Speeding up Greedy Forward Selection for Regularized Least-Squares

Tapio Pahikkala, Antti Airola, and Tapio Salakoski
Department of Information Technology
University of Turku and Turku Centre for Computer Science,
Turku, Finland,
e-mail: firstname.lastname@utu.fi.

***Abstract*—We propose a novel algorithm for greedy forward feature selection for regularized least-squares (RLS) regression and classification, also known as the least-squares support vector machine or ridge regression. The algorithm, which we call greedy RLS, starts from the empty feature set, and on each iteration adds the feature whose addition provides the best leave-one-out cross-validation performance. Our method is considerably faster than the previously proposed ones, since its time complexity is linear in the number of training examples, the number of features in the original data set, and the desired size of the set of selected features. Therefore, as a side effect we obtain a new training algorithm for learning sparse linear RLS predictors which can be used for large scale learning. This speed is possible due to matrix calculus based short-cuts for leave-one-out and feature addition. We experimentally demonstrate the scalability of our algorithm compared to previously proposed implementations.**

I. Introduction

In this paper, we propose a novel algorithm for greedy forward feature selection for regularized least-squares (RLS) regression and classification. RLS (see e.g [1]), also known as the least-squares support vector machine (LS-SVM) [2] and ridge regression [3], is a state-of-the art machine learning method suitable both for regression and classification, and it has also been extended for ranking [4], [5].

In the literature, there are many approaches for the task of feature selection (see e.g. [6]). We consider the wrapper type of feature selection [7] in which the feature selection and training of the RLS predictor are done simultaneously.

Performing feature selection for linear RLS results in a linear predictor which is sparse, that is, the predictor has nonzero parameters only for the selected features. Sparse predictors are beneficial for many reasons. For example, when constructing an instrument for diagnosing a disease according to a medical sample, sparse predictors can be deployed in the instrument more easily than dense ones, because they require less memory for storing and less information from the sample in order to perform the diagnosis. Another benefit of sparse representations is in their interpretability. If the predictor consists of only a small number of nonzero parameters, it makes it easier for a human expert to explain the underlying concept.

The number of possible feature sets grows exponentially with the number of available features. Therefore, the wrapper type of feature selection methods need a search strategy over the power set of features. As a search strategy, we use the greedy forward selection that adds one feature at a time to the set of selected features but no features are removed from the set at any stage. In addition to the search strategy, a heuristic for assessing the goodness of the feature subsets is required. Measuring the prediction performance on the training set is known to be unreliable. Therefore, as suggested by [8], we use the leave-one-out cross-validation (LOO) approach, where each example in turn is left out of the training set and used for testing, as a search heuristic. The merits of using greedy forward selection as a search strategy and LOO criterion as a heuristic when doing feature selection for RLS have been empirically shown by [9], [10].

An important property of the RLS algorithm is that it has a closed form solution, which can be fully expressed in terms of matrix operations. This allows developing efficient computational shortcuts for the method, since small changes in the training data matrix correspond to low-rank changes in the learned predictor. Especially it makes possible the development of efficient cross-validation algorithms. An updated predictor, corresponding to a one learned from a training set from which one example has been removed, can be obtained via a well-known computationally efficient short-cut (see e.g. [11]) which in turn enables the fast computation of LOO-based performance estimates. Analogously to removing the effects of training examples from learned RLS predictors, the effects of a feature can be added or removed by updating the learned RLS predictors via similar computational short-cuts.

Learning a linear RLS predictor with k features and m training examples requires $O(\min\{k^2m, km^2\})$ time, since the training can be performed either in primal or dual form depending whether $m > k$ or vice versa. Let n be the overall number of features available for selection. Given that the computation of LOO performance requires m retrainings, that the forward selection tests $O(n)$ possibilities in each iteration, and that the forward selection has k iterations, the overall time complexity of the forward selection with LOO criterion becomes $O(\min\{k^3m^2n, k^2m^3n\})$ in case RLS is used as a black-box method.

