Brualdi and Ryser 1991), the matrix M is sometimes called the oriented incidence matrix of a graph and the product $L = MM^{\mathrm{T}}$ is called the Laplacian matrix of the graph. We also note that Laplacian matrix is always positive semidefinite, since it is a product of a real-valued matrix with its own transpose. We consider M as a linear transformation from $\mathbb{R}^m$ to $\mathbb{R}^l$, that is, it can be used to map the vector $f(X)$ consisting of the values $f(x_h)$, $1 \leq h \leq m$, to a vector $M^{\mathrm{T}} f(X)$ containing the values $w_i g(e_i)$, $1 \leq i \leq l$. Further, let us write

$$N = (w_1 z_1, \ldots, w_l z_l)^{\mathrm{T}}. \tag{13}$$

Using these notations, we can rewrite the cost function (11) in a matrix form as

$$c(f(X), G) = (N - M^{\mathrm{T}} f(X))^{\mathrm{T}} (N - M^{\mathrm{T}} f(X)). \tag{14}$$

The next theorem characterizes a method called RankRLS.

Theorem 1 *Let $G = (X, E)$ and let*

$$\mathcal{A}(G) = \operatorname*{argmin}_{f \in \mathcal{H}_{k,\mathcal{X}}} J(f, G), \tag{15}$$

where

$$J(f, G) = \sum_{i=1}^{l} w_i^2 (z_i - g(e_i))^2 + \lambda \| f \|_k^2, \tag{16}$$

is the algorithm under consideration. A coefficient vector $A \in \mathbb{R}^m$ that determines a minimizer of (16) is

$$A = (MM^T K + \lambda I)^{-1} MN. \tag{17}$$

Proof According to the representer theorem, the minimizer of (16) is of the form (6), that is, the problem of finding the optimal hypothesis can be solved by finding the coefficients a_h, $1 \leq h \leq m$.

According to (8), the vector consisting of the input predictions can be written as $f(X) = KA$. We use the matrix M to transform the input predictions to a vector of prediction differences $M^{\mathrm{T}}KA$. Then, the ith entry of the vector $M^{\mathrm{T}}KA$ contains the value $w_i g(e_i)$. We use the matrix forms (9) and (14) to rewrite the algorithm (15) as follows:

$$\mathcal{A}(G) = \operatorname*{argmin}_{A \in \mathbb{R}^m} J(A, G),$$

where

$$J(A, G) = (N - M^{\mathrm{T}}KA)^{\mathrm{T}}(N - M^{\mathrm{T}}KA) + \lambda A^{\mathrm{T}}KA.$$

We take the derivative of $J(A, G)$ with respect to A:

$$\begin{aligned}\frac{d}{dA}J(A, G) &= -2KM(N - M^{\mathrm{T}}KA) + 2\lambda KA \\ &= -2KMN + 2(KMM^{\mathrm{T}}K + \lambda K)A.\end{aligned}$$

We set the derivative to zero and solve with respect to A:

$$\begin{aligned}A &= (KMM^{\mathrm{T}}K + \lambda K)^{-1}KMN \\ &= (MM^{\mathrm{T}}K + \lambda I)^{-1}MN,\end{aligned}$$

where the last equality follows from our assumption of the kernel matrix being strictly positive definite. □

We refer to (17) as the dual solution of RankRLS in contrast to the primal solution considered in Sect. 3.1.

The multiplication of M with N can be performed in $O(l)$ time, since M contains only $2l$ nonzero elements. This is also the complexity of calculating the Laplacian matrix $L = MM^{\mathrm{T}}$ of the preference graph as can be shown in the following way. First, we note (see e.g. Brualdi and Ryser 1991) that, if $h \neq j$, $L_{h,j} = -\sum_i w_i^2$, where i goes through the indices of the edges that are between the hth and jth vertex. Further, $L_{h,h} = \sum_i w_i^2$, where i goes through the indices of the edges starting from or ending to the hth vertex. Therefore, for constructing the matrix L, four operations per edge are needed. For example, an edge starting from the h vertex and ending to the jth vertex affects the entries $L_{h,h}$, $L_{h,j}$, $L_{j,h}$, and $L_{j,j}$. Thus, we have:

$$\text{the complexity of calculating the products } MM^{\mathrm{T}} \text{ and } MN \text{ is } O(l). \tag{18}$$

The subsequent multiplication of L with K and the inversion of the matrix $MM^{\mathrm{T}}K + \lambda I$ can be done in $O(m^3)$ time. Note that, in the time complexities considered in this paper, we do not count the complexity of calculating the kernel matrix, because it depends on the kernel function used. Thus, provided that the kernel matrix is already calculated,

$$\text{the complexity of dual RankRLS training is } O(m^3 + l). \tag{19}$$

Note that the preference graph determined by the sequence of observed preferences E is a multigraph, and hence the number l of the pairwise preferences may not necessarily be

dominated by the term m^3 in (19). However, in the scoring setting, which we discuss more in Sect. 3, we have $l = O(m^2)$, because the number of edges is bounded above by the number of all possible input pairs.

2.4 Specializations of the cost function

We now consider three different specializations of the least-squares cost function (11) for approximating the disagreement error. The variations are depicted in Fig. 1. In the first version, we just set $z_i = 1$ and $w_i = 1$ for all $1 \leq i \leq l$:

$$c(f(X), G) = \sum_{i=1}^{l} (1 - g(e_i))^2. \tag{20}$$

This is a cost function that, similarly to the disagreement error (1), simply ignores the preference magnitudes treating every input pair in E as if their magnitudes would be equal to one.

The second approach uses the magnitude information to determine the zero points, that is, we set $z_i = y_i$ and $w_i = 1$ for all $1 \leq i \leq l$:

$$c(f(X), G) = \sum_{i=1}^{l} (y_i - g(e_i))^2. \tag{21}$$

This cost function is equal to the one proposed by us in Pahikkala et al. (2007b). It has many computational advantages in case preference magnitudes are induced by a scoring of

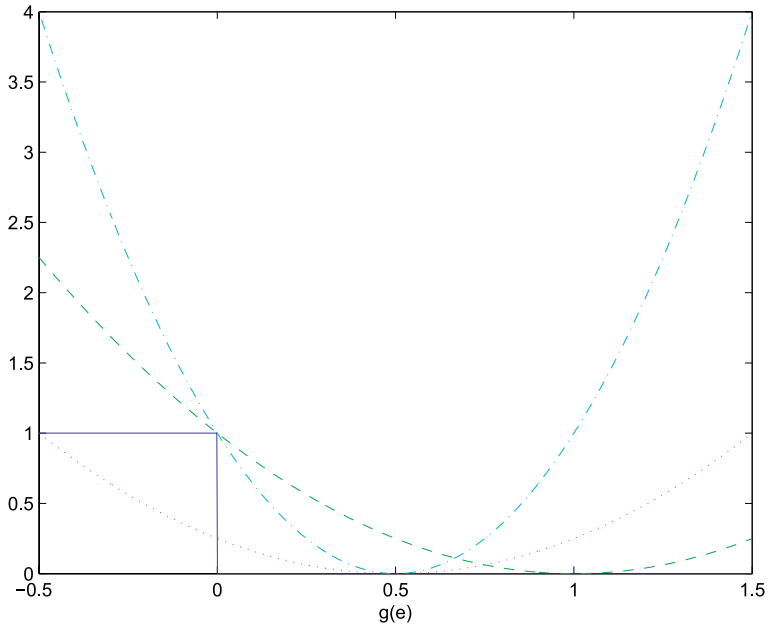

Fig. 1 The x-axis represents the value of g and the y-axis the value of the cost functions when the preference magnitude is 0.5. The disagreement cost is depicted with a *solid line* and the functions (20), (21), and (22) with '–', '. . .', and '-·-', respectively

the inputs as discussed in Sect. 3. A disadvantage of this cost is that it does not form an upper bound on the disagreement error, and therefore this cost function is harder to analyze theoretically.

The third cost function also uses the magnitude information to determine the zero points, and thus we set $z_i = y_i$. In addition, the width parameters are set to $w_i = 1/y_i$ which ensures that the disagreement error is bounded above by this cost function:

$$c(f(X), G) = \sum_{i=1}^{l} \frac{1}{y_i^2}(y_i - g(e_i))^2. \tag{22}$$

Moreover, this cost can be intuitively justified so that the RankRLS algorithm is allowed to make larger prediction errors for edges having a large magnitude than for edges having a small magnitude. This is, because even a small prediction error can reverse the direction of preference for an edge with a small magnitude but not the with a large one.

We observe that the functions (20), (21), and (22) are equal if we have only preferences but not magnitudes, that is, $y_i = 1$ for all $1 \leq i \leq l$. This is the case especially in bipartite ranking, that is, when the preferences are induced by the scoring in which there are only two different scores, say 1 and 0. If $y_i = 1$ for all $1 \leq i \leq l$, also the function (21) forms an upper bound on the disagreement error. Therefore, one may derive results similar to those of Agarwal and Niyogi (2005) and Cortes et al. (2007b) to analyze the generalization performance of the ranking algorithms.

The possibility to give importance weights to the preferences with the parameters w_i also enables the design of cost functions that are more suitable for label ranking tasks than those using only the magnitudes. Consider, for example, the task of parse ranking in which the aim is to rank for each sentence the set of parses according to some preference criterion. The parse candidate sets can be of different size for each sentence, while each sentence is equally important. In this case, it may be beneficial to use a normalized version of the cost function in which each edge associated to a sentence has a weight equal to the inverse of the number of edges associated to the same sentence.

3 Efficient implementations

In this section, we consider ways to speed up the training process of RankRLS. We pay special attention to the preference learning task in the scoring setting using the magnitude preserving cost function (21). In the scoring setting, the inputs $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$ and the corresponding scores $S = (s_1, \ldots, s_m)^{\mathrm{T}} \in \mathbb{R}^m$ are given. Recall the definitions (12) and (13) of M and N, respectively. We observe, that in case we use (21), the following equation holds:

$$N = M^{\mathrm{T}} S. \tag{23}$$

Therefore, the matrix form (14) can be rewritten as

$$c(f(X), G) = (S - f(X))^{\mathrm{T}} M M^{\mathrm{T}} (S - f(X)). \tag{24}$$

Note that we use the notation $c(f(X), G)$, while we also have the sequence of scores S. Further, while considering the efficient implementations with the cost function (21) in the scoring setting, we also allow preferences with magnitude zero. Namely, we consider cases in which the scoring S induces exactly one preference between every input x_h and x_j, where

$h \neq j$, even if x_h and x_j have equal scores. If the scores are equal, the direction of the edge does not matter, and hence only one edge is needed between the corresponding vertices in the preference graph also in this case. These preferences with zero magnitude are generated only for efficiency reasons and they are ignored when the performance of a ranking algorithm is measured with the disagreement error.

When there are preferences between every input, we can take advantage of the regularities of the matrix M in order to speed up the computations. We first consider the case in which every possible preference induced by the scoring is relevant to the task in question, as is often the case in object ranking tasks. We observe that we can write

$$MM^{\mathrm{T}} = D - PP^{\mathrm{T}}, \tag{25}$$

where $D \in \mathbb{R}^{m \times m}$ is a diagonal matrix whose every diagonal entry is equal to m, and $P \in \mathbb{R}^m$ is a vector whose every entry is equal to 1. It is much more efficient to perform matrix multiplications with the form $D - PP^{\mathrm{T}}$ than with MM^{T}, because P is an m-dimensional vector and D is a diagonal matrix, and thus having only m nonzero elements.

