Greedy RankRLS: a Linear Time Algorithm for Learning Sparse Ranking Models

Tapio Pahikkala
University of Turku and Turku Centre for Computer Science (TUCS), Turku, Finland
firstname.lastname@utu.fi

Antti Airola
University of Turku and Turku Centre for Computer Science (TUCS), Turku, Finland
firstname.lastname@utu.fi

Pekka Naula
University of Turku
Turku, Finland
firstname.lastname@utu.fi

Tapio Salakoski
University of Turku and Turku Centre for Computer Science (TUCS), Turku, Finland
firstname.lastname@utu.fi

ABSTRACT

Ranking is a central problem in information retrieval. Much work has been done in the recent years to automate the development of ranking models by means of supervised machine learning. Feature selection aims to provide sparse models which are computationally efficient to evaluate, and have good ranking performance. We propose integrating the feature selection as part of the training process for the ranking algorithm, by means of a wrapper method which performs greedy forward selection, using leave-query-out cross-validation estimate of performance as the selection criterion. We introduce a linear time training algorithm we call greedy RankRLS, which combines the aforementioned procedure, together with regularized risk minimization based on pairwise least-squares loss. The training complexity of the method is $O(kmn)$, where k is the number of features to be selected, m is the number of training examples, and n is the overall number of features. Experiments on the LETOR benchmark data set demonstrate that the approach works in practice.

Keywords

feature selection, learning to rank, ranking, RankRLS, regularized least-squares, variable selection

1. INTRODUCTION

Learning to rank for information retrieval has been a topic of intense research during the recent years. The possible benefits of automatically inducing ranking models from data, compared to purely handcrafted systems, include reduced manual labor, increased ranking performance, and adaptivity to individual user preferences. A number of supervised machine learning methods have been proposed, and successfully applied for this task. These include both pairwise approaches such as RankSVM [8], RankNet [2], and RankRLS [16, 17], as well approaches which optimize multivariate loss functions defined over queries, also known as the listwise approach [3, 4, 25].

In this article we consider the task of feature selection for learning to rank, specifically concentrating on the task of document retrieval. The task is to recognize an informative subset of the features, such that a machine learning method trained on the subset achieves good ranking performance on unseen future data. Perhaps the most fundamental advantage of this approach is that it leads to sparse models, as only a limited subset of features is used for prediction. Since applications such as web-search engines are typically constrained by strict real-time response demands, being able to restrict the number of features that need to be calculated can be quite useful. Further, feature selection can also provide feedback on the quality of different features, which can be very useful when developing and testing new ones.

Feature selection methods are typically divided into three categories [6]. In the filter approach the features are selected as a pre-processing step before applying a learning algorithm, wrapper methods select features through interaction with a learning algorithm, and embedded methods perform the selection as part of the learning process itself. Feature selection for ranking is not as of yet a well studied area, but a number of approaches have been introduced during the past few years. Geng et al. [5] proposed a filter method for selecting such features which produce good rankings, while at the same time aiming to minimize the redundancy in the set of selected features. The work was further improved upon by Yu et al. [24]. Metzler [12] and Pan et al. [19] considered feature selection for ranking with Markov random fields and boosted trees, respectively.

Since the final goal of the feature selection process is to produce a sparse ranking model with good performance, we argue that the most natural selection criterion is the so-called wrapper approach [6, 10, 11]. We select such features, which result in maximal ranking performance for the supervised learning method that we are actually using to learn our model. A standard approach for estimating the generalization performance of a model trained on a subset of features

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGIR Workshop on Feature Generation and Selection for Information Retrieval 2010 Geneva, Switzerland
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$10.00.

is to use cross-validation, as proposed by [10] for wrapper based feature selection. More specifically, we propose using leave-query-out (LQO) cross-validation. This allows one to make maximal use of training data, and guarantees that data points related to the same query are never split between the training and test folds. Further, to make the search over the power set of features feasible, we propose to use greedy forward selection, where on each iteration the feature whose addition yields best cross-validation performance is selected.

In this article we propose the greedy RankRLS algorithm, that is able to efficiently perform the aforementioned selection procedure. The algorithm is equivalent to a wrapper method that for each tested feature set, and each round of cross-validation would train the RankRLS method [16, 17], which minimizes the pairwise regularized least-squares loss. It can also be considered as an embedded method, since the proposed training algorithm for greedy RankRLS training is far more efficient than the straightforward approach of using RankRLS as a black-box method within the selection process would be. Previously, RankRLS has been shown to produce good ranking results in document retrieval, and in general achieve ranking performance similar to that of RankSVM [16, 17].

To achieve computational efficiency for the wrapper method, we combine the training algorithm with matrix algebra based shortcuts. These are made possible by the fact that RankRLS has a closed form solution, which can be fully expressed in terms of matrix operations. Firstly, the developed shortcuts allow efficient update of the solution when new features are added, without having to recompute the solution from scratch. We have previously proposed similar shortcuts for the greedy RLS algorithm [13,14], which allows one to train sparse regressors and classifiers in linear time. Second, based on the results in [1,15] we derive a formula for the exact LQO estimate that is more efficient than the one previously proposed in [16], and combine it with the update operation for feature addition. The resulting complexity of greedy RankRLS training is $O(kmn)$, where k is the number of features to be selected, m is the number of training examples, and n is the overall number of features. The memory complexity of the method is $O(mn)$. We are not aware of as efficient greedy forward selection methods with cross-validation based selection criterion for other state-of-the-art learning to rank methods.

2. SETTING

We start by introducing some notation. Let $\mathbb{R}^m$ and $\mathbb{R}^{n\times m}$, where $n, m \in \mathbb{N}$, denote the sets of real valued column vectors and $n \times m$-matrices, respectively. To denote real valued matrices and vectors we use bold capital letters and bold lower case letters, respectively. Moreover, index sets are denoted with calligraphic capital letters. By denoting $\mathbf{M}_i$, $\mathbf{M}_{:,j}$, and $\mathbf{M}_{i,j}$, we refer to the ith row, jth column, and i, jth entry of the matrix $\mathbf{M} \in \mathbb{R}^{n\times m}$, respectively. Similarly, for index sets $\mathcal{R} \subseteq \{1, \ldots, n\}$ and $\mathcal{L} \subseteq \{1, \ldots, m\}$, we denote the submatrices of $\mathbf{M}$ having their rows indexed by $\mathcal{R}$, the columns by $\mathcal{L}$, and the rows by $\mathcal{R}$ and columns by $\mathcal{L}$ as $\mathbf{M}_{\mathcal{R}}$, $\mathbf{M}_{:,\mathcal{L}}$, and $\mathbf{M}_{\mathcal{R},\mathcal{L}}$, respectively. We use an analogous notation also for column vectors, that is, $\mathbf{v}_i$ refers to the ith entry of the vector $\mathbf{v}$.