In machine learning and statistics literature, there have been several studies in which the computational short-cuts for LOO have been used to speed up the evaluation of feature subset quality for RLS (see e.g. [9]). However, the considered approaches are still computationally quite demanding, since the LOO estimate needs to be re-calculated from scratch for each considered subset of features. Recently, [10] introduced a method which uses additional computational short-cuts for efficient updating of the LOO predictions when adding new features to those already selected. The computational cost of the incremental forward selection procedure is only $O(km^2n)$ for the method. The speed improvement is notable especially in cases, where the aim is to select a large number of features but the training set is small. However, the method is still impractical for large training sets due to its quadratic scaling. In [12] we proposed computational short-cuts for speeding up feature selection for RLS with LOO criterion and in [13] we proposed a feature selection algorithm for RankRLS, a RLS-based algorithm for learning to rank.

This paper concerns the construction of a greedy forward selection algorithm for RLS that takes advantage of our previously proposed computational short-cuts. The algorithm, which we call greedy RLS, can be carried out in $O(kmn)$ time, where k is the number of selected features. The method is computationally faster than the previously proposed implementations.

The contributions of this paper can be summarized as follows:

- We present greedy RLS, a linear time feature selection algorithm for RLS that uses LOO as a selection criterion.
- We show that the predictors learned by greedy RLS are exactly equivalent to those obtained via a standard wrapper feature selection approach which uses a greedy selection strategy, LOO selection criterion, and RLS as a black-box method.
- We show both theoretically and empirically that greedy RLS is computationally faster than the previously proposed algorithms that produce equivalent predictors. Namely, it is shown that the computational complexity of greedy RLS is $O(kmn)$ which makes it faster than low-rank update LS-SVM proposed by [10] which requires $O(km^2n)$ time.

II. Regularized Least-Squares

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 $\mathbf{v}$.

Let $\mathbf{X}\in\mathbb{R}^{n\times m}$ be a matrix containing the whole feature representation of the examples in the training set, where n is the total number of features and m is the number of training examples. The i,jth entry of $\mathbf{X}$ contains the value of the ith feature in the jth training example. Moreover, let $\mathbf{y}\in\mathbb{R}^m$ be a vector containing the labels of the training examples. In binary classification, the labels can be restricted to be either -1 or 1, for example, while they can be any real numbers in regression tasks.

In this paper, we consider linear predictors of type

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

where $\mathbf{w}$ is the $|\mathcal{S}|$-dimensional vector representation of the learned predictor and $\mathbf{x}_{\mathcal{S}}$ can be considered as a mapping of the data point x into $|\mathcal{S}|$-dimensional feature space.[1] 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 the prediction phase. The computational complexity of making predictions with (1) and the space complexity of the predictor are both $O(|\mathcal{S}|)$ provided that the feature vector representation $\mathbf{x}_{\mathcal{S}}$ for the data point x is given.

Given training data and a set of feature indices $\mathcal{S}$, we find $\mathbf{w}$ by minimizing the RLS risk. This can be expressed as the following problem:

$$\underset{\mathbf{w}\in\mathbb{R}^{|\mathcal{S}|}}{\operatorname{argmin}}\left\{((\mathbf{w}^{\mathrm{T}}\mathbf{X}_{\mathcal{S}})^{\mathrm{T}}-\mathbf{y})^{\mathrm{T}}((\mathbf{w}^{\mathrm{T}}\mathbf{X}_{\mathcal{S}})^{\mathrm{T}}-\mathbf{y})+\lambda\mathbf{w}^{\mathrm{T}}\mathbf{w}\right\}. \tag{2}$$

The first term in (2), called the empirical risk, measures how well the prediction function fits to the training data. The second term is called the regularizer and it controls the tradeoff between the loss on the training set and the complexity of the prediction function.