All preferences induced by the scoring are not always relevant to the task in question. In label ranking tasks, for example, we may want to exclude the preferences that are not relevant to the task, that is, the input pairs in which the inputs are associated to different objects. Next, we consider the removal of this type of irrelevant preferences. Assume there are q objects in the training data and each of the m inputs are associated to one of the objects. In the task of parse ranking, for example, an object is a sentence and an input is comprised of a sentence and a parse candidate generated for the sentence. Each parse is associated only with the sentence it is generated for and the aim is to learn to rank the parses for each sentence separately. The scoring induces preferences also between parses that are associated with different sentences but they are considered to be irrelevant to the task of parse ranking. We redefine $P \in \mathbb{R}^{m \times q}$ to be a matrix whose rows are indexed by the inputs and the columns are indexed by the q objects. The value of $P_{i,j}$ is defined to be 1 if the ith input is associated with the jth object and 0 otherwise. Further, we redefine D to be a diagonal matrix whose entries are defined as follows. If the ith input is associated to a certain object, then the ith diagonal element of D is the number of inputs that are associated to the same object. For example, assume that our training data consists of altogether 5 inputs and two objects. The first object is associated with the first two inputs and the second object with the last three inputs. Then, the matrices P and D are

$$P = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{pmatrix},$$

respectively. Now, we observe that MM^{T} can again be written as in (25). Similarly to the object ranking case, the matrix P contains only m nonzero elements, and hence the matrix multiplications involving P are as efficient to compute as with the object ranking case.

Further, provided that (23) and (25) hold, solving the dual form (17) involves calculating

$$MM^{\mathrm{T}}K = DK - PP^{\mathrm{T}}K$$

and

$$\begin{aligned} MN &= MM^{\mathrm{T}}S \\ &= DS - PP^{\mathrm{T}}S. \end{aligned}$$

These can be done in $O(m^2)$ and $O(m)$ time, respectively, because there are exactly m nonzero elements in both D and P. Therefore, the cubic complexity of the matrix inversion dominates the computation time of dual RankRLS training, and instead of (19),

$$\begin{aligned} &\text{the complexity of solving (17) using the cost function (21)} \\ &\text{in the scoring setting is } O(m^3). \end{aligned} \tag{26}$$

In some cases, we may also want to exclude the tied input pairs from the training process, that is, the ones whose both inputs belong to the same equivalence class as determined by the scoring. In this case, it is also possible to construct a form analogous to the one presented in (25) that can be efficiently used in computations. However, presenting it would lead to very technical and detailed considerations, and hence we leave it out from this paper.

The rest of this section is organized as follows. Section 3.1 concerns the primal form of RankRLS and its efficient implementation when using scored training data. In Sect. 3.2, we consider a way to train RankRLS simultaneously with several values of the regularization parameter. Computationally efficient cross-validation algorithms are proposed for label ranking in Sect. 3.3 and for object ranking in Sect. 3.4. Finally, Sect. 3.5 considers a large-scale training algorithm based on sparse approximation.

3.1 Primal RankRLS

In some cases, the number m of inputs $x_1, \ldots, x_m$ in the training data is much larger than the number of dimensions n in the feature space $\mathcal{F}$, that is, $n < m$ and $\mathcal{F} = \mathbb{R}^n$. Then, the sequence of mapped inputs is a matrix

$$H = (\Phi(x_1), \ldots, \Phi(x_m)) \in \mathbb{R}^{n \times m}$$

and the function (6) minimizing (5) can be equivalently expressed as

$$f(x) = \Phi(x)^{\mathrm{T}} HA = \Phi(x)^{\mathrm{T}} W, \tag{27}$$

where

$$W = HA$$

denotes the n-dimensional normal vector of the hyperplane corresponding to the RankRLS solution in the feature space, and A is the vector that determines the function (6). Output prediction for unseen inputs is more efficient with (27) than with (6) if $n < m$ and the mapping is fast to compute.

We next show that, if $n < m$, also the training process can be performed in a more efficient way than with dual RankRLS (17). We call this method the primal version of RankRLS. With the primal version, the computational complexity of the training process becomes more dependent on the dimensionality n of the feature space rather than on the number of inputs m.

Now we may write the algorithm (15) as

$$\mathcal{A}(G) = \operatorname*{argmin}_{W \in \mathbb{R}^n} J(W, G),$$

where

$$J(W, G) = (N - M^{\mathrm{T}} H^{\mathrm{T}} W)^{\mathrm{T}} (N - M^{\mathrm{T}} H^{\mathrm{T}} W) + \lambda W^{\mathrm{T}} W.$$

We take the derivative of $J(W, G)$ with respect to W:

$$\begin{aligned} \frac{d}{dW} J(W, G) &= -2HM(N - M^{\mathrm{T}} H^{\mathrm{T}} W) + 2\lambda W \\ &= -2HMN + 2(HMM^{\mathrm{T}} H^{\mathrm{T}} + \lambda I)W. \end{aligned}$$

Then, we set the derivative to zero and solve it with respect to W:

$$W = (HMM^{\mathrm{T}} H^{\mathrm{T}} + \lambda I)^{-1} HMN. \tag{28}$$

The computational complexity of the matrix inversion in (28) is in this case $O(n^3)$. Recall from (18) that constructing the matrices MM^{T} and MN needs $O(l)$ time. The other dominant operations involved in (28) are the multiplication of H with MM^{T} and HMM^{T} with H^{T} which require $O(m^2 n)$ and $O(n^2 m)$ time, respectively. Alternatively, one can first multiply H with M in $O(nl)$ time, because M contains only $2l$ nonzero elements, and then HM with its transpose in $O(n^2 l)$ time. In this case, the dominant terms are $O(n^3)$ and $O(n^2 l)$. Therefore we have:

$$\text{the complexity of calculating (28) is } O(n^3 + \min(n^2 m + m^2 n + l, n^2 l)). \tag{29}$$

However, there are some special cases in which the matrix multiplications can be performed more efficiently, as shown in the following.

Let us consider ranking in the scoring setting using the magnitude preserving cost function (21), that is, (23) and (25) hold. Then, the matrix $HMM^{\mathrm{T}} H^{\mathrm{T}}$ can be computed efficiently from

$$HMM^{\mathrm{T}} H^{\mathrm{T}} = HDH^{\mathrm{T}} - (HP)(P^{\mathrm{T}} H^{\mathrm{T}})$$

and HMN from

$$\begin{aligned} HMN &= HMM^{\mathrm{T}} S \\ &= H(DS - P(P^{\mathrm{T}} S)). \end{aligned}$$

The multiplication HDH^{T} needs $O(n^2 m)$ time, because D is a diagonal matrix. Further, the multiplication of H with P can be performed in $O(nm)$ time, because H has n rows and there are exactly m nonzero elements in P. The other complexities can be analyzed analogously. Therefore, we have:

$$\begin{aligned} &\text{the complexity of calculating (28) using the cost function (21)} \\ &\text{in the scoring setting is } O(n^3 + n^2 m), \end{aligned} \tag{30}$$

where the first term corresponds to the matrix inversion involved in (28) and the second to the matrix multiplications.

3.2 Efficient regularization and learning multiple outputs

For simplicity, we assume in this section that the equations (23) and (25) hold, that is, the cost function (21) is used in the scoring setting. Generalizations to cases in which the assumption does not hold can also be made but we omit their consideration from this paper, because their presentations would be too long and technical.

As noted in Sect. 2, the Laplacian matrix $D - PP^{\mathrm{T}}$ of a preference graph is positive semidefinite and the kernel matrix K is assumed to be strictly positive definite. Therefore, it can be shown that the matrix $(D - PP^{\mathrm{T}})K$ is diagonalizable and has real non-negative eigenvalues (see Horn and Johnson 1985, p. 465). By performing the eigen decomposition of $(D - PP^{\mathrm{T}})K$, it is possible to calculate the solutions (17) of dual RankRLS for several values of the regularization parameter λ with only small increase in the computational cost compared to calculating with just one value. Let us consider the eigen decomposition $V\Lambda V^{-1} = (D - PP^{\mathrm{T}})K$, where V is the matrix of eigenvectors and Λ is a diagonal matrix containing the corresponding eigenvalues. The decomposition and the inversion V^{-1} can be calculated in $O(m^3)$ time, and hence the complexity is not worse than that given in (26). Let $Q = V^{-1}(D - PP^{\mathrm{T}})S$, where the matrix products and subtractions can be computed in $O(m^2)$ time. Then, the solution for a regularization parameter value λ can be calculated from

$$A = V(\Lambda + \lambda I)^{-1}Q, \tag{31}$$

in $O(m^2)$ time, since $\Lambda + \lambda I$ is a diagonal matrix and Q is an m-dimensional vector.

An analogous approach can be used also in the primal form (28) by calculating the matrix $H(D - PP^{\mathrm{T}})H^{\mathrm{T}}$, its eigen decomposition $V\Lambda V^{\mathrm{T}}$, and the vector $Q = V^{\mathrm{T}}H(D - PP^{\mathrm{T}})S$ in altogether $O(n^3 + n^2m)$ time. In this case, the solutions for different values of regularization parameter can be subsequently obtained in $O(n^2)$ time, since Q is in an n-dimensional vector.

We now consider learning multiple outputs simultaneously, that is, we assume that we have multiple scores per each input. Analogously to the standard RLS, instead of having a single column matrix S for the outputs, we now have a $m \times v$-matrix S, where v is the number of outputs. We observe that only the time complexity of calculating Q and the subsequent use of (31) for different values of the regularization parameter depend on v. In the dual case, the complexity of calculating Q is $O(m^2v)$. Therefore, training RankRLS in the dual case needs altogether $O(m^3 + m^2v)$ time in the first phase. Subsequently, $O(m^2v)$ time is needed per each different value of the regularization parameter λ. In the primal case, the calculation of Q needs $O(n^2v + mnv)$ time, and hence the time complexity of the first phase in the primal case is $O(n^3 + n^2m + n^2v + mnv)$. Subsequently, the solution for a regularization parameter value can be calculated in $O(n^2v)$ time.

3.3 Cross-validation for label ranking

In Pahikkala et al. (2006a), we described an efficient method for calculating hold-out estimates for the standard RLS regression algorithm in which several inputs were held out simultaneously. We also described a similar hold-out algorithm for label ranking with RankRLS in the scoring setting (Pahikkala et al. 2007a), that is, by leaving out all inputs associated to the same object simultaneously. Here, we make a more thorough consideration of the label ranking hold-out algorithm for dual RankRLS without tying it to the scoring setting.

Recall that in label ranking tasks, we assume that each input consists of an object and a label and one object is associated to several inputs. However, each input is associated to only one object. Therefore, we assume that there are no preferences between inputs that are associated to different objects. Let $U \subset \{1, \ldots, m\}$ denote the index set that contains the indices of the inputs that are associated to a hold-out object. Leaving more than one object out can be defined analogously. In that case, U would refer to the union of index sets of every hold-out object.

With any matrix (or a column vector) Ψ that has its rows indexed by a superset of U, we use the subscript U so that the matrix Ψ_U contains only the rows that are indexed by U. Similarly, for any matrix Ψ that has its rows indexed by a superset of U and columns indexed by a superset of V, we use Ψ_{UV} to denote a matrix that contains only the rows indexed by U and the columns indexed by V. Moreover, we also denote $\overline{U} = \{1, \ldots, m\} \setminus U$. Further, let $f_{\overline{U}}$ be the hypothesis obtained by training RankRLS without the preferences between the inputs indexed by U. We will frequently make use of the following block matrix multiplication identity:

$$\Psi^{\mathrm{T}} \Upsilon = (\Psi_U)^{\mathrm{T}} \Upsilon_U + (\Psi_{\overline{U}})^{\mathrm{T}} \Upsilon_{\overline{U}},$$

where Ψ and Υ are matrices whose rows are indexed by a superset of U.

Note that in the case of label ranking, we obtain the incidence matrix corresponding to the training data from which the inputs associated to the hold-out object have been removed by just removing the rows indexed by U. After removing the rows, the columns corresponding to the edges incident to the hold-out inputs contain only zeros. This is because both the start and end vertices of those edges are indexed by U. Therefore, the columns have no effect on the values of the matrix multiplications.