We assume, that we are given a training set in a form of data matrix $\mathbf{X} \in \mathbb{R}^{n\times m}$ and a label vector $\mathbf{y} \in \mathbb{R}^m$. The rows of the data matrix are indexed by the n features and the columns by the m training examples. Thus, by $\mathbf{X}_{:,i}$ we denote the column vector containing the features of the ith example, sliced from the data matrix and by $\mathbf{y}_i$ we denote its corresponding real value label. Let $\mathcal{I} = \{1 \ldots m\}$ denote the index set for the training set. The index set is divided into a number of disjoint queries, where $Q = \{\mathcal{Q}^{(1)}, \ldots, \mathcal{Q}^{(|Q|)}\}$ is the set of queries, and $\mathcal{Q}^{(i)} \subset \mathcal{I}$, $\bigcup_{i=1}^{|Q|} \mathcal{Q}^{(i)} = \mathcal{I}$ and $\mathcal{Q}^{(i)} \cap \mathcal{Q}^{(j)} = \emptyset$, if $i \neq j$. Each example represents a query-document pair. The features are a joint feature representation for the query the example is associated with, and a candidate document, and the label denotes how relevant the document is with respect to the query. The ranking of the documents associated with a query can be obtained by sorting them according to the values of their labels.

In feature selection, the task is to select a subset $\mathcal{S} \subset \{1 \ldots n\}$, $|\mathcal{S}| = k$, of the n available features, such that the resulting predictor is sparse, but still produces a good ranking performance on new data. The number of selected features k may be decided in advance, or selected against a validation set, according to application specific criteria. We consider linear predictors of type

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_{\mathcal{S}}, \tag{1}$$

where $\mathbf{w}$ is the k-dimensional vector representation of the learned predictor and $\mathbf{x}_{\mathcal{S}}$ can be considered as a mapping of the data point x into k-dimensional feature space. Note that the vector $\mathbf{w}$ only contains entries corresponding to the features indexed by $\mathcal{S}$. The rest of the features of the data points are not used in prediction phase. The computational complexity of making predictions with (1) and the space complexity of the predictor are both $O(k)$, provided that the feature vector representation $\mathbf{x}_{\mathcal{S}}$ for the data point x is given.

The pairwise ranking error for a learned predictor f can be defined as

$$\frac{1}{|Q|}\sum_{\mathcal{Q}^{(i)}\in Q}\frac{1}{N^{(i)}}\sum_{j,k\in\mathcal{Q},\mathbf{y}_j<\mathbf{y}_k} H(f(\mathbf{X}_{:,j}) - f(\mathbf{X}_{:,k})), \tag{2}$$

where H is the Heaviside step function defined as

$$H(a) = \begin{cases} 1, & \text{if } a > 0 \\ 1/2, & \text{if } a = 0 \\ 0, & \text{if } a < 0 \end{cases}$$

and $N^{(i)}$ is the number of pairs in the i:th query, for which $\mathbf{y}_j < \mathbf{y}_k$ holds true. In this definition, the error is normalized so that each considered query has the same importance, regardless of size. Since (2) is non-convex, successful pairwise approaches for learning to rank typically minimize convex approximations instead.

The RankRLS algorithm [16, 17] is based on regularized risk minimization, where a least-squares based approximation of (2) is minimized, together with a quadratic regularizer. We approximate (2) with the pairwise least-squares loss, where step function H is replaced with the pairwise squared loss

$$l(i, j) = (\mathbf{y}_i - \mathbf{y}_j - f(\mathbf{X}_{:,i}) + f(\mathbf{X}_{:,j}))^2.$$

Compared to the discrete pairwise loss, this loss also enforces the magnitudes of the prediction differences. For the purpose of simplifying the derivation and implementation of the learning algorithm, we modify the normalizers, and also

include tied predictions within the same query in the loss. The linear RankRLS solution is found by solving

$$\underset{\mathbf{w}\in\mathbb{R}^{|\mathcal{S}|}}{\operatorname{argmin}} \left\{ \sum_{\mathcal{Q}\in Q} \frac{1}{2|\mathcal{Q}|} \sum_{i,j\in\mathcal{Q}} l(i,j) + \lambda\|\mathbf{w}\|^2 \right\} \tag{3}$$

where the first term is the empirical risk measuring how well the model determined by $\mathbf{w}$ fits to the training data, and the second term called regularizer penalizes complex models. The regularization parameter $\lambda > 0$ controls the tradeoff between these terms.

Let

$$\mathbf{L}_l = \mathbf{I} - \frac{1}{l}\mathbf{1}\mathbf{1}^{\mathrm{T}}$$

be the $l \times l$-centering matrix with $l \in \mathbb{N}$. The matrix $\mathbf{L}$ is an idempotent matrix and multiplying it with a vector removes the mean of the vector entries from all elements of the vector. Moreover, the following equality can be shown

$$\frac{1}{2l}\sum_{i,j=1}^{l}(c_i - c_j) = \mathbf{c}^{\mathrm{T}}\mathbf{L}_l\mathbf{c}\,,$$

where c_i are the entries of any vector $\mathbf{c}$. Without loss of generality, we can assume that the training data is ordered according to the queries, so that first come the examples belonging to the first query, next to the second, etc. Now, let us consider the following quasi-diagonal matrix:

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_{l_1} & & \\ & \ddots & \\ & & \mathbf{L}_{l_{|Q|}} \end{pmatrix},$$

where $l_i = |\mathcal{Q}_i|$ for $i \in \{1,\ldots,|Q|\}$. The matrix $\mathbf{L}$ is again symmetric and idempotent, and can be interpreted as a query-wise centering matrix, that removes the mean from the prediction errors for each query [18]. It is also a normalized version of the Laplacian matrix encoding the structure of the preference graph induced by the queries, which has been used in previous derivations of RankRLS [16, 17].