A straightforward approach to solve (2) is to set the derivative of the objective function with respect to $\mathbf{w}$ to zero. Then, by solving it with respect to $\mathbf{w}$, we get

$$\mathbf{w} = (\mathbf{X}_{\mathcal{S}}(\mathbf{X}_{\mathcal{S}})^{\mathrm{T}}+\lambda\mathbf{I})^{-1}\mathbf{X}_{\mathcal{S}}\mathbf{y}, \tag{3}$$

where $\mathbf{I}$ is the identity matrix. We note (see e.g. [14]) that an equivalent result can be obtained from

$$\mathbf{w} = \mathbf{X}_{\mathcal{S}}((\mathbf{X}_{\mathcal{S}})^{\mathrm{T}}\mathbf{X}_{\mathcal{S}}+\lambda\mathbf{I})^{-1}\mathbf{y}. \tag{4}$$

If the size of the set $\mathcal{S}$ of currently selected features is smaller than the number of training examples m, it is computationally beneficial to use the form (3) while using (4) is faster in the opposite case. Namely, the computational complexity of the former is $O(|\mathcal{S}|^3+|\mathcal{S}|^2m)$, while the that

[1] In the literature, the formula of the linear predictors often also contain a bias term. Here, we assume that if such a bias is used, it will be realized by using an extra constant valued feature in the data points.

Algorithm 1: Standard wrapper algorithm for RLS

```
   Input: $\mathbf{X} \in \mathbb{R}^{n \times m}$, $\mathbf{y} \in \mathbb{R}^m$, $k$, $\lambda$
   Output: $\mathcal{S}$, $\mathbf{w}$
1  $\mathcal{S} \leftarrow \emptyset$;
2  while $|\mathcal{S}| < k$ do
3      $e \leftarrow \infty$;
4      $b \leftarrow 0$;
5      foreach $i \in \{1, \dots, n\} \setminus \mathcal{S}$ do
6          $\mathcal{R} \leftarrow \mathcal{S} \cup \{i\}$;
7          $e_i \leftarrow \mathrm{LOO}(\mathbf{X}_\mathcal{R}, \mathbf{y}, \lambda)$;
8          if $e_i < e$ then
9              $e \leftarrow e_i$;
10             $b \leftarrow i$;
11     $\mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$;
12 $\mathbf{w} \leftarrow t(\mathbf{X}_\mathcal{S}, \mathbf{y}, \lambda)$;
```

Figure 1. Standard wrapper algorithm for RLS

of the latter is $O(m^3 + m^2|\mathcal{S}|)$, and hence the complexity of training a predictor is $O(\min\{|\mathcal{S}|^2 m, m^2|\mathcal{S}|\})$.

To support the following considerations, we introduce the dual form of the prediction function and some extra notation. According to [15], the prediction function (1) can be represented in dual form as follows

$$f(\mathbf{x}) = \mathbf{a}^{\mathrm{T}}(\mathbf{X}_\mathcal{S})^{\mathrm{T}}\mathbf{x}_\mathcal{S}.$$

Here $\mathbf{a} \in \mathbb{R}^m$ is the vector of so-called dual variables, which can be obtained from $\mathbf{a} = \mathbf{G}\mathbf{y}$, where $\mathbf{G} = ((\mathbf{X}_\mathcal{S})^{\mathrm{T}}\mathbf{X}_\mathcal{S} + \lambda\mathbf{I})^{-1}$.

Next, we consider a well-known efficient approach for evaluating the LOO performance of a trained RLS predictor (see e.g. [11]). Provided that we have the vector of dual variables $\mathbf{a}$ and the diagonal elements of $\mathbf{G}$ available, the LOO prediction for the jth training example can be obtained in constant number of floating point operations from