Let $Q = M_{\overline{U}}(M_{\overline{U}})^{\mathrm{T}} K_{\overline{U}\overline{U}} + \lambda I_{\overline{U}\overline{U}}$. Then, according to (7) and (17), the predicted scores for the inputs of the hold-out object can be obtained from

$$\begin{aligned} f_{\overline{U}}(X_U) &= K_{U\overline{U}} Q^{-1} M_{\overline{U}} N \\ &= K_{U\overline{U}} Q^{-1} (MN)_{\overline{U}}. \end{aligned} \tag{32}$$

However, having already calculated the solution with the whole training data, the predictions for the hold-out instance can be performed more efficiently than using (32) which calculates Q^{-1}.

Let $R = (MM^{\mathrm{T}} K + \lambda I)^{-1}$. In the case of label ranking, the entries of the matrix $M_{\overline{U}}(M_U)^{\mathrm{T}}$ are zeros for all U, because there are no preferences between the inputs indexed by U and inputs indexed by $\overline{U}$. Therefore, we can write

$$\begin{aligned} Q &= M_{\overline{U}}(M_{\overline{U}})^{\mathrm{T}} K_{\overline{U}\overline{U}} + \lambda I_{\overline{U}\overline{U}} \\ &= M_{\overline{U}}(M_{\overline{U}})^{\mathrm{T}} K_{\overline{U}\overline{U}} + M_{\overline{U}}(M_U)^{\mathrm{T}}(K_{U\overline{U}}) + \lambda I_{\overline{U}\overline{U}} \\ &= M_{\overline{U}}((M_{\overline{U}})^{\mathrm{T}} K_{\overline{U}} + (M_U)^{\mathrm{T}} K_U)(I_{\overline{U}})^{\mathrm{T}} + \lambda I_{\overline{U}\overline{U}} \\ &= M_{\overline{U}} M^{\mathrm{T}} (K_{\overline{U}})^{\mathrm{T}} + \lambda I_{\overline{U}\overline{U}} \\ &= (R^{-1})_{\overline{U}\overline{U}}. \end{aligned}$$

Then, due to the matrix inversion lemma (see e.g. Horn and Johnson 1985),

$$Q^{-1} = R_{\overline{U}\overline{U}} - R_{\overline{U}U}(R_{UU})^{-1} R_{U\overline{U}}.$$

Therefore,

$$\begin{aligned} f_{\overline{U}}(X_U) &= K_{U\overline{U}}(R_{\overline{U}\overline{U}} - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}})(MN)_{\overline{U}} \\ &= K_{U\overline{U}}(R_{\overline{U}\overline{U}}(MN)_{\overline{U}} - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}}(MN)_{\overline{U}}) \\ &= K_{U\overline{U}}(R_{\overline{U}}(I_{\overline{U}})^{\mathrm{T}}M_{\overline{U}}N - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}}(MN)_{\overline{U}}) \\ &= K_{U\overline{U}}(R_{\overline{U}}(M - (I_U)^{\mathrm{T}}M_U)N - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}}(MN)_{\overline{U}}) \\ &= K_{U\overline{U}}(R_{\overline{U}}MN - R_{\overline{U}U}(MN)_U - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}}(MN)_{\overline{U}}) \\ &= K_{U\overline{U}}((RMN)_{\overline{U}} - R_{\overline{U}U}(MN)_U - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}}(MN)_{\overline{U}}). \end{aligned} \tag{33}$$

If (15) has been solved with the whole training data, we already have the matrices R, MN, and RMN stored in the memory, and hence the computational complexity of calculating the matrix inversions, products, and subtractions (in the optimal order) involved in (33) is $O(|\overline{U}||U|+|U|^3) = O(m|U|+|U|^3)$. This is more efficient than the method (32) which calculates the inverse of Q with complexity $O(m^3)$. Assuming that the size of the label sets are of the same size, there is $m/|U|$ objects in the training data, and hence the complexity of calculating a leave-object-out cross-validation is $O((m/|U|)(m|U|+|U|^3)) = O(m^2+m|U|^2)$. This is more efficient than the training of dual RankRLS with the whole training data. Therefore, the cross-validation method can also be combined with the method of selecting the regularization parameter described in Sect. 3.2. We omit the formulas describing this combination, because their presentation would be too long and technical.

3.4 Leave-pair-out cross-validation for object ranking

Here, we consider object ranking in the scoring setting, and hence there is an edge between every input in the preference graph. We consider only the magnitude preserving cost function (21). Therefore, (23) and (25) hold. We now consider leave-pair-out cross-validation (LPOCV) in which every pair of inputs is held out from the training data at a time. If a certain pair of inputs is held out, the edges that are incident to those inputs are not used in the training process in that cross-validation round. In each round, the predictions for the hold-out inputs are calculated. These predictions can then be used for ranking performance measurement. This method is very useful for performance estimation when dealing with so small amounts of data that using a subset of the inputs as a separate test data does not provide reliable enough performance estimate. For a more detailed description of this method, we refer to Pahikkala et al. (2008a).

Cortes et al. (2007a) have proposed an algorithm that approximates the result of LPOCV for the object ranking in $O(m^2)$ time, provided that an inversion of a certain $m \times m$-matrix is already computed and stored in the memory. The larger the number of inputs m is, the closer the approximation to the exact result of the cross-validation is. Here, we improve their result by presenting an algorithm that calculates an exact result of LPOCV in $O(m^2)$ time, again given that the inverse of a certain $m \times m$-matrix is already computed and stored in the memory.

Before presenting the LPOCV algorithm, we consider the following result (for a proof, see e.g. Rifkin and Lippert 2007b; Johnson and Zhang 2008; Pahikkala et al. 2008a).

Lemma 2 *Let $U \subset \{1,\ldots,m\}$ denote the index set that contains the indices of the inputs belonging to the hold-out set and let $\overline{U} = \{1,\ldots,m\} \setminus U$. Further, let $K \in \mathbb{R}^{m\times m}$ be the*

kernel matrix constructed from the inputs X, $f_{\overline{U}}$ be the hypothesis obtained by training RankRLS without using the inputs indexed by U, and $c_{\overline{U}}$ be a cost function that depends only on the predictions made for the inputs indexed by $\overline{U}$. Then, the vector of predictions $f_{\overline{U}}(X)$ can be computed from

$$f_{\overline{U}}(X) = \operatorname*{arginf}_{r \in \mathbb{R}^m} \left\{ c_{\overline{U}}(r, G) + \lambda r^T K^{-1} r \right\}. \tag{34}$$

If K is singular, the term $r^T K^{-1} r$ should be interpreted as

$$\lim_{\epsilon \to 0^+} r^T (K + \epsilon I)^{-1} r.$$

The main insight of the lemma is that we can obtain the hold-out predictions by using a modified cost $c_{\overline{U}}$ that only takes account of the predictions of the inputs not belonging to the hold-out set U. In contrast, the regularizer $\lambda r^{\mathrm{T}} K^{-1} r$ does not have to be modified and it can still depend on all of the m predictions. Note that this property holds for any cost function, and hence the lemma provides us a powerful framework for designing cross-validation algorithms.

Next, we apply Lemma 2 to the cost function (21). Let

$$U = \{h_1, h_2\}$$

be the index set containing the indices h_1 and h_2 of the two hold-out inputs and let $\overline{U} = \{1, \ldots, m\} \setminus U$. Further, let $S = (s_1, \ldots, s_m)^{\mathrm{T}} \in \mathbb{R}^m$ be a vector of real valued scores of the inputs. We now reformulate the matrix form (24) so that its value is independent of the predictions for the hold-out inputs. Recall that the preference magnitudes in the scoring setting can be expressed with the differences of the scores of the inputs and that we also include the preferences with zero magnitudes in training for efficiency reasons. Therefore, the cost function (21) which is calculated over the whole training data can be expressed as

$$c(r, G) = \frac{1}{2} \sum_{i,j=1}^{m} ((s_i - s_j) - (r_i - r_j))^2.$$

The sum is multiplied with the constant $\frac{1}{2}$, because the sum contains each index pair twice, since in this setting $((s_i - s_j) - (r_i - r_j))^2 = ((s_j - s_i) - (r_j - r_i))^2$ and $((s_i - s_i) - (r_i - r_i))^2 = 0$ for all $i, j \in \{1, \ldots, m\}$. In order to make the cost function independent of the predictions for the hold-out inputs, we remove the terms involving the hold-out inputs from the sum. Thus, the cost function from which the terms have been removed is

$$\begin{aligned} c_{\overline{U}}(r, G) &= \frac{1}{2} \sum_{i,j \in \overline{U}} ((s_i - s_j) - (r_i - r_j))^2 \\ &= (m-2) \sum_{i \in \overline{U}} (s_i - r_i)^2 - \sum_{i,j \in \overline{U}} (s_i - r_i)(s_j - r_j) \\ &= (S - r)^{\mathrm{T}} \widetilde{L} (S - r), \end{aligned} \tag{35}$$

where $\widetilde{L} \in \mathbb{R}^{m \times m}$ is a matrix whose entries are defined as

$$\widetilde{L}_{i,j} = \begin{cases} -1 & \text{if } i \neq j \text{ and } i, j \in \overline{U}, \\ m-3 & \text{if } i = j \text{ and } i \in \overline{U}, \\ 0 & \text{otherwise.} \end{cases}$$

The matrix form (35) is similar to (24) except that the Laplacian matrix $\widetilde{L}$ corresponds to a graph from which all edges incident to the hold-out inputs have been removed.

Next, we substitute (35) into (34). Then, derivating (34) with respect to r and setting it to zero provides us the predictions for all of the m inputs made by $f_{\overline{U}}$:

$$f_{\overline{U}}(X) = (\widetilde{L} + \lambda K^{-1})^{-1} \widetilde{L} S.$$

Now, multiplying with I_U from the left provides us the predictions for the hold-out inputs

$$f_{\overline{U}}(X_U) = I_U (\widetilde{L} + \lambda K^{-1})^{-1} \widetilde{L} S. \tag{36}$$

We continue by observing that we can also write

$$\widetilde{L} = \widetilde{D} - B B^{\mathrm{T}}, \tag{37}$$

where $B \in \mathbb{R}^{m \times 3}$ is a matrix whose values are determined by

$$B_{i,j} = \begin{cases} 1 & \text{if } i \in \overline{U} \text{ and } j = 1, \\ \sqrt{m-2} & \text{if } i = h_1 \text{ and } j = 2, \\ \sqrt{m-2} & \text{if } i = h_2 \text{ and } j = 3, \\ 0 & \text{otherwise} \end{cases}$$

and

$$\widetilde{D} = (m-2) I \in \mathbb{R}^{m \times m}.$$

Let

$$Q = (\widetilde{D} + \lambda K^{-1})^{-1}.$$

Using (37), we can rewrite (36) as

$$\begin{aligned} f_{\overline{U}}(X_U) &= I_U (Q^{-1} - B B^{\mathrm{T}})^{-1} \widetilde{L} S \\ &= I_U (Q - Q B (-I + B^{\mathrm{T}} Q B)^{-1} B^{\mathrm{T}} Q) \widetilde{L} S \\ &= (Q \widetilde{L} S)_U - (Q B)_U (-I + B^{\mathrm{T}} Q B)^{-1} B^{\mathrm{T}} Q \widetilde{L} S, \end{aligned} \tag{38}$$

where the second equality is due to the Sherman-Morrison-Woodbury formula. Let $C \in \mathbb{R}^{m \times 3}$ be a matrix whose values are determined by

$$C_{i,j} = \begin{cases} 1 & \text{if } j = 1, \\ 0 & \text{otherwise} \end{cases}$$

and let

$$R = \begin{pmatrix} -1 & \sqrt{m-2} & 0 \\ -1 & 0 & \sqrt{m-2} \end{pmatrix}.$$