Now, (3) can be re-written in matrix notation as

$$\underset{\mathbf{w}\in\mathbb{R}^{|\mathcal{S}|}}{\operatorname{argmin}} \left\{ ((\mathbf{w}^{\mathrm{T}}\mathbf{X}_{\mathcal{S}})^{\mathrm{T}} - \mathbf{y})^{\mathrm{T}}\mathbf{L}((\mathbf{w}^{\mathrm{T}}\mathbf{X}_{\mathcal{S}})^{\mathrm{T}} - \mathbf{y}) + \lambda\mathbf{w}^{\mathrm{T}}\mathbf{w} \right\}. \tag{4}$$

Analogously to the results in [16, 17], a minimizer of (4) is

$$\mathbf{w} = (\mathbf{X}_{\mathcal{S}}\mathbf{L}(\mathbf{X}_{\mathcal{S}})^{\mathrm{T}} + \lambda\mathbf{I})^{-1}\mathbf{X}_{\mathcal{S}}\mathbf{L}\mathbf{y}.$$

Due to the symmetry and idempotence of $\mathbf{L}$, this can be re-written as

$$\mathbf{w} = (\widehat{\mathbf{X}}_{\mathcal{S}}(\widehat{\mathbf{X}}_{\mathcal{S}})^{\mathrm{T}} + \lambda\mathbf{I})^{-1}\widehat{\mathbf{X}}_{\mathcal{S}}\widehat{\mathbf{y}}, \tag{5}$$

where $\widehat{\mathbf{X}}_{\mathcal{S}} = \mathbf{X}_{\mathcal{S}}\mathbf{L}$ and $\widehat{\mathbf{y}} = \mathbf{L}\mathbf{y}$. Using the sparse decomposition of $\mathbf{L}$, the first multiplication can be performed in $O(m|\mathcal{S}|)$, and the second in $O(m)$ time. We note (see e.g. [7]) that equivalently, the RankRLS solution can be obtained from

$$\mathbf{w} = \widehat{\mathbf{X}}_{\mathcal{S}}((\widehat{\mathbf{X}}_{\mathcal{S}})^{\mathrm{T}}\widehat{\mathbf{X}}_{\mathcal{S}} + \lambda\mathbf{I})^{-1}\widehat{\mathbf{y}}. \tag{6}$$

The overall complexity of solving (5) is $O(m|\mathcal{S}|^2 + |\mathcal{S}|^3)$, and that of solving (6) $O(m^2|\mathcal{S}| + m^3)$. We note that these are solutions to the ordinary regularized least squares (RLS) problem. Thus, by the query-wise centering of the data matrix the RankRLS problem can be mapped to that of solving the ordinary RLS problem.

Finally, we consider the issue of cross-validation. As discussed before, we aim to perform LQO cross-validation, where in turn each query is left out of the training set, and used for testing. Let $\mathcal{Q}$ be the index set of a query, and let $\overline{\mathcal{Q}} = \mathcal{I} \setminus \mathcal{Q}$ be the complement of this index set.

Let us consider the centered data matrix from which the rows corresponding to the $\mathcal{Q}$ have been removed, $\widehat{\mathbf{X}}_{\mathcal{S}\overline{\mathcal{Q}}}$. Due to the quasi-diagonal structure of $\mathbf{L}$, the submatrices $\mathbf{L}_{\mathcal{Q}\overline{\mathcal{Q}}}$ have only zero entries. Therefore, we have

$$\mathbf{X}_{\mathcal{S}\overline{\mathcal{Q}}}\mathbf{L}_{\overline{\mathcal{Q}}\overline{\mathcal{Q}}} = \widehat{\mathbf{X}}_{\mathcal{S}\overline{\mathcal{Q}}} - \mathbf{X}_{\mathcal{S}\mathcal{Q}}\mathbf{L}_{\mathcal{Q}\overline{\mathcal{Q}}} = \widehat{\mathbf{X}}_{\mathcal{S}\overline{\mathcal{Q}}}.$$

The significance of this result is that when removing a query from the training set, we can recover the centered representation of the remaining data simply by slicing $\widehat{\mathbf{X}}$. The centering operation does not need to be re-calculated. The feature representation for the test query is also centered, if we recover it from the centered training data matrix. Since using centered data does not affect the relative ordering or relative differences in prediction values, as long as ranking based performance measures are used, this makes no difference.

These results considerably simplify the development of efficient cross-validation methods for RankRLS. As long as folds are defined along query lines, the task of performing cross-validation with RankRLS is identical to that of performing cross-validation with RLS using centered training data. In problem settings where $\mathbf{L}$ does not have quasi-diagonal structure, such as when learning from a single global ranking, such results do not exist, making development of cross-validation shortcuts more challenging.

3. ALGORITHM DESCRIPTION

Here, we present the computational short-cuts that enable the efficient feature subset search strategy for RankRLS with LQO error as a heuristic. First, we recall an approach for computing the hold-out error for the RLS algorithm. By hold-out, we indicate the method that is used to estimate the performance of the learning algorithm by holding a part of the given data set as a test set and training a learner with the rest of the data. Our hold-out formulation assumes that the whole data set is used to train a RLS predictor and the hold-out set is then "unlearned" afterwards. The formulation can then be used, for example, to perform a N-fold CV by holding out a different part of the data set at a time and averaging the results. In this paper, we use it specifically for LQO-CV, that is, each query is held out from the training set at a time, and for a particular query, the corresponding hold-out set consists of all the training examples associated with the query.

Now, let us define

$$\mathbf{G} = ((\widehat{\mathbf{X}}_{\mathcal{S}})^{\mathrm{T}}\widehat{\mathbf{X}}_{\mathcal{S}} + \lambda\mathbf{I})^{-1}$$

and

$$\mathbf{a} = \mathbf{G}\widehat{\mathbf{y}}.$$

The following theorem can be straightforwardly inferred from the results presented by [1, 15].

THEOREM 3.1. *The predictions for the data points indexed by $\mathcal{Q}$ made by a RLS predictor trained using the features indexed by $\mathcal{S}$ and with the whole training set except the examples indexed by $\mathcal{Q}$ can be obtained from*

$$\widehat{\mathbf{y}}_{\mathcal{Q}} - (\mathbf{G}_{\mathcal{Q}\mathcal{Q}})^{-1}\mathbf{a}_{\mathcal{Q}}.$$

According to the above theorem, the result of LQO-CV with squared error as a performance measure can be obtained from

$$\sum_{\mathcal{Q} \in Q} (\mathbf{p}_{\mathcal{Q}})^{\mathrm{T}} \mathbf{p}_{\mathcal{Q}}, \tag{7}$$

where

$$\mathbf{p}_{\mathcal{Q}} = (\mathbf{G}_{\mathcal{Q}\mathcal{Q}})^{-1} \mathbf{a}_{\mathcal{Q}}.$$

It is quite straightforward to show that the vector of hold-out errors is centered query-wise, that is, $\mathbf{p} = \mathbf{L}\mathbf{p}$, because we use $\widehat{\mathbf{X}}$ and $\widehat{\mathbf{y}}$ in place of $\mathbf{X}$ and $\mathbf{y}$. Therefore, the sum of squared hold-out errors (7) is, in fact, the sum of squared query-wise centered hold-out errors. As shown earlier, this corresponds to the sum of pairwise squared losses, calculated for each query separately.