We observe that we can write

$$(I_U)^{\mathrm{T}} R = B - C,$$

where I is an $m \times m$-identity matrix, and hence

$$R = B_U - C_U.$$

Let us assume that we have calculated the matrices

$$Q, \widetilde{D}S, Q\widetilde{D}S, QC, C^{\mathrm{T}}QC, C^{\mathrm{T}}S, QCC^{\mathrm{T}}S, \text{ and } C^{\mathrm{T}}Q\widetilde{D}S, \tag{39}$$

and stored them into the memory before starting the hold-out calculations. In order to calculate (38), we have to compute the following matrices:

$$\begin{aligned} B^{\mathrm{T}}QB &\in \mathbb{R}^{3\times 3}, \\ B^{\mathrm{T}}Q\widetilde{L}S &\in \mathbb{R}^{3\times 1}, \\ (QB)_U &\in \mathbb{R}^{2\times 3}, \\ (Q\widetilde{L}S)_U &\in \mathbb{R}^{2\times 1}. \end{aligned}$$

Given that the matrices (39) have already been calculated, the above matrices can be calculated in a constant time as follows:

$$\begin{aligned} B^{\mathrm{T}}QB &= (C + (I_U)^{\mathrm{T}}R)^{\mathrm{T}}Q(C + (I_U)^{\mathrm{T}}R) \\ &= C^{\mathrm{T}}QC + R^{\mathrm{T}}I_U QC + C^{\mathrm{T}}Q(I_U)^{\mathrm{T}}R + R^{\mathrm{T}}I_U Q(I_U)^{\mathrm{T}}R \\ &= C^{\mathrm{T}}QC + R^{\mathrm{T}}(QC)_U + (R^{\mathrm{T}}(QC)_U)^{\mathrm{T}} + R^{\mathrm{T}}Q_{UU}R, \\ B^{\mathrm{T}}Q\widetilde{L}S &= B^{\mathrm{T}}Q\widetilde{D}S - B^{\mathrm{T}}QBB^{\mathrm{T}}S \\ &= C^{\mathrm{T}}Q\widetilde{D}S + R^{\mathrm{T}}(Q\widetilde{D}S)_U - B^{\mathrm{T}}QB(C^{\mathrm{T}}S + R^{\mathrm{T}}S_U), \\ (QB)_U &= (QC)_U + (Q(I_U)^{\mathrm{T}}R)_U \\ &= (QC)_U + Q_{UU}R, \\ (Q\widetilde{L}S)_U &= (Q\widetilde{D}S)_U - (QBB^{\mathrm{T}}S)_U \\ &= (Q\widetilde{D}S)_U - (QB)_U B^{\mathrm{T}}S \\ &= (Q\widetilde{D}S)_U - ((QC)_U + Q_{UU}R)(C^{\mathrm{T}}S + R^{\mathrm{T}}S_U). \end{aligned}$$

By substituting these into (38), the hold-out predictions for a pair of inputs can be calculated in a constant time.

Concerning the matrices (39) calculated in advance, the calculation of the matrix Q is the computationally dominant one. Namely, its time complexity is $O(m^3)$ in the worst case of K being of full rank. This is the same as that of training the RankRLS algorithm in the worst case. However, if the rank of K is not full, the matrix Q can be calculated as follows.

Let $K = V\Lambda V^{\mathrm{T}}$ be the eigen decomposition of the kernel matrix, where V contains the eigenvectors of K and Λ is a diagonal matrix containing the eigenvalues of K. Then,

$$Q = V\widehat{\Lambda}V^{\mathrm{T}},$$

where $\widehat{\Lambda}$ is the diagonal matrix whose elements are determined by

$$\widehat{\Lambda}_{i,i} = \frac{\Lambda_{i,i}}{\lambda + (m-2)\Lambda_{i,i}}.$$

The calculation of the other matrices in (39) need only $O(m^2)$ time if Q is already calculated.

After the matrices (39) are calculated, the overall complexity of LPOCV is $O(m^2)$, since only a constant time is needed to compute (38) for each hold-out set U and there are $O(m^2)$ different hold-out sets. This is advantageous, for example, if we have many independent ranking tasks we aim to learn from the same input data. In this case, the outputs are stored, instead of a single column matrix, in a matrix $S \in \mathbb{R}^{m\times v}$, where v is the number of tasks. Then, the time complexity of the cross-validation is $O(m^3 + m^2 v)$, since the complexity of calculating Q is not affected by the number of tasks.

3.5 Sparse approximation

If the number of inputs m is large, the time complexities (19) or (26) of training dual RankRLS may become infeasible and approximative techniques are needed. In this section, we propose an approach that is based on a similar kind of idea as the subset of regressors method (see e.g. Poggio and Girosi 1990; Smola and Schölkopf 2000; Rifkin et al. 2003; Quiñonero-Candela et al. 2007) for the standard regularized least-squares regression. More detailed considerations and experimental results of this approach for RankRLS are presented in Tsivtsivadze et al. (2008).

Recall that the training data contains a sequence of inputs $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$ of length m and let $R \subseteq \{1, \ldots, m\}$, where $|R| \ll m$. The inputs indexed by the set R are called the basis vectors. Now we consider instead of (6) a solution that allows only the inputs indexed by R to have nonzero coefficient, that is,

$$f(x) = \sum_{i\in R} a_i k(x, x_i). \tag{40}$$

Note that it is not guaranteed that the optimal solution with only $|R|$ nonzero coefficients a_i will have a representation as in formula (40), because the sparse approximation cannot straightforwardly resort to the representer theorem anymore. Clearly, the selection of the index set R may have an influence on results obtained by our method. Different approaches for selecting R are discussed, for example, in Rifkin et al. (2003). There, it was found that simply selecting the elements of R randomly performs no worse than more sophisticated methods.

The problem of finding this type of hypothesis can be solved by finding the coefficients a_i, where $i \in R$. In this case, the predictions for the training inputs can be expressed as $f(X) = (K_R)^{\mathrm{T}}A$ and the regularizer as $\lambda A^{\mathrm{T}} K_{RR} A$, where $A \in \mathbb{R}^{|R|}$ is a vector consisting of the coefficients a_i. Using these definitions, we present a method we call sparse RankRLS:

$$\mathcal{A}(G) = \operatorname*{argmin}_{A\in\mathbb{R}^{|R|}} J(A, G),$$

where

$$J(A, G) = (N - M^{\mathrm{T}}(K_R)^{\mathrm{T}}A)^{\mathrm{T}}(N - M^{\mathrm{T}}(K_R)^{\mathrm{T}}A) + \lambda A^{\mathrm{T}}K_{RR}A.$$

We take the derivative of $J(A, G)$ with respect to A, set it to zero, and solve with respect to A:

$$A = (K_R M M^{\mathrm{T}}(K_R)^{\mathrm{T}} + \lambda K_{RR})^{-1} K_R M N. \tag{41}$$

The computational complexity of calculating (41) can be analyzed in a similar way as that of the primal RankRLS (28), because the former contains the matrix K_R in place of H and K_{RR} in place of I, that is, we substitute $|R|$ in place of n in (29). However, the solution can be found more efficiently if we assume the scoring setting and that the magnitude preserving cost function (21) is used making (23) and (25) to hold. Then, the training complexity of the sparse RankRLS algorithm can be analyzed by substituting $|R|$ in place of n in (30) resulting in $O(m|R|^2)$ complexity, since $|R| \ll m$. Thus, the size of R can be selected so that these computation times are feasible.

The efficient selection of regularization parameter discussed in Sect. 3.2 can also be performed with sparse RankRLS using the following method. Here, we again assume that we use the cost function (21) in the scoring setting, and hence (23) and (25) hold. Using the Cholesky decomposition $K_{RR} = ZZ^{\mathrm{T}}$, where $Z \in \mathbb{R}^{|R|\times|R|}$ is a lower triangular matrix called the Cholesky triangle of K_{RR}, we can rewrite the solution (41) as follows:

$$A = (K_R M M^{\mathrm{T}}(K_R)^{\mathrm{T}} + \lambda Z Z^{\mathrm{T}})^{-1} K_R M M^{\mathrm{T}} S.$$

Note that since we assume the kernel matrix K to be strictly positive definite, it follows that also its principal submatrix K_{RR} is strictly positive definite, and hence Z is invertible. Let

$$V \Lambda V^{\mathrm{T}} = Z^{-1} K_R M M^{\mathrm{T}}(K_R)^{\mathrm{T}}(Z^{-1})^{\mathrm{T}} \tag{42}$$

be the eigen decomposition of $Z^{-1}K_R M M^{\mathrm{T}}(K_R)^{\mathrm{T}}(Z^{-1})^{\mathrm{T}}$, where V and Λ are the eigenvector matrix and diagonal matrix containing the corresponding eigenvalues, respectively. Further, let $\hat{\Lambda}_\lambda = (\Lambda + \lambda I)^{-1}$. Then,

$$\begin{aligned}(K_R M M^{\mathrm{T}}(K_R)^{\mathrm{T}} + \lambda Z Z^{\mathrm{T}})^{-1} &= (Z^{-1})^{\mathrm{T}}(V \Lambda V^{\mathrm{T}} + \lambda I)^{-1} Z^{-1} \\ &= (Z^{-1})^{\mathrm{T}} V \hat{\Lambda}_\lambda V^{\mathrm{T}} Z^{-1}.\end{aligned}$$

Therefore, we rewrite the solution (41) as follows:

$$A = (Z^{-1})^{\mathrm{T}} V \hat{\Lambda}_\lambda V^{\mathrm{T}} Z^{-1} K_R M M^{\mathrm{T}} S.$$

The decompositions and the inversion of Z can be calculated in $O(|R|^3)$ time, and hence the overall training complexity is not increased. The computational cost of calculating $\hat{\Lambda}_\lambda$ is $O(|R|)$, since $\Lambda + \lambda I$ is a diagonal matrix. When the matrices $V^{\mathrm{T}}Z^{-1}K_R M M^{\mathrm{T}} S \in \mathbb{R}^{|R|\times 1}$ and $(Z^{-1})^{\mathrm{T}}V \in \mathbb{R}^{|R|\times|R|}$ are stored in the memory, the subsequent training with different values of regularization parameters can be performed in $O(|R|^2)$ time.

We also note that the sparsity of the learned solution speeds up the prediction when nonlinear kernels are used. Namely, the prediction complexity scales with respect to $|R|$ times the complexity of calculating the kernel function.

4 Summary of computational benefits and comparison to RankSVM

Here, we make a summary about the computational properties of RankRLS and compare them to those of RankSVM. RankSVM (Herbrich et al. 1999; Joachims 2002) is a state-of-the-art ranking method very closely related to RankRLS. Their objective functions are the same except that RankSVM uses the hinge cost function:

$$c(f(X), G) = \sum_{i=1}^{l} \max(1 - g(e_i)\text{sign}(y_i), 0),$$

where $e_i = (x_h, x_j, y_i)$ are the observed preferences, $g(e_i) = f(x_h) - f(x_j)$, and $\text{sign}(y_i)$ are the directions of the preferences e_i. As for RankRLS, there are also various different methods for finding a minimizer of the objective function. It can be argued that the hinge cost is a better approximation of the disagreement error than the squared costs as it does not penalize correct predictions with magnitudes larger than one. However, in our experiments, we observe that the ranking performance of RankRLS is essentially the same as that of RankSVM. A similar phenomenon has also been observed between support vector machine and regularized least-squares classifiers (see e.g. Rifkin 2002; Gestel et al. 2004; Zhang and Peng 2004). Thus, the computational issues become the main factor for deciding whether RankSVM or RankRLS is preferable.

Next, we investigate in which circumstances it is more beneficial to use RankSVM or RankRLS method from the computational complexity point of view. For simplicity, we make the investigation only in the case the preferences are induced by a scoring of the inputs. Further, we only consider the cost function (21), and hence (23) and (25) hold.

Recently, Joachims (2006) proposed an efficient linear support vector machine type of ranking algorithm for scored data. The training complexity of the algorithm is $O(\bar{n}m \log m)$, where $\bar{n}$ is the average number of nonzero features in the inputs. In our comparisons, we consider this algorithm in the linear case. Further, we investigate the possibilities to use this algorithm also in the nonlinear case.

We have divided our consideration into four cases. We start with small-scale learning using linear kernel in Sect. 4.1 and continue with nonlinear case in Sect. 4.2. With small-scale learning we refer to the case in which the number m of inputs in the training data is such that $O(m^3)$ time complexity can be afforded, and with large-scale learning we refer to the opposite case. Large-scale learning with linear and nonlinear kernels are considered in Sects. 4.3 and 4.4, respectively.