Algorithm 1: Greedy RankRLS

Input: $\widehat{\mathbf{X}} \in \mathbb{R}^{n \times m}$, $\widehat{\mathbf{y}} \in \mathbb{R}^m$, k, λ
Output: $\mathcal{S}$, $\mathbf{w}$

$\mathbf{a} \leftarrow \lambda^{-1}\widehat{\mathbf{y}}$;
$\mathbf{C} \leftarrow \lambda^{-1}\widehat{\mathbf{X}}^{\mathrm{T}}$;
$\mathbf{U} \leftarrow \widehat{\mathbf{X}}^{\mathrm{T}}$;
$\mathbf{p} \leftarrow \widehat{\mathbf{y}}$;
$\mathcal{S} \leftarrow \emptyset$;
while $|\mathcal{S}| < k$ **do**
  $e \leftarrow \infty$;
  $b \leftarrow 0$;
  foreach $i \in \{1, \ldots, n\} \setminus \mathcal{S}$ **do**
    $c \leftarrow (1 + \widehat{\mathbf{X}}_i \mathbf{C}_{:,i})^{-1}$;
    $d \leftarrow c\mathbf{C}^{\mathrm{T}}{}_i \widehat{\mathbf{y}}$;
    $e_i \leftarrow 0$;
    foreach $\mathcal{Q} \in Q$ **do**
      $\gamma \leftarrow (-c^{-1} + \mathbf{C}^{\mathrm{T}}{}_{i,\mathcal{Q}} \mathbf{U}_{\mathcal{Q},i})^{-1}$;
      $\tilde{\mathbf{p}}_{\mathcal{Q}} \leftarrow \mathbf{p}_{\mathcal{Q}} - d\mathbf{U}_{\mathcal{Q},i} - \gamma \mathbf{U}_{\mathcal{Q},i}(\mathbf{U}^{\mathrm{T}}{}_{i,\mathcal{Q}}(\mathbf{a}_{\mathcal{Q}} - d\mathbf{C}_{\mathcal{Q},i}))$;
      $e_i \leftarrow e_i + (\tilde{\mathbf{p}}_{\mathcal{Q}})^{\mathrm{T}} \tilde{\mathbf{p}}_{\mathcal{Q}}$;
    if $e_i < e$ **then**
      $e \leftarrow e_i$;
      $b \leftarrow i$;
  $c \leftarrow (1 + \widehat{\mathbf{X}}_b \mathbf{C}_{:,b})^{-1}$;
  $d \leftarrow c\mathbf{C}^{\mathrm{T}}{}_b \widehat{\mathbf{y}}$;
  $\mathbf{t} \leftarrow c\widehat{\mathbf{X}}_b \mathbf{C}$;
  foreach $\mathcal{Q} \in Q$ **do**
    $\gamma \leftarrow (-c^{-1} + \mathbf{C}^{\mathrm{T}}{}_{b,\mathcal{Q}} \mathbf{U}_{\mathcal{Q},b})^{-1}$;
    $\mathbf{p}_{\mathcal{Q}} \leftarrow \mathbf{p}_{\mathcal{Q}} - d\mathbf{U}_{\mathcal{Q},b} - \gamma \mathbf{U}_{\mathcal{Q},b}(\mathbf{U}^{\mathrm{T}}{}_{b,\mathcal{Q}}(\mathbf{a}_{\mathcal{Q}} - d\mathbf{C}_{\mathcal{Q},b}))$;
    $\mathbf{U}_{\mathcal{Q}} \leftarrow \mathbf{U}_{\mathcal{Q}} - \mathbf{U}_{\mathcal{Q},b}\mathbf{t} - \gamma \mathbf{U}_{\mathcal{Q},b}(\mathbf{U}^{\mathrm{T}}{}_{b,\mathcal{Q}}(\mathbf{C}_{\mathcal{Q}} - \mathbf{C}_{\mathcal{Q},b}\mathbf{t}))$;
  $\mathbf{a} \leftarrow \mathbf{a} - d\mathbf{C}_{:,b}$;
  $\mathbf{C} \leftarrow \mathbf{C} - \mathbf{C}_{:,b}\mathbf{t}$;
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$;
$\mathbf{w} \leftarrow \widehat{\mathbf{X}}_{\mathcal{S}}\mathbf{a}$;

Next, we go through the actual feature selection algorithm whose pseudo code is presented in Algorithm 1. Let us first define the following quasi-diagonal matrix:

$$\mathbf{Q} = \begin{pmatrix} (\mathbf{G}_{\mathcal{Q}_1,\mathcal{Q}_1})^{-1} & & \\ & \ddots & \\ & & (\mathbf{G}_{\mathcal{Q}_{|Q|},\mathcal{Q}_{|Q|}})^{-1} \end{pmatrix}.$$

In order to take advantage of the computational short-cuts, the feature selection algorithm maintains the current set of selected features $\mathcal{S} \subseteq \{1, \ldots, n\}$, the vectors $\mathbf{a}, \mathbf{p} \in \mathbb{R}^m$, and the matrices $\mathbf{C}, \mathbf{U} \in \mathbb{R}^{m \times n}$ whose values are defined as

$$\begin{aligned} \mathbf{a} &= \mathbf{G}\widehat{\mathbf{y}}, \\ \mathbf{C} &= \mathbf{G}\widehat{\mathbf{X}}^{\mathrm{T}}, \\ \mathbf{U} &= \mathbf{Q}\mathbf{G}\widehat{\mathbf{X}}^{\mathrm{T}}, \\ \mathbf{p} &= \mathbf{Q}\mathbf{G}\widehat{\mathbf{y}}. \end{aligned}$$

In the initialization phase of the greedy RankRLS algorithm the set of selected features is empty, and hence the values of $\mathbf{a}$, $\mathbf{C}$, $\mathbf{U}$, and $\mathbf{p}$ are initialized to $\lambda^{-1}\widehat{\mathbf{y}}$, $\lambda^{-1}\widehat{\mathbf{X}}^{\mathrm{T}}$, $\widehat{\mathbf{X}}^{\mathrm{T}}$, and $\widehat{\mathbf{y}}$, respectively. The computational complexity of the initialization phase is dominated by the $O(mn)$ time required for storing the matrices $\mathbf{C}$, $\mathbf{U}$, and $\widehat{\mathbf{X}}$ in memory. Thus, the initialization phase is no more complex than one pass through the training data.

Let us now consider the computation of the LQO performance for the modified feature set $\mathcal{S} \cup \{i\}$, where i is the index of the feature to be added. Recall that the hold-out prediction for the examples that are associated with query $\mathcal{Q}$ can be computed from $\mathbf{p}_{\mathcal{Q}} = (\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1}\mathbf{a}_{\mathcal{Q}}$, where $\mathbf{a} = \mathbf{G}\widehat{\mathbf{y}}$. However, since a new feature is temporarily added into the set of selected features, we must use the matrix

$$\widetilde{\mathbf{G}} = ((\widehat{\mathbf{X}}_{\mathcal{S}})^{\mathrm{T}}\widehat{\mathbf{X}}_{\mathcal{S}} + (\widehat{\mathbf{X}}_i)^{\mathrm{T}}\widehat{\mathbf{X}}_i + \lambda\mathbf{I})^{-1}$$

in place of $\mathbf{G}$. Due to the well-known Sherman-Morrison-Woodbury (SMW) formula, the matrix $\widetilde{\mathbf{G}}$ can be rewritten as

$$\begin{aligned} \widetilde{\mathbf{G}} &= \mathbf{G} - \mathbf{G}(\widehat{\mathbf{X}}_i)^{\mathrm{T}}(1 + \widehat{\mathbf{X}}_i\mathbf{G}(\widehat{\mathbf{X}}_i)^{\mathrm{T}})^{-1}\widehat{\mathbf{X}}_i\mathbf{G} \\ &= \mathbf{G} - c\mathbf{C}_{:,i}\mathbf{C}^{\mathrm{T}}{}_i, \end{aligned}$$

where

$$c = (1 + \widehat{\mathbf{X}}_i\mathbf{C}_{:,i})^{-1}.$$

Accordingly, the updated vector of dual variables $\tilde{\mathbf{a}}$ can be written as

$$\begin{aligned} \tilde{\mathbf{a}} &= \widetilde{\mathbf{G}}\widehat{\mathbf{y}} \\ &= (\mathbf{G} - c\mathbf{C}_{:,i}\mathbf{C}^{\mathrm{T}}{}_i)\widehat{\mathbf{y}} \\ &= \mathbf{a} - d\mathbf{C}_{:,i}, \end{aligned}$$

where

$$d = c\mathbf{C}^{\mathrm{T}}{}_i\widehat{\mathbf{y}}.$$

Now, concerning $(\widetilde{\mathbf{G}}_{\mathcal{Q},\mathcal{Q}})^{-1}$, we have

$$\begin{aligned} (\widetilde{\mathbf{G}}_{\mathcal{Q},\mathcal{Q}})^{-1} &= ((\mathbf{G} - c\mathbf{C}_{:,i}\mathbf{C}^{\mathrm{T}}{}_i)_{\mathcal{Q},\mathcal{Q}})^{-1} \\ &= (\mathbf{G}_{\mathcal{Q},\mathcal{Q}} - c\mathbf{C}_{\mathcal{Q},i}\mathbf{C}^{\mathrm{T}}{}_{i,\mathcal{Q}})^{-1} \\ &= (\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1} \\ &\quad -\gamma(\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1}\mathbf{C}_{\mathcal{Q},i}\mathbf{C}^{\mathrm{T}}{}_{i,\mathcal{Q}}(\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1} \\ &= (\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1} - \gamma\mathbf{U}_{\mathcal{Q},i}\mathbf{U}^{\mathrm{T}}{}_{i,\mathcal{Q}}, \end{aligned}$$

where

$$\gamma = (-c^{-1} + \mathbf{C}^{\mathrm{T}}{}_{i,\mathcal{Q}}\mathbf{U}_{\mathcal{Q},i})^{-1}$$

and the equality between the second and third rows are again due to the SMW formula. Finally, we can compute the hold-out predictions $\tilde{\mathbf{p}}_{\mathcal{Q}}$ for the updated feature set as

$$\begin{aligned} \tilde{\mathbf{p}}_{\mathcal{Q}} &= (\widetilde{\mathbf{G}}_{\mathcal{Q},\mathcal{Q}})^{-1}\tilde{\mathbf{a}}_{\mathcal{Q}} \\ &= (\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1}\tilde{\mathbf{a}}_{\mathcal{Q}} - \gamma\mathbf{U}_{\mathcal{Q},i}(\mathbf{U}^{\mathrm{T}}{}_{i,\mathcal{Q}}\tilde{\mathbf{a}}_{\mathcal{Q}}) \\ &= \mathbf{p}_{\mathcal{Q}} - d\mathbf{U}_{\mathcal{Q},i} - \gamma\mathbf{U}_{\mathcal{Q},i}(\mathbf{U}^{\mathrm{T}}{}_{i,\mathcal{Q}}(\mathbf{a}_{\mathcal{Q}} - d\mathbf{C}_{\mathcal{Q},i})). \end{aligned}$$

The calculation of the last row requires only $O(|\mathcal{Q}|)$ time provided that we have all the required caches available. Since the sizes of the query index sets sum up to m, the overall complexity of LQO-CV for the updated feature set becomes $O(m)$. Further, as the greedy forward selection approach tests each of the order of n unselected features before the best of them is added into the set of selected features, the complexity of the selection step is $O(mn)$.

What is still left in our consideration is the phase in which the caches are updated after an new feature is added into the set of selected features. The vectors $\mathbf{a}$ and $\mathbf{p}$ are updated in the same way as they were temporarily updated in the LQO-CV computations. The update processes of the matrices $\mathbf{U}$ and $\mathbf{C}$ are analogous to those of the vectors $\mathbf{p}$ and $\mathbf{a}$ except that the matrix $\mathbf{C}$ is used in place of the vector $\mathbf{a}$ and the vector

$$\mathbf{t} = c\widehat{\mathbf{X}}_b\mathbf{C},$$

where b is the index of the selected feature, is used in place of the constant d. The computational time required for updating $\mathbf{U}$ and $\mathbf{C}$ is $O(mn)$, that is, updating the caches after the selection step is not more complex than the selection step itself.