4.1 Linear small scale learning

Because of the efficient training algorithm in the linear case, RankSVM has a computational advantage over RankRLS when only one instance of the ranking method is needed and no parameter selection or performance evaluation with cross-validation are performed. However, the advantage becomes less clear when there is a need for selecting the value of the regularization parameter, learning multiple outputs, or performing cross-validation. For example, the ability to perform cross-validation efficiently is very important when the number of inputs with known scores is small, since in many cases large enough test sets for reliable performance estimation can not be afforded. Next, we present some examples in which these properties are especially beneficial.

Small-size data appears frequently, for example, when solving medical and biological tasks, and hence cross-validation is often the only reliable way to measure the ranking

performance. In this case, the efficient methods presented in Sects. 3.3 and 3.4 are very useful. For example, when aiming for a maximal AUC with biological data as considered by Parker et al. (2007), a common practice for performance evaluation is to use a ten-fold cross-validation. Then, the overall AUC is obtained by computing AUC for each fold and taking their average or by first pooling the predictions and computing AUC afterwards. Taking the average suffers from large variance, because the number of input pairs in each fold may be too small. Moreover, Parker et al. (2007) reported that the pooling technique suffers from a pessimistic bias. The efficient leave-pair-out cross-validation provides a third way for AUC calculation that avoids many of the pitfalls associated to the pooling and averaging techniques.

Another advantage of RankRLS in linear small-scale learning is its ability to learn multiple outputs at the cost of only one, provided that the number of outputs is linear in m or in n. For example, in our experiments with the Reuters data in Sect. 5.4, there are 25 outputs that can be learned in parallel. Of course, learning multiple outputs is also very efficient with the fast RankSVM training methods. However, the fast cross-validation algorithms of RankRLS can be combined with multiple output learning. This makes it possible, for example, to perform permutation tests similar to those used for classification (see e.g. Golland et al. 2005). In the permutation tests, the outputs or scores of the training data are shuffled randomly and the learner is then trained and cross-validated with the data having permuted outputs. The shuffling and training is repeated many times and the cross-validation results are used, for example, to estimate the reliability of the cross-validation results with the original training data. This method can be used very efficiently with the RLS-based learning algorithms, because the permuted output vectors can be considered as extra outputs that can be learned and cross-validated in parallel.

4.2 Nonlinear small scale learning

In general, when nonlinear kernel functions are used, support vector machine (SVM) type of learners have an advantage in prediction time, because the form of the SVM solution may be sparse. However, this depends on the level of regularization and the amount of noise in the training data.

RankRLS has the advantage that its training time scales linearly instead of quadratically with respect to the number m of inputs in the training data. To our knowledge, at least the most commonly used implementations of nonlinear RankSVM scale roughly quadratically with respect to the number of preferences, and hence their computational complexity can be considered to be at least of the order $O(m^4)$, because the number of preferences in the scoring setting is assumed to be of order m^2. This makes RankSVM impractical even on small datasets. The cubic complexity of RankRLS makes it thus the method of choice when using nonlinear kernels and datasets which consist of at most a few thousand inputs. However, the dual implementations of the RankSVM do not necessarily represent the state-of-the-art in kernel based SVM ranking.

To demonstrate the scalability properties of nonlinear RankRLS and RankSVM algorithms, we provide a comparison of running times on the acq dataset of the Reuters AUC-maximization task (see Sect. 5.4 for description of the task and data). The RankSVM implementation is the one included in the SVM-light package, for RankRLS we use our own Python implementation. The runs are performed on a modern desktop computer using the Gaussian kernel and the default parameter values of the software packages are used. The results are presented in Table 1.

The runtime comparison of training provides further empirical support to the conclusions derived from the computational complexity considerations. RankRLS is efficient to use in

Table 1 Runtime comparisons of training for nonlinear RankRLS and RankSVM on the acq dataset. The number of inputs in the training data ranges from 200 to 6000, the runtimes are measured in seconds

	Running times								
Inputs	200	500	750	1000	1500	2000	2500	4000	6000
RankRLS	1	3	5	10	24	48	83	280	841
RankSVM	2	150	579	1740	4685	13707	20055	–	–

the small-scale setting where the number of inputs in the training data is measured in thousands. RankSVM however does not scale well, for example, at a point of 2500 inputs where RankRLS training takes less than one and a half minutes, training RankSVM takes five and a half hours. Taking further into consideration the efficient regularization, cross-validation and multiple output learning algorithms presented for RankRLS, it is clearly the better choice in this setting.

Next, we consider an alternative approach using the empirical kernel map (Schölkopf et al. 1999) to transform the SVM dual problem into a primal one, and hence to achieve cubic complexity also for the RankSVM. Formally, if a full rank kernel matrix K is decomposed, for example, with the Cholesky decomposition

$$K = ZZ^{\mathrm{T}},$$

where the Cholesky triangle $Z \in \mathbb{R}^{m \times m}$ of K can be considered as an empirical feature space representation of the input sequence X. It can be shown that after a linear RankSVM is trained with these features, the dual variables needed in prediction for new inputs can be obtained by multiplying the normal vector of the learned separating hyperplane with the inverse of Z^{T}. The computational complexity of the Cholesky decomposition of K is $O(m^3)$. After the decomposition is performed, the training of RankSVM with this feature representation is of complexity $O(m^2 \log m)$, because the average number of nonzero features per input in this case is m. When testing in practice this approach for training a single RankSVM learner, we observed training times that were very close to that of training a single RankRLS learner. This is because the $O(m^3)$ complexities of the eigen decomposition used in training RankRLS and the Cholesky decomposition in training RankSVM dominate the running times.

On the one hand, the Cholesky decomposition has to be performed only once, since the same feature space representation can be used for multiple outputs, multiple values of the regularization parameter, and in each cross-validation round. On the other hand, the $O(m^2 \log m)$ time is spent for every combination of the regularization parameter, every separate output, and every round in a cross-validation. Compared to that, RankRLS spends $O(m^2)$ time for every combination of the regularization parameter and output. However, the fast cross-validation properties of RankRLS make it more suitable than RankSVM for small-scale nonlinear ranking tasks. For example, the constant time hold-out computation introduced in Sect. 3.4 means that the eigen decomposition still dominates the RankRLS running time in leave-pair-out cross-validation but the complexity of RankSVM would rise to $O(m^4 \log m)$, since there are m^2 cross-validation rounds.

4.3 Linear large scale learning

Recall from (30) that the computational complexity of training the primal RankRLS is $O(n^3 + n^2 m)$, where n is the dimensionality of the feature space. Further, after RankRLS is

trained once, the level of regularization can be adjusted and multiple tasks learned efficiently as shown in Sect. 3.2. If n is a small constant and m is large enough, these properties make RankRLS faster to train than RankSVM that has the $O(\bar{n}m \log m)$, where $\bar{n}$ is the average number of nonzero features per input, training complexity.

If both the number of inputs in the training data m and the number of features n are large, the cubic time complexities of training RankRLS with the matrix calculus based techniques become infeasible. However, it may still be possible to take advantage of the sparsity of the feature representation of the inputs, that is, $\bar{n}$ being small. We note that, similarly to the standard RLS regression (see e.g. Rifkin et al. 2003; Shewchuk 1994), RankRLS can also be trained in such circumstances using conjugate gradient type of algorithms where the complexity of each iteration is $O(\bar{n}m)$. How close the coefficient vector obtained with this method is to the minimizer of (16) depends of the number of iterations. Since we assume the use of the cost function (21) in the scoring setting, we can write $MM^{\mathrm{T}} = D - PP^{\mathrm{T}}$, where D and P are defined as in the beginning of Sect. 3. Moreover, recall that in both the object and label ranking cases, the matrices P and D have only m nonzero entries. Further, let $H \in \mathbb{R}^{n \times m}$ be the sparse matrix containing the feature vectors of the training inputs having an average of $\bar{n}$ nonzero features per input and let $v \in \mathbb{R}^m$ be a vector. Then, we can compute the product

$$(KMM^{\mathrm{T}}K + \lambda K)v = H^{\mathrm{T}}HDH^{\mathrm{T}}Hv - H^{\mathrm{T}}HPP^{\mathrm{T}}H^{\mathrm{T}}Hv + \lambda H^{\mathrm{T}}Hv$$

in $O(\bar{n}m)$ time, since H contains approximately $\bar{n}m$ nonzero elements, and both D and P contain only m nonzero elements. Computing this product is the most expensive operation in each conjugate gradient iteration.

We run a test of the conjugate gradient algorithm using the Reuters classification task and linear kernel (see Sect. 5.4) using more than 12000 inputs and features, which generate over 23 million pairwise preferences. The algorithm needs only a couple of hundred iterations to converge, and hence the training takes only a few seconds.

4.4 Nonlinear large scale learning

The cubic complexity of nonlinear RankRLS is impractical in large-scale learning. However, it is possible to use sparse approximations as discussed in Sect. 3.5 having $O(m|R|^2)$ training complexity, where R is the set consisting of the indices of the basis vectors and $|R| \ll m$. This type of approximations are also possible for SVM type of learners as outlined in the following. Similarly to the empirical kernel map approach described in Sect. 4.2, the training tasks can again be transformed in $O(m|R|^2)$ time into a more efficient linear learning task, where the dimension of the feature space is $|R|$. Formally, the use of the sparse approximation corresponds to the use of the following type of modified kernel function (see e.g. Quiñonero-Candela and Rasmussen 2005):

$$\tilde{k}(x, x') = k(x, X_R)(K_{RR})^{-1}k(x', X_R)^{\mathrm{T}}, \tag{43}$$

where X_R is a sequence of basis vectors and $k(x, X_R) \in (\mathbb{R}^{|R|})^{\mathrm{T}}$ is a row vector consisting of the kernel evaluations between the input x and the training inputs indexed by R. Therefore, the kernel matrix corresponding to the modified kernel function $\tilde{k}$ can be written as

$$\widetilde{K} = (K_R)^{\mathrm{T}}(K_{RR})^{-1}K_R.$$

Now, if $(K_{RR})^{-1} = ZZ^{\mathrm{T}}$ is the Cholesky decomposition of $(K_{RR})^{-1}$, we can use $(K_R)^{\mathrm{T}}Z \in \mathbb{R}^{m \times |R|}$ as an empirical kernel map with which linear RankSVM can be trained. It can be

shown that after a linear RankSVM is trained using the feature representation obtained from this empirical kernel map, the vector of $|R|$ dual variables needed in making predictions for new inputs with the original kernel function k can be calculated by multiplying the normal vector of the learned separating hyperplane with Z.

After the feature representation based on the empirical kernel map has been constructed in $O(m|R|^2)$ time, the complexity of training a RankSVM is $O(|R|m \log m)$ for a single output and for a single value of the regularization parameter, since the number of dimensions in the feature representation determined by the empirical kernel map is $|R|$ and the representation is usually dense. Compared to that, after the eigen decomposition (42) and the other matrix operations needed in training a sparse RankRLS for a single output and a single value of the regularization parameter have been performed in $O(m|R|^2)$ time, subsequent training with different values of the regularization parameter for the same output is even more efficient, namely $O(|R|^2)$ per each parameter value.

5 Experiments

We test our ranking algorithms with various different tasks. The tasks considered are: ranking of dependency parses, document retrieval, binary document classification, and collaborative filtering. The two first tasks are instances of label ranking while the other two can be considered as object ranking problems. The pairwise preferences in all of the four tasks are induced by a scoring of the inputs. For example, in the document retrieval task the score of an input consisting of a query and a document is 1 if the document is relevant to the query and 0 otherwise. The document retrieval and binary classification tasks can be considered as bipartite ranking problems, that is, there are only two possible score values for the inputs. On the other hand, the true scores of the inputs in the parse ranking and collaborative filtering tasks are real numbers between a certain interval.