Putting everything together, the overall computational complexity of greedy RankRLS is $O(kmn)$, where k is the number of features the algorithm selects until it stops. This is because the algorithm performs k iterations during which it adds one new feature to the set of selected features and each iteration requires $O(mn)$ time as shown above. The space complexity of the algorithm is $O(mn)$ which is dominated by keeping the matrices $\mathbf{C}$, $\mathbf{U}$, and $\widehat{\mathbf{X}}$ in memory.

4. EXPERIMENTS

We perform experiments on the publicly available LETOR benchmark data set (version 4.0) for learning to rank for information retrieval [1] [21]. We run experiments on two data sets, MQ2007 and MQ2008. MQ2007 consists of 69623 examples divided into 1700 queries, and MQ2800 contains 15211 examples divided into 800 queries. In both data sets the examples have the same 46 high-level features.

We follow the experimental setup proposed by the authors of LETOR. All results are averages from 5-fold cross-validation, where on each round 3 folds are used for training, 1 for parameter selection and 1 for testing. We use the exact splits provided in the data sets. Mean Average Precision (MAP) is used when selecting parameters. In addition to average precision, we measure Normalized Discountive Cumulative Gain (NDCG) when calculating test performance. In the results we present MAP, P@10, mean NDCG, and NDCG@10 values.

We compare greedy RankRLS to RankRLS and RankSVM, which are trained on all the features. For greedy RankRLS, we choose via grid search both the number of selected features and value of regularization parameter, such that lead to best MAP performance on the validation fold. For normal RankRLS, which is trained on all the features, only the value of the regularization parameter needs to be tuned. RankRLS and greedy RankRLS are implemented as part of the RLScore open source machine learning framework [2]. The RankSVM results are taken directly from the baselines section of the LETOR distribution website. The experimental setup for the RankSVM runs, as described by LETOR authors, is the same as outlined here, the used implementation was the SVM^{rank} of Joachims[3] [9]. We also plot performance curves as a function of the number of selected features on the validation sets, and examine the feature sets selected on different folds.

Tables 1 and 2 contains the selected features on the MQ2007 and MQ2008 data sets, respectively. Where more than 10 features was selected, we present only the first 10. On two of the folds of MQ2007, the optimal number of features are 11 and 12, on three of the folds almost all of the features are chosen. On MQ2008 relatively few features were chosen on all of the folds, on two of the folds the best validation performance was reached with only one feature. There are differences in the feature sets selected in the different rounds of cross-validation, but one thing remains constant. On both data sets, and on each cross-validation round, the feature selected first is feature number 39, "LMIR.DIR of whole document". The feature is a language model based feature which corresponds to a posteriori estimate of the likelihood of the query given the whole document, where a Dirichlet prior over the documents is used [26]. Based on our results this feature seems to be very useful for ranking, since as it turns out models using only it can in some cases be competitive with models trained on all the features.

In Figures 1, 2, 3, and 4 are the average MAP and mean NDCG performances over the validation folds, plotted for different regularization parameter values. We note that the results are quite unstable, suggesting that reliable selection of the regularization parameter and number of selected features remains a challenging problem. On MQ2007 the performance increases with the number of selected features. This is why in three of the folds the selection strategy used in our study lead to selecting almost all of them. However, on MQ2008 best validation performances are reached with relatively few features, after which the performance decreases. On MQ2007 close to optimal validation results can be reached already with around 15 features. This suggests that perhaps a multi-objective criterion should be used in parameter selection, which in addition to favoring high validation performance would also penalize models that use too many features.

In Tables 3, 4, 5, and 6, are the test results for MQ2007, and in Tables 7, 8, 9, and 10 are the test results for MQ2008. Overall, the results for greedy RankRLS, RankRLS and RankSVM are very close to each other, even on fold-by-fold basis. The results further verify the earlier results in [16,17], which suggest that RankRLS and RankSVM optimization often lead to very similar results. Further, the results show that at least on this data, sparse models learned using greedy forward selection are competitive with models learned using all the features.

5. DISCUSSION AND FUTURE WORK

The greedy RankRLS implementation presented in this paper is computationally feasible when dealing with data sets, such as LETOR, in which the overall number of available features is not very large. However, the situation is different if the data points are represented, for example, as

[1] `http://research.microsoft.com/en-us/um/beijing/projects/letor/`

[2] `http://www.tucs.fi/rlscore`

[3] `http://www.cs.cornell.edu/People/tj/svm_light/svm_rank.html`

Table 1: Selected features on MQ2007.

Model	fold1	fold2	fold3	fold4	fold5
λ	2^8	2^6	2^9	2^8	2^7
k	11	40	46	44	12
selected 1	39	39	39	39	39
selected 2	19	32	27	28	25
selected 3	25	19	23	45	19
selected 4	23	26	19	23	43
selected 5	32	23	13	43	23
selected 6	16	16	18	33	29
selected 7	43	5	42	13	22
selected 8	22	33	33	18	18
selected 9	5	18	16	22	5
selected 10	33	3	5	15	16

Table 2: Selected features on MQ2008

Model	fold1	fold2	fold3	fold4	fold5
λ	2^0	2^{10}	2^3	2^6	2^0
k	1	4	7	4	1
selected 1	39	39	39	39	39
selected 2		23	29	29	
selected 3		37	25	25	
selected 4		32	23	23	
selected 5			46		
selected 6			37		
selected 7			19		

raw text documents, and the words or their composites occurring in the documents form the set of available features. In this case, there is the drawback that $m \times n$-dimensional dense matrices has to be maintained in memory, while the data are stored in a sparse matrix of the same size having only a few nonzero entries. This is, because each document has nonzero values only for a small subset of features. In addition to the feasibility problems with memory, the $O(mn)$ time required per iteration may be too expensive in practise. Fortunately, it is possible to design such variations of greedy RankRLS that are better suited for this type of data.