In all tasks, we test whether the irrelevant input pairs would be beneficial if included in the training process. For example, in document retrieval we do not measure the disagreement error between the inputs that are associated to different queries, but test if they are still useful in training.

We also compare the RankRLS algorithm with RankSVM and standard RLS regressor in all of the four tasks. The RankSVM baseline is always trained with only the relevant pairs, since the irrelevant pairs were found to be non-beneficial in our experiments with RankRLS. Moreover, in the document retrieval experiments, RankBoost is used as additional baseline. We also compare the cost functions (20), (21), and (22) with each other on the non-bipartite ranking tasks, since they are equal in bipartite tasks.

Both RankRLS and RLS regressor have a regularization parameter λ that controls the trade-off between the minimization of the training error and the complexity of the function. RankSVM has a similar parameter. The parameters are selected using cross-validation from the scale $[2^{-15}, 2^{-14}, \ldots, 2^{-14}, 2^{15}]$. Further, the used kernel functions have parameters that are set by cross-validation on the training data. Whenever a statistical significance is reported, the Wilcoxon signed-ranks test (Wilcoxon 1945) has been used with 0.05 as a significance threshold.

In Sect. 5.1, a simple example is presented in which we consider the effect of having irrelevant input pairs in the training process. Section 5.2 presents our experiments with parse ranking and Sect. 5.3 with document retrieval. We consider maximizing the area under ROC curve in Sect. 5.4 and collaborative filtering in Sect. 5.5.

5.1 The case of irrelevant input pairs

To investigate the possible effects of the irrelevant input pairs in the training process, we now present an artificial label ranking example that is illustrated in Fig. 2. In both figures, the feature vectors $\Phi(x_i)$ of the four inputs x_i are depicted as circles and they reside on a one-dimensional feature space. We assume that there are two objects, and inputs denoted x_1 and x_2 are associated to the first object and x_3 and x_4 to the second one. Therefore, only two pairs of inputs are relevant to the label ranking task, namely the pairs (x_1, x_2) and (x_3, x_4).

The four inputs are given scores that are $s_1 = 2$, $s_2 = 1$, $s_3 = 4$, and $s_4 = 3$. The scores induce a direction of preference for the two relevant input pairs. These preferences are depicted with arrows between the inputs in Fig. 2 (left). The scores also induce preferences for the four input pairs that are not relevant to the label ranking task in question. Both the relevant and irrelevant preferences are depicted in Fig. 2 (right). We observe that the preference direction of the relevant edges goes from left to right in the feature space but the direction of the irrelevant edges is opposite.

Since the feature vectors in the example are one-dimensional and we are learning only linear scoring functions, there are only two possible ways in which the inputs can be ordered. Namely, from left to right or in the opposite way. In Fig. 2, the direction is determined by the normal vector of the hyperplane corresponding to the RankRLS solution. Therefore, if the irrelevant input pairs are excluded from the training process, RankRLS learns the first type of ordering and it can correctly predict the preferences for both of the relevant inputs pairs. However, if the four irrelevant pairs are included in training, they overwhelm the effect of the two relevant pairs, and hence RankRLS learns the wrong type of ordering. This type of phenomenon may occur frequently in the label ranking task, since the inputs associated to the same object are often clustered in a similar way as in this example. Therefore, we speculate that the irrelevant pairs are usually more harmful than useful if included in the training of RankRLS for label ranking tasks.

Another type of input pair that may turn out to have an effect to the ranking performance is a tied one, that is, a pair whose both inputs have the same score. The tied pairs of inputs are not considered in our definition of the disagreement error. However, in our experiments we test whether it has a beneficial or harmful effect on the ranking performance if the tied pairs are used in the training. For simplicity, we perform the test only with (21), because it is the only cost function in which the ties can be treated in a trivial way, that is, by setting the zero point of the parabola to be zero.

We may also consider leaving out some of the relevant input pairs in case there is some redundancy created by the transitivity of the preferences, for example, in the scoring setting. However, this may not be an optimal strategy if the data is too noisy.

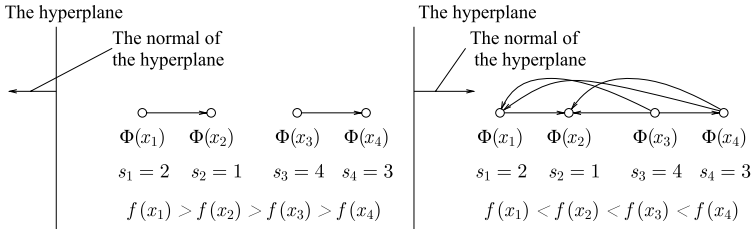

Fig. 2 Artificial label ranking example. Only the relevant input pairs are included in the training process (*left*). Both the relevant and irrelevant input pairs are included in the training process (*right*)

5.2 Ranking of dependency parses

First, we give a short description of the characteristics of the data. For a more detailed description, we refer to Tsivtsivadze et al. (2005). Next, we describe the task of parse ranking. Finally, we present the experimental evaluation.

Throughout our experiments, we use the BioInfer corpus (Pyysalo et al. 2007) which consists of 1100 manually annotated sentences.[1] For each sentence, we generate a set of candidate parses with a link grammar (LG) parser (Sleator and Temperley 1991). The LG parser is a full dependency parser based on a broad-coverage hand-written grammar. It generates all parses allowed by its grammar and applies a set of built-in heuristics to rank the parses. However, the ranking performance of the heuristics has been found to be poor when applied to biomedical text (Pyysalo et al. 2006), and hence subsequent ranking or selection methods are needed. In our previous studies, we used regularized least-squares regression for the reranking task that notably outperformed the LG heuristics (Tsivtsivadze et al. 2005). In these experiments, we use the graph kernel described in Pahikkala et al. (2006b).

In the task of parse ranking, each input consists of a sentence and a parse generated for it. We obtain a scoring for an input by comparing its parse to the hand annotated correct parse of its sentence. Tsivtsivadze et al. (2005) describes in detail how the scores are calculated. The relevant input pairs are those of which both inputs are associated to the same sentence and have different scores. All the other pairs are considered to be irrelevant to the task of parse ranking. We evaluate whether these irrelevant input pairs are beneficial if included in the training process. Furthermore, we compare the performance of RankRLS with the cost functions (20), (21), and (22).

In order to select the parameter values, we divide the set 1100 annotated sentences into two data sets containing 500 and 600 sentences. The first dataset is used for the parameter estimation and the second one is reserved for the final validation. The appropriate values of the regularization and the kernel parameters are determined by grid search with 10-fold cross-validation on the parameter estimation data.

Finally, the algorithm is trained on the whole parameter estimation data set with the selected parameter values and tested with the 600 sentences reserved for the final validation. The results of the validation are presented in Table 2. We observe that the regression approach is clearly worse than RankRLS. The performance differences between RLS regressor and RankRLS in the relevant pairs case are all statistically significant. Moreover, the performance differences of the results obtained by RankRLS methods when trained using

Table 2 Disagreement errors for the validation set using different methods. RankRLS is tested with the cost functions (20), (21), and (22), and both with only relevant pairs and all pairs

Method	Disagreement error
RankRLS (20)	0.225
RankRLS (20) All pairs	0.234
RankRLS (21)	0.222
RankRLS (21) All pairs	0.247
RankRLS (22)	0.216
RankRLS (22) All pairs	0.228
RankSVM	0.214
RLS Regressor	0.252

[1] Available at www.it.utu.fi/BioInfer.

relevant and all pairs are statistically significant indicating that the irrelevant pairs are harmful. Interestingly, the three cost functions seem to differ more in the all pairs experiments, while the results were clearly worse than with the relevant pairs only. When considering the relevant pairs, the results of RankSVM are very close to those of RankRLS.

5.3 Learning to rank for information retrieval

In this section, we present an evaluation of the RankRLS algorithm on the task of ranking documents according to queries using the publicly available Letor information retrieval dataset (Liu et al. 2007).[2] The problem is an example of a typical label ranking task. Given a set of query-document pairs, our aim is to learn to rank all the documents related to the same query according to how well they match the query. We also test how the inclusion of irrelevant preferences in the training data affects the performance of RankRLS. By irrelevant preferences, we mean in this setting pairs of inputs related to different queries or input pairs that are related to the same query, but have the same score associated with them. In addition to RankSVM and the standard RLS regressor comparisons, we also compare our results to those of RankBoost (Freund et al. 2003). Further details of these experiments can be found in Pahikkala et al. (2007b).

Recently, the Letor dataset for learning to rank in information retrieval containing three datasets of query-document pairs known as Ohsumed (16140 pairs), Trec2003 (49171 pairs) and Trec2004 (74170 pairs), as well as baseline results on RankBoost and RankSVM algorithms, has been made available. The Trec datasets contain only two possible scores for the inputs 0 and 1, while Ohsumed has three possible scores, 0, 1 and 2. In these experiments, we consider Ohsumed to be a bipartite ranking task by combining the scores 0 and 1 together.

We perform experiments on all of the three datasets. Because of the small dimensionality of the feature space (25 features in Ohsumed, 44 in Trecs) coupled with the large dataset sizes, we use the primal version of RankRLS which scales well in such settings as discussed in Sect. 3.1. Because of this choice, we use the linear kernel. The data is preprocessed by normalizing all of the feature values between 0 and 1 on per query basis. 5-fold cross-validation is used so that in each phase the learners are trained with three folds, parameters chosen on a fourth one and testing is done on the remaining fold. The fold split used is the one provided in the dataset. All results are averaged over the folds. We evaluate the results using disagreement error averaged over the different test queries. Such queries that are related only to documents that have score 1, or only to documents that have score 0, and thus contain no preferences, are not considered in the performance evaluation. The experimental results are presented in Table 3.

Table 3 Disagreement errors on the Letor datasets. RankRLS is tested in two settings: only relevant pairs are included and all pairs are included. Standard RLS regression, RankSVM and RankBoost are used as baselines

Method	Ohsumed	Trec2003	Trec2004
RankRLS	0.340	0.145	0.034
RankRLS All pairs	0.346	0.141	0.048
RLS Regressor	0.346	0.153	0.044
RankSVM	0.336	0.150	0.041
RankBoost	0.351	0.138	0.034

[2] Available at http://research.microsoft.com/users/tyliu/LETOR/.

In the Trec2004, RankRLS achieves best results when only those pairs that come from the same query and have documents with different relevance levels are used. On the other two datasets, the differences between the performance results of the two approaches are not statistically significant. There seems to be little to be gained from adding the irrelevant pairs to the training data, suggesting that the approach of training only with relevant pairs should be the default approach to take if given no prior information indicating otherwise. Compared to the baseline ranking algorithms, RankRLS achieves very similar performance. The standard RLS regressor, though slightly losing to the ranking algorithms, proves also to be quite a competitive choice.

5.4 Maximizing area under curve

It has been argued that for many types of binary classification tasks the area under the receiver operating characteristics curve (AUC) provides a more fitting performance measure than simple accuracy (Bradley 1997; Provost et al. 1998; Huang and Ling 2005). The task of AUC maximization can be considered as a bipartite ranking problem where each positive input is preferred over each negative one. Thus, it is equivalent to the task of disagreement error minimization (see e.g. Clémençon et al. 2005 for a more detailed consideration). Recent work in the field of support vector machines has shown AUC maximization to be a challenging task (see e.g. Rakotomamonjy 2004; Brefeld and Scheffer 2005; Joachims 2005). The need to consider all positive-negative input pairs easily leads to too cumbersome computations, or the use of approximative heuristics results in gains that are not statistically significant. However, the computational complexity of RankRLS is proportional to the number of individual inputs in the training data instead of the number of input pairs. This makes RankRLS a natural candidate for efficient AUC maximizing learner. For more discussion about this topic, see Pahikkala et al. (2008b).

In our experiments, we evaluate the capability of RankRLS to maximize AUC on the task of assigning topic labels to Reuters newswire documents. We approach the problem by transforming the original multi-label classification task into a series of binary classification tasks, where each sub-task consists of learning to classify documents on the basis of whether they have a certain topic or not.