First, we can avoid storing the dense $m \times n$-matrices by spending more computational resources. This is possible with a variation whose time complexity is $O(k^2mn)$. As an additional modification, we can reduce the time spent in each iteration by selecting the new feature from a random subset of the available features, resulting to a time complexity $O(k^2m\kappa)$, where κ is the size of random subsets. This type of idea is used, for example, for selecting the basis vectors for Gaussian process regressors by [22]. Finally, we can take advantage of the sparsity of the data matrix in reducing the time complexity down to $O(k^2\overline{m}\kappa)$, where $\overline{m}$ is the average number of training examples for which the features have nonzero values, if we use so-called back-fitting variation of our algorithm instead of performing pre-fitting as our current implementation does. For descriptions of the terms back-fitting and pre-fitting, we refer to [23]. The detailed

Table 3: Map results on MQ2007

Fold	GRankRLS	RankRLS	RankSVM
1	0.4859	0.4912	0.4894
2	0.4571	0.4573	0.4573
3	0.4655	0.4655	0.4676
4	0.4423	0.4425	0.4401
5	0.4709	0.4687	0.4680
avg	0.4643	0.4650	0.4645

Table 4: P@10 results on MQ2007

Fold	GRankRLS	RankRLS	RankSVM
1	0.3958	0.3997	0.4036
2	0.3858	0.3855	0.3932
3	0.3684	0.3684	0.3699
4	0.3670	0.3673	0.3652
5	0.3808	0.3811	0.3847
avg	0.3796	0.3804	0.3833

Table 5: MeanNDCG results on MQ2007

Fold	GRankRLS	RankRLS	RankSVM
1	0.5228	0.5281	0.5278
2	0.4840	0.4841	0.4810
3	0.5056	0.5056	0.5042
4	0.4757	0.4754	0.4699
5	0.5033	0.5003	0.5003
avg	0.4983	0.4987	0.4966

Table 6: NDCG@10 results on MQ2007

Fold	GRankRLS	RankRLS	RankSVM
1	0.4735	0.4784	0.4818
2	0.4247	0.4246	0.4266
3	0.4466	0.4466	0.4461
4	0.4221	0.4221	0.4163
5	0.4487	0.4460	0.4485
avg	0.4431	0.4435	0.4439

Table 7: Map results on MQ2008

Fold	GRankRLS	RankRLS	RankSVM
1	0.4311	0.4524	0.4502
2	0.4239	0.4300	0.4213
3	0.4582	0.4542	0.4529
4	0.5283	0.5225	0.5284
5	0.5183	0.5006	0.4950
avg	0.4720	0.4719	0.4696

Table 8: P@10 results on MQ2008

Fold	GRankRLS	RankRLS	RankSVM
1	0.2333	0.2391	0.2423
2	0.2178	0.2217	0.2229
3	0.2363	0.2325	0.2357
4	0.2975	0.2949	0.2981
5	0.2484	0.2503	0.2465
avg	0.2467	0.2477	0.2491

Table 9: MeanNDCG results on MQ2008

Fold	GRankRLS	RankRLS	RankSVM
1	0.4454	0.4633	0.4577
2	0.4186	0.4269	0.4296
3	0.4787	0.4741	0.4686
4	0.5403	0.5407	0.5442
5	0.5369	0.5138	0.5159
avg	0.4840	0.4838	0.4832

Table 10: NDCG@10 results on MQ2008

Fold	GRankRLS	RankRLS	RankSVM
1	0.1920	0.2145	0.2117
2	0.1585	0.1669	0.1738
3	0.2558	0.2489	0.2494
4	0.2940	0.2874	0.2892
5	0.2254	0.2165	0.2155
avg	0.2251	0.2268	0.2279

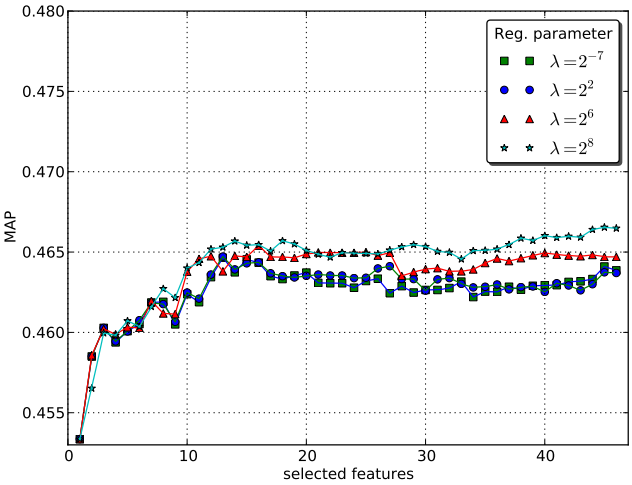

Figure 1: Average MAP on validation sets for MQ2007.

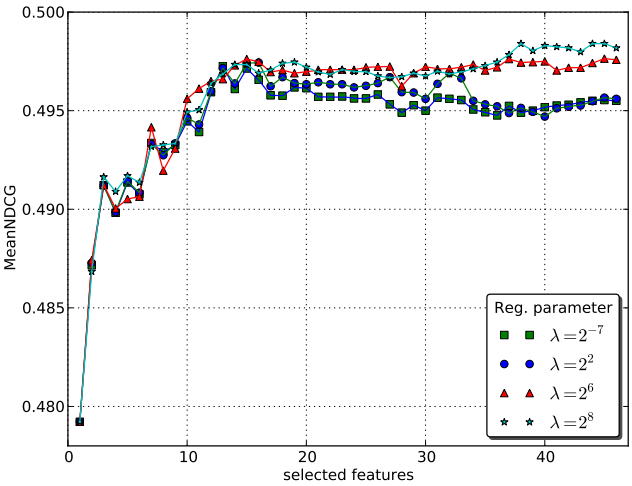

Figure 2: Average Mean NDCG on validation sets for MQ2007.

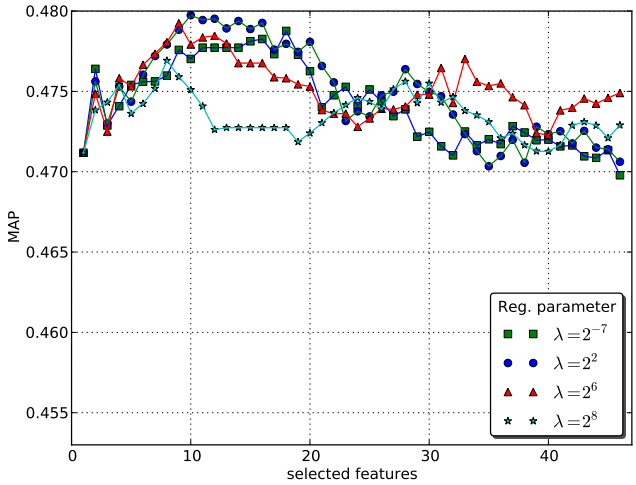

Figure 3: Average MAP on validation sets for MQ2008.