Similarly to Brefeld and Scheffer (2005), we conduct the experiments on a subset of the Reuters-21578 dataset.[3] We limit the number of inputs in the training data to 500 to test the performance of the ranking methods on small imbalanced datasets. The rest 12397 documents are used as a test data. We take into account only the 25 most numerous classes, each of which corresponds to one possible topic a document can have. We consider the assignment of each of these labels as a separate binary classification problem, where the task is to decide whether a document should have the given label or not. Some of the documents belong to more than one class, and some to none of them.

We use the linear kernel. The regularization parameter λ is set using tenfold cross-validation on the training data, the chosen parameter is the one that provides maximal AUC on the pooled together cross-validation predictions (for a description of the pooling method, see e.g. Bradley 1997). We also calculate the 0.95 confidence intervals for the classifiers' AUC scores for each class. These statistical analyses are performed with SPSS 11.0. The comparison of RankRLS and standard RLS regression results is presented in Table 4 and similar comparison with RankSVM results is presented in Table 5.

[3] Available at http://www.daviddlewis.com/resources/testcollections/reuters21578/.

Table 4 Comparison of the AUC performance of the RankRLS and RLS algorithms on the Reuters-21578 dataset. In the first column is the name of the predicted class and in the next two are the AUC-values and corresponding confidence intervals for the tested algorithms. The last two columns present the numbers of positive inputs in the training set of 500 documents and test data of 12397 documents

Class	RankRLS	RLS Regressor	+train	+test
acq	0.980 (0.978–0.983)	0.979 (0.977–0.982)	94	2275
bop	0.966 (0.947–0.985)	0.880 (0.843–0.917)	4	101
cocoa	0.931 (0.891–0.970)	0.837 (0.776–0.899)	2	71
coffee	0.969 (0.948–0.990)	0.962 (0.950–0.975)	5	134
corn	0.970 (0.959–0.982)	0.950 (0.936–0.964)	11	226
cpi	0.947 (0.925–0.969)	0.601 (0.555–0.648)	3	94
crude	0.976 (0.969–0.982)	0.975 (0.969–0.982)	23	555
dlr	0.971 (0.961–0.981)	0.946 (0.926–0.965)	10	165
earn	0.994 (0.993–0.995)	0.993 (0.991–0.994)	158	3806
gnp	0.987 (0.981–0.993)	0.923 (0.891–0.956)	5	131
gold	0.970 (0.953–0.986)	0.922 (0.897–0.948)	4	120
grain	0.979 (0.973–0.985)	0.974 (0.968–0.980)	23	559
interest	0.965 (0.956–0.974)	0.952 (0.941–0.962)	19	459
livestock	0.701 (0.642–0.761)	0.637 (0.578–0.696)	3	96
money-fx	0.954 (0.946–0.962)	0.947 (0.938–0.957)	28	689
money-supply	0.949 (0.930–0.968)	0.907 (0.877–0.937)	7	165
nat-gas	0.957 (0.933–0.981)	0.941 (0.920–0.962)	5	100
oilseed	0.898 (0.877–0.919)	0.816 (0.783–0.849)	6	165
reserves	0.943 (0.908–0.977)	0.511 (0.458–0.564)	2	71
ship	0.949 (0.934–0.963)	0.925 (0.907–0.942)	13	273
soybean	0.876 (0.839–0.913)	0.805 (0.757–0.853)	4	107
sugar	0.985 (0.979–0.991)	0.964 (0.952–0.976)	6	156
trade	0.978 (0.970–0.986)	0.969 (0.960–0.977)	20	466
veg-oil	0.890 (0.865–0.914)	0.697 (0.656–0.739)	4	120
wheat	0.984 (0.978–0.990)	0.976 (0.969–0.983)	12	271

The results show that the RankRLS clearly outperforms the standard RLS regressor in the task of AUC maximization on the Reuters-21578 dataset. We observe that the smaller the amount of positive inputs is, the larger the performance gains seem to be. Between RankRLS and RankSVM no statistically significant differences are found.

We further examined whether including the ties in the training process has a beneficial or a harmful effect on the ranking performance. The effect was found to be negligible.

5.5 Collaborative filtering

We next present the results of a series of experiments run on a publicly available collaborative filtering dataset, the Jester Joke (Goldberg et al. 2001).[4] The task it to learn to predict the joke preferences of a user based on the preferences of other users, an approach com-

[4] Available at http://www.ieor.berkeley.edu/ goldberg/jester-data/.

Table 5 Comparison of the AUC performance of the RankRLS and RankSVM algorithms on the Reuters-21578 dataset

Class	RankRLS	RankSVM	+train	+test
acq	0.980 (0.978–0.983)	0.979 (0.977–0.982)	94	2275
bop	0.966 (0.947–0.985)	0.966 (0.949–0.983)	4	101
cocoa	0.931 (0.891–0.970)	0.923 (0.881–0.966)	2	71
coffee	0.969 (0.948–0.990)	0.962 (0.939–0.984)	5	134
corn	0.970 (0.959–0.982)	0.966 (0.956–0.975)	11	226
cpi	0.947 (0.925–0.969)	0.947 (0.925–0.969)	3	94
crude	0.976 (0.969–0.982)	0.976 (0.970–0.983)	23	555
dlr	0.971 (0.961–0.981)	0.972 (0.962–0.982)	10	165
earn	0.994 (0.993–0.995)	0.994 (0.992–0.995)	158	3806
gnp	0.987 (0.981–0.993)	0.987 (0.980–0.993)	5	131
gold	0.970 (0.953–0.986)	0.960 (0.940–0.980)	4	120
grain	0.979 (0.973–0.985)	0.979 (0.974–0.984)	23	559
interest	0.965 (0.956–0.974)	0.968 (0.960–0.976)	19	459
livestock	0.701 (0.642–0.761)	0.741 (0.698–0.784)	3	96
money-fx	0.954 (0.946–0.962)	0.959 (0.952–0.966)	28	689
money-supply	0.949 (0.930–0.968)	0.963 (0.950–0.976)	7	165
nat-gas	0.957 (0.933–0.981)	0.957 (0.933–0.981)	5	100
oilseed	0.898 (0.877–0.919)	0.895 (0.873–0.918)	6	165
reserves	0.943 (0.908–0.977)	0.920 (0.902–0.938)	2	71
ship	0.949 (0.934–0.963)	0.951 (0.939–0.964)	13	273
soybean	0.876 (0.839–0.913)	0.882 (0.848–0.916)	4	107
sugar	0.985 (0.979–0.991)	0.977 (0.970–0.985)	6	156
trade	0.978 (0.970–0.986)	0.982 (0.976–0.988)	20	466
veg-oil	0.890 (0.865–0.914)	0.853 (0.818–0.888)	4	120
wheat	0.984 (0.978–0.990)	0.983 (0.978–0.989)	12	271

mon to many recommender systems. In these experiments, we compare the performance of RankRLS with cost functions (20), (21) and (22) as measured by the disagreement error.

Jester Joke is a dataset of joke ratings, where a group of 73496 users has assigned real-valued ratings in the scale -10.0 to 10.0 to a set of 100 jokes, each rating describing how much they liked/disliked the joke in question. The task is to learn to predict the preferences of individual users from the preferences of the other users.

Our experimental setting follows that of Cortes et al. (2007b). We choose a set of 300 active users, for whom the task is to learn to predict their joke preferences. For each user, half of the jokes are chosen for training and half for testing. The preferences of the users are derived from the differences of the rating scores, a joke with a higher score is preferred to a joke with a lower score. To generate the features for the instances, a set of 300 reference users is chosen, and their given ratings for the corresponding joke are used as the feature values. In cases where these users have not rated the joke, the median of their ratings is used as the feature value.

In accordance to the original experimental setup, we perform three rounds of experiments, where we first choose the reference reviewers from people with 20–40, then with 40–60, and finally with 60–80 ratings. Finally, we remove these restrictions on feature den-

Table 6 Disagreement errors for the different versions of RankRLS, RankSVM, and the basic RLS regressor on the Jester dataset. RankRLS is tested with the cost functions (20), (21), and (22)

Method	20–40	40–60	60–80	All sizes
RankRLS (20)	0.413	0.400	0.378	0.371
RankRLS (21)	0.413	0.400	0.379	0.371
RankRLS (22)	0.445	0.426	0.388	0.379
RankSVM	0.413	0.400	0.378	0.371
RLS Regressor	0.414	0.401	0.378	0.371

sities and perform a fourth round of experiments using simply randomly chosen set of reference users. The kernel used is the Gaussian kernel and its width parameter was chosen from the interval $[2^{-15}, 2^{-14}, \ldots, 2^{14}, 2^{15}]$. The parameters for each experiment are chosen by taking the average over the performances on a hold-out set. The hold-out sets are created for each experiment similarly as the corresponding training/test data.

The results of the collaborative filtering experiments are included in Table 6. In these experiments, we found no difference between the performance of the cost functions (20) and (21). Further, we noticed that the cost function weighted by the inverse of the magnitude of the difference (22) performed worse than the other cost functions. This difference was statistically significant in each of the test settings. The performance of RankSVM was identical with that of the discretized (20) and magnitude preserving cost functions (21). Further, standard RLS also achieved as good performance as the ranking algorithms. We also tested the effects of including all pairs instead of only relevant ones in the training data. No performance differences were observed in the results.

6 Conclusions

There are many problems in which the aim is not to classify or to regress but to learn a ranking function. Inspired by the recent success of the regularized least-squares (RLS) based algorithms, we introduce a framework for RLS based ranking cost functions. Further, we propose three cost functions. We investigate their benefits and drawbacks from the perspectives of applicability and computational complexity. Finally, we propose a kernel-based preference learning algorithm, which we call RankRLS, for minimizing such cost functions.

RankRLS can be trained with a sequence of pairwise preferences between input data points and it outputs a ranking function for the individual inputs. The training time of RankRLS grows linearly with respect to the number of preferences and is cubic either with respect to the number of inputs or to the number of dimensions in the input space.

An important special case is the one in which the preference relation is induced by a scoring of input data points. For this case, it is possible to develop efficient shortcut methods using techniques based on matrix calculus. Namely, we introduce training algorithms whose complexities do not depend on the number of preferences, cross-validation algorithms for both object and label ranking, method for selection of the regularization parameter, and a method for learning multiple scorings simultaneously. These methods can be combined together. In addition, we show that some of these efficient methods can also be used in large-scale learning when the sparse approximation is used.

We also make a thorough comparison of the computational benefits and drawbacks of RankRLS in both small-scale and large-scale learning tasks with those of RankSVM that can be considered as a state-of-the-art ranking method. Moreover, both the linear and non-linear learning problems are considered in the comparison. While a single instance of a

RankSVM may be faster to train than a single instance of RankRLS in the linear learning tasks, the computational shortcuts of RankRLS in cross-validation, parameter selection, and multiple output learning make RankRLS in many situations much faster method to use than RankSVM. This is especially the case if nonlinear kernel functions are used and if cross-validation is used for performance estimation.

We evaluate RankRLS on four tasks with different characteristics. We use as the baseline method RankSVM. The results show that the performance of RankSVM and RankRLS is very similar. Further, the three proposed cost functions are compared with each other and it is found that the performance differences are task dependent. We also show that in all of the experiments RankRLS always performs better or as well as the RLS regressor trained to regress the scores of the input data points. Often some of the pairs of input data points are not relevant with respect to the learning task in question. We show that they may be even harmful to the ranking performance, because the RankRLS algorithm has to minimize their RLS error at the expense of the relevant pairs.

There has been recently discussion within the community about the importance of sharing open source implementations of introduced methods (see Sonnenburg et al. 2007). Inspired by this, we make freely available a software package called RLScore containing an implementation of RankRLS and the efficient cross-validation methods.[5]

Acknowledgements This work has been supported by Academy of Finland and Tekes, the Finnish Funding Agency for Technology and Innovation. We would like to thank CSC, the Finnish IT center for science, for providing us extensive computing resources. We are grateful to Hanna Suominen for her contributions to the document classification experiments. We also thank the anonymous reviewers for their insightful comments.