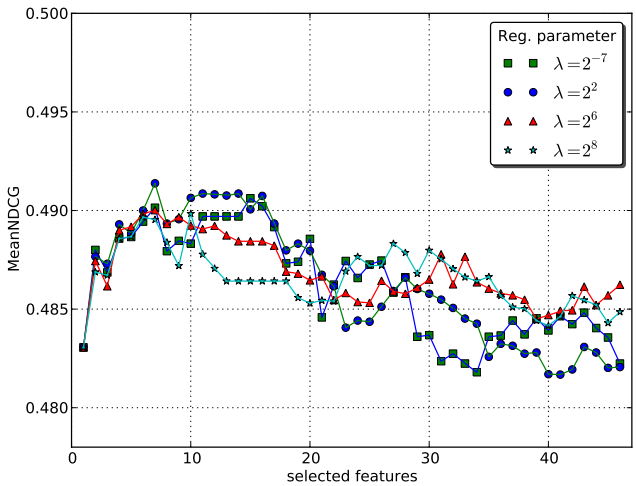

Figure 4: Average Mean NDCG on validation sets for MQ2008.

descriptions of these variations are left for future work.

The greedy forward selection approach can sometimes suffer from the so-called nesting effect, meaning that the best subset of size k, for example, may not necessarily cover the features included in the best subset of size $k-1$. Floating search methods (see e.g. [13, 20, 27]), which are able to discard features selected in previous iterations, have been proposed as a means to deal with this issue. Replacing the greedy search strategy with a floating search would be a fairly straightforward extension to the presented algorithm.

6. CONCLUSION

To conclude, we propose a computationally efficient method for learning sparse predictors for ranking tasks. The method uses on greedy forward selection as a search strategy and leave-query-out cross-validation as a selection criterion. The computational complexity of the method is linear in the number of training examples, in the overall number of features, and in the number of features to be selected. Thus, the method is computationally highly efficient despite the fact that the method optimizes a pairwise ranking loss function and uses a complex cross-validation criterion. Empirical evaluation with the LETOR benchmark data set demonstrates the soundness of the proposed approach.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments. This work has been supported by the Academy of Finland.

7. REFERENCES

[1] S. An, W. Liu, and S. Venkatesh. Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression. *Pattern Recognition*, 40(8):2154–2162, 2007.

[2] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In L. D. Raedt and S. Wrobel, editors, *Proceedings of the 22nd*

international conference on Machine learning (ICML 2005), pages 89–96. ACM, 2005.

[3] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In Z. Ghahramani, editor, *Proceedings of the 24th international conference on Machine learning (ICML 2007)*, pages 129–136. ACM, 2007.

[4] O. Chapelle, Q. Le, and A. Smola. Large margin optimization of ranking measures. In *NIPS Workshop: Machine Learning for Web Search*, 2007.

[5] X. Geng, T.-Y. Liu, T. Qin, and H. Li. Feature selection for ranking. In C. L. Clarke, N. Fuhr, N. Kando, W. Kraaij, and A. P. de Vries, editors, *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2007)*, pages 407–414. ACM, 2007.

[6] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[7] H. V. Henderson and S. R. Searle. On deriving the inverse of a sum of matrices. *SIAM Review*, 23(1):53–60, 1981.

[8] T. Joachims. Optimizing search engines using clickthrough data. In D. Hand, D. Keim, and R. Ng, editors, *Proceedings of the 8th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2002)*, pages 133–142. ACM, 2002.

[9] T. Joachims. Training linear SVMs in linear time. In T. Eliassi-Rad, L. H. Ungar, M. Craven, and D. Gunopulos, editors, *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2006)*, pages 217–226. ACM, 2006.

[10] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In W. W. Cohen and H. Hirsch, editors, *Proceedings of the Eleventh International Conference on Machine Learning (ICML 1994)*, pages 121–129, San Fransisco, CA, 1994. Morgan Kaufmann Publishers.

[11] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[12] D. A. Metzler. Automatic feature selection in the markov random field model for information retrieval. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management (CIKM'07)*, pages 253–262. ACM, 2007.

[13] T. Pahikkala, A. Airola, and T. Salakoski. Feature selection for regularized least-squares: New computational short-cuts and fast algorithmic implementations. In *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2010. To appear.

[14] T. Pahikkala, A. Airola, and T. Salakoski. Linear time feature selection for regularized least-squares, 2010. Preprint http://arxiv.org/abs/1003.3570.

[15] T. Pahikkala, J. Boberg, and T. Salakoski. Fast n-fold cross-validation for regularized least-squares. In T. Honkela, T. Raiko, J. Kortela, and H. Valpola, editors, *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)*, pages 83–90. Otamedia, 2006.

[16] T. Pahikkala, E. Tsivtsivadze, A. Airola, J. Boberg, and J. Järvinen. An efficient algorithm for learning to rank from preference graphs. *Machine Learning*, 75(1):129–165, 2009.

[17] T. Pahikkala, E. Tsivtsivadze, A. Airola, J. Boberg, and T. Salakoski. Learning to rank with pairwise regularized least-squares. In T. Joachims, H. Li, T.-Y. Liu, and C. Zhai, editors, *SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, pages 27–33, 2007.

[18] T. Pahikkala, W. Waegeman, A. Airola, T. Salakoski, and B. De Baets. Conditional ranking on relational data. In *ECML PKDD '10: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, 2010. To appear.

[19] F. Pan, T. Converse, D. Ahn, F. Salvetti, and G. Donato. Feature selection for ranking using boosted trees. In D. W.-L. Cheung, I.-Y. Song, W. W. Chu, X. Hu, and J. J. Lin, editors, *Proceeding of the 18th ACM conference on Information and knowledge management (CIKM'09)*, pages 2025–2028. ACM, 2009.

[20] P. Pudil, J. Novovičová, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119–1125, 1994.

[21] T. Qin, T.-Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 2010.

[22] A. J. Smola and P. Bartlett. Sparse greedy gaussian process regression. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 619–625. MIT Press, 2001.

[23] P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48(1–3):165–187, 2002.

[24] H. Yu, J. Oh, and W.-S. Han. Efficient feature weighting methods for ranking. In D. W.-L. Cheung, I.-Y. Song, W. W. Chu, X. Hu, and J. J. Lin, editors, *Proceeding of the 18th ACM conference on Information and knowledge management (CIKM'09)*, pages 1157–1166. ACM, 2009.

[25] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In W. Kraaij, A. P. de Vries, C. L. A. Clarke, N. Fuhr, and N. Kando, editors, *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2007)*, pages 271–278. ACM, 2007.

[26] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2001)*, pages 334–342. ACM, 2001.

[27] T. Zhang. Adaptive forward-backward greedy algorithm for sparse learning with linear models. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1921–1928. MIT Press, 2009.