References

Agarwal, S. (2006). Ranking on graph data. In W. W. Cohen & A. Moore (Eds.), *ACM international conference proceeding series: Vol. 148. Proceedings of the 23rd international conference on machine learning* (pp. 25–32). New York: ACM.

Agarwal, S., & Niyogi, P. (2005). Stability and generalization of bipartite ranking algorithms. In P. Auer & R. Meir (Eds.), *Lecture notes in computer science: Vol. 3559. Proceedings of the 18th annual conference on learning theory* (pp. 32–47). Berlin: Springer.

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

Brefeld, U., & Scheffer, T. (2005). AUC maximizing support vector learning. In N. Lachiche, C. Ferri, S. A. Macskassy, & A. Rakotomamonjy (Eds.), *Proceedings of the 2nd workshop on ROC analysis in machine learning (ROCML'05)*.

Brualdi, R. A., & Ryser, H. J. (1991). *Combinatorial matrix theory*. Cambridge: Cambridge University Press.

Clémençon, S., Lugosi, G., & Vayatis, N. (2005). Ranking and scoring using empirical risk minimization. In P. Auer & R. Meir (Eds.), *Lecture notes in computer science: Vol. 3559. Proceedings of the 18th annual conference on learning theory* (pp. 1–15). Berlin: Springer.

Cortes, C., Mohri, M., & Rastogi, A. (2007a). An alternative ranking problem for search engines. In C. Demetrescu (Ed.), *Lecture notes in computer science: Vol. 4525. Proceedings of the 6th workshop on experimental algorithms* (pp. 1–21). Berlin: Springer.

Cortes, C., Mohri, M., & Rastogi, A. (2007b). Magnitude-preserving ranking algorithms. In Z. Ghahramani (Ed.), *ACM international conference proceeding series: Vol. 227. Proceedings of the 24th annual international conference on machine learning* (pp. 169–176). New York: ACM.

Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal Machine Learning Research*, *4*, 933–969.

Fürnkranz, J., & Hüllermeier, E. (2005). Preference learning. *Künstliche Intelligenz*, *19*(1), 60–61.

[5] Available at http://www.tucs.fi/RLScore.

Gestel, T. V., Suykens, J. A. K., Baesens, B., Viaene, S., Vanthienen, J., Dedene, G., Moor, B. D., & Vandewalle, J. (2004). Benchmarking least squares support vector machine classifiers. *Machine Learning*, *54*(1), 5–32.

Goldberg, K., Roeder, T., Gupta, D., & Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, *4*(2), 133–151.

Golland, P., Liang, F., Mukherjee, S., & Panchenko, D. (2005). Permutation tests for classification. In P. Auer & R. Meir (Eds.), *Lecture notes in computer science: Vol. 3559. Proceedings of the 18th annual conference on learning theory* (pp. 501–515). Berlin: Springer.

Herbrich, R., Graepel, T., & Obermayer, K. (1999). Support vector learning for ordinal regression. In *Proceedings of the ninth international conference on artificial neural networks* (pp. 97–102). London, Institute of Electrical Engineers.

Horn, R., & Johnson, C. R. (1985). *Matrix analysis*. Cambridge: Cambridge University Press.

Huang, J., & Ling, C. X. (2005). Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, *17*(3), 299–310.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In D. Hand, D. Keim, & R. Ng (Eds.), *Proceedings of the 8th ACM SIGKDD conference on knowledge discovery and data mining* KDD'02 (pp. 133–142). New York: ACM.

Joachims, T. (2005). A support vector method for multivariate performance measures. In L. D. Raedt & S. Wrobel (Eds.), *ACM international conference proceeding series: Vol. 119. Proceedings of the 22nd international conference on machine learning* (pp. 377–384). New York: ACM.

Joachims, T. (2006). Training linear SVMs in linear time. In T. Eliassi-Rad, L. H. Ungar, M. Craven, & D. Gunopulos (Eds.), *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* KDD'06 (pp. 217–226). New York: ACM.

Johnson, R., & Zhang, T. (2008). Graph-based semi-supervised learning and spectral kernel design. *IEEE Transactions on Information Theory*, *54*(1), 275–288.

Liu, T.-Y., Xu, J., Qin, T., Xiong, W., & Li, H. (2007). LETOR: Benchmark dataset for research on learning to rank for information retrieval. In T. Joachims, H. Li, T.-Y. Liu, & C. Zhai (Eds.), *SIGIR 2007 workshop on learning to rank for information retrieval* (pp. 3–10).

Pahikkala, T., Airola, A., Boberg, J., & Salakoski, T. (2008a). Exact and efficient leave-pair-out cross-validation for ranking RLS. In T. Honkela, M. Pöllä, M.-S. Paukkeri, & O. Simula (Eds.), *Proceedings of the 2nd international and interdisciplinary conference on adaptive knowledge representation and reasoning (AKRR'08)* (pp. 1–8). Helsinki University of Technology.

Pahikkala, T., Airola, A., Suominen, H., Boberg, J., & Salakoski, T. (2008b). Efficient AUC maximization with regularized least-squares. In A. Holst, P. Kreuger, & P. Funk (Eds.), *Frontiers in artificial intelligence and applications: Vol. 173. Proceedings of the 10th Scandinavian conference on artificial intelligence* SCAI, 2008 (pp. 12–19). Amsterdam: IOS Press.

Pahikkala, T., Boberg, J., & Salakoski, T. (2006a). Fast n-fold cross-validation for regularized least-squares. In T. Honkela, T. Raiko, J. Kortela, & H. Valpola (Eds.), *Proceedings of the ninth Scandinavian conference on artificial intelligence*, Espoo, Finland (pp. 83–90). Otamedia Oy.

Pahikkala, T., Suominen, H., Boberg, J., & Salakoski, T. (2007a). Transductive ranking via pairwise regularized least-squares. In P. Frasconi, K. Kersting, & K. Tsuda (Eds.), *Workshop on mining and learning with graphs* (pp. 175–178).

Pahikkala, T., Tsivtsivadze, E., Airola, A., Boberg, J., & Salakoski, T. (2007b). Learning to rank with pairwise regularized least-squares. In T. Joachims, H. Li, T.-Y. Liu, C. Zhai (Eds.), *SIGIR 2007 workshop on learning to rank for information retrieval* (pp. 27–33).

Pahikkala, T., Tsivtsivadze, E., Boberg, J., & Salakoski, T. (2006b). Graph kernels versus graph representations: a case study in parse ranking. In T. Gärtner, G. C. Garriga, & T. Meinl (Eds.), *Proceedings of the ECML/PKDD'06 workshop on mining and learning with graphs*, Berlin, Germany (pp. 181–188).

Parker, B. J., Gunter, S., & Bedo, J. (2007). Stratification bias in low signal microarray studies. *BMC Bioinformatics*, *8*, 326.

Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, *78*(9), 1481–1497.

Provost, F. J., Fawcett, T., & Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. In J. Shavlik (Ed.), *Proceedings of the fifteenth international conference on machine learning* (pp. 445–453). San Mateo: Morgan Kaufmann.

Pyysalo, S., Ginter, F., Heimonen, J., Björne, J., Boberg, J., Järvinen, J., & Salakoski, T. (2007). BioInfer: A corpus for information extraction in the biomedical domain. *BMC Bioinformatics*, *8*, 50.

Pyysalo, S., Ginter, F., Pahikkala, T., Boberg, J., Järvinen, J., & Salakoski, T. (2006). Evaluation of two dependency parsers on biomedical corpus targeted at protein-protein interactions. *Recent Advances in Natural Language Processing for Biomedical Applications, special issue of the International Journal of Medical Informatics*, *75*(6), 430–442.

Quiñonero-Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, *6*, 1939–1959.

Quiñonero-Candela, J., Rasmussen, C. E., & Williams, C. K. I. (2007). Approximation methods for Gaussian process regression. In L. Bottou, O. Chapelle, D. DeCoste, & J. Weston (Eds.), *Large-scale kernel machines* (pp. 203–224). Cambridge: MIT Press.

Rakotomamonjy, A. (2004). Optimizing area under ROC curve with SVMs. In J. Hernández-Orallo, C. Ferri, N. Lachiche, & P. A. Flach (Eds.), *Proceedings of the 1st international workshop on ROC analysis in artificial intelligence* (pp. 71–80).

Rifkin, R. (2002). *Everything old is new again: a fresh look at historical approaches in machine learning*. Ph.D. thesis, Massachusetts Institute of Technology.

Rifkin, R., & Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, *5*, 101–141.

Rifkin, R., & Lippert, R. (2007a). *Notes on regularized least squares* (Technical Report MIT-CSAIL-TR-2007-025). Massachusetts Institute of Technology.

Rifkin, R., & Lippert, R. (2007b). Value regularization and Fenchel duality. *Journal of Machine Learning Research*, *8*, 441–479.

Rifkin, R., Yeo, G., & Poggio, T. (2003). Regularized least-squares classification. In J. Suykens, G. Horvath, S. Basu, C. Micchelli, & J. Vandewalle (Eds.), *NATO science series III: computer and system sciences: Vol. 190. Advances in learning theory: methods, model and applications* (pp. 131–154). Amsterdam: IOS Press. Chap. 7.

Schölkopf, B., Herbrich, R., & Smola, A. J. (2001). A generalized representer theorem. In D. Helmbold & R. Williamson (Eds.), *Proceedings of the 14th annual conference on computational learning theory and 5th European conference on computational learning theory* (pp. 416–426). Berlin: Springer.

Schölkopf, B., Mika, S., Burges, C., Knirsch, P., Müller, K.-R., Rätsch, G., & Smola, A. (1999). Input space versus feature space in kernel-based methods. *IEEE Transactions On Neural Networks*, *10*(5), 1000–1017.

Shewchuk, J. R. (1994). *An introduction to the conjugate gradient method without the agonizing pain* (Technical Report CMU-CS-94-125). Carnegie Mellon University, Pittsburgh, PA, USA.

Sleator, D. D., & Temperley, D. (1991). *Parsing English with a link grammar* (Technical Report CMU-CS-91-196). Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

Smola, A. J., & Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. In P. Langley (Ed.), *Proceedings of the seventeenth international conference on machine learning* (pp. 911–918). San Mateo: Morgan Kaufmann.

Sonnenburg, S., Braun, M. L., Ong, C. S., Bengio, S., Bottou, L., Holmes, G., Lecun, Y., Müller, K. R., Pereira, F., Rasmussen, C. E., Rätsch, G., Schölkopf, B., Smola, A., Vincent, P., Weston, J., & Williamson, R. (2007). The need for open source software in machine learning. *Journal of Machine Learning Research*, *8*, 2443–2466.

Suykens, J. A. K., & Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural Processing Letters*, *9*(3), 293–300.

Tsivtsivadze, E., Pahikkala, T., Airola, A., Boberg, J., & Salakoski, T. (2008). A sparse regularized least-squares preference learning algorithm. In A. Holst, P. Kreuger, & P. Funk (Eds.), *Frontiers in artificial intelligence and applications: Vol. 173. Proceedings of the 10th Scandinavian conference on artificial intelligence* SCAI, 2008 (pp. 76–83). Amsterdam: IOS Press.

Tsivtsivadze, E., Pahikkala, T., Pyysalo, S., Boberg, J., Mylläri, A., & Salakoski, T. (2005). Regularized least-squares for parse ranking. In A. F. Famili, J. N. Kok, J. M. Peña, A. Siebes, & A. J. Feelders (Eds.), *Lecture notes in computer science: Vol. 3646. Proceedings of the 6th international symposium on intelligent data analysis* (pp. 464–474). Berlin: Springer.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, *1*, 80–83.

Zhang, P., & Peng, J. (2004). SVM vs regularized least squares classification. In J. Kittler, M. Petrou, & M. Nixon (Eds.), *Proceedings of the 17th international conference on pattern recognition* ICPR'04 (Vol. 1, pp. 176–179). Los Alamitos: IEEE Computer Society.