$$\mathbf{y}_j - (\mathbf{G}_{j,j})^{-1}\mathbf{a}_j. \tag{5}$$

III. Algorithm Descriptions

A. Wrapper Approach

Here, we consider greedy forward feature selection for RLS with LOO criterion. In this approach, feature sets up to size k are searched and the goodness of a feature subset is measured via LOO classification error. The method is greedy in the sense that it adds one feature at a time to the set of selected features, and no features are removed from the set at any stage. A high level pseudo code of greedy RLS is presented in Fig. 1. In the algorithm description, the outermost loop adds one feature at a time into the set of selected features $\mathcal{S}$ until the size of the set has reached the desired number of selected features k. The inner loop goes through every feature that has not yet been added into the set of selected features and, for each of feature, computes the LOO performance of the RLS predictor trained using the feature and the previously added features. With $\mathrm{LOO}(\mathbf{X}_\mathcal{R}, \mathbf{y}, \lambda)$, we denote the LOO performance obtained with a data matrix $\mathbf{X}_\mathcal{R}$, a label vector $\mathbf{y}$, and a regularization parameter λ, for RLS. In the end of the algorithm description, $t(\mathbf{X}_\mathcal{S}, \mathbf{y}, \lambda)$ denotes the black-box training procedure for RLS which takes a data matrix, a label vector, and a value of the regularization parameter as input and returns a vector representation of the learned predictor $\mathbf{w}$.

Training a linear RLS predictor with k features and m training examples requires $O(\min\{k^2 m, km^2\})$ time. Given that the computation of LOO performance requires m retrainings, that the selection of a new feature is done from a set of size $O(n)$ in each iteration, and that k features are chosen, the overall time complexity of the forward selection with LOO criterion is $O(\min\{k^3 m^2 n, k^2 m^3 n\})$. Thus, the wrapper approach is feasible with small training sets and in cases where the aim is to select only a small number of features. However, at the very least quadratic complexity with respect to both the size of the training set and the number of selected features makes the standard wrapper approach impractical in large scale learning settings. The space complexity of the wrapper approach is $O(nm)$ which is equal to the cost of storing the data matrix $\mathbf{X}$.

An immediate reduction for the above considered computational complexities can be achieved via the short-cut methods for calculating the LOO performance presented in Section II. In machine learning and statistics literature, there have been several studies in which the computational short-cuts for LOO, as well as other types of cross-validation, have been used to speed up the evaluation of feature subset quality for RLS. For the greedy forward selection, the approach proposed by [9] has a computational complexity to $O(k^2 m^2 n + km^3 n)$, since they train RLS in dual form and use the LOO short-cut (5) enabling the calculation of LOO as efficiently as training the predictor itself.

We note that an analogous short-cut for the LOO performance exists also for the primal form of RLS (see e.g. [11]), that is, the LOO performance can be computed for RLS as efficiently as training RLS in the primal. Accordingly, we can remove a factor m corresponding to the LOO computation from the time complexity of the wrapper approach which then becomes $O(\min\{k^3 mn, k^2 m^2 n\})$. However, we are not aware of any studies in which the LOO short-cut for the primal formulation would be used for the greedy forward feature selection for RLS.

Recently, [10] proposed an algorithm implementation which they call low-rank updated LS-SVM. The features selected by the algorithm are equal to those selected by the standard wrapper algorithm and by the method proposed by [9], but it uses certain additional computational short-cuts to speed up the selection process. The overall time complexity of the low-rank updated LS-SVM algorithm is $O(km^2 n)$. The complexity with respect to k is linear which is much better than that of the standard wrapper approach, and hence

Algorithm 2: Greedy RLS

```
Input: $\mathbf{X} \in \mathbb{R}^{n\times m}$, $\mathbf{y} \in \mathbb{R}^{m}$, $k$, $\lambda$
Output: $\mathcal{S}$, $\mathbf{w}$
1  $\mathbf{a} \leftarrow \lambda^{-1}\mathbf{y}$;
2  $\mathbf{d} \leftarrow \lambda^{-1}\mathbf{1}$;
3  $\mathbf{C} \leftarrow \lambda^{-1}\mathbf{X}^{\mathrm{T}}$;
4  $\mathcal{S} \leftarrow \emptyset$;
5  while $|\mathcal{S}| < k$ do
6      $e \leftarrow \infty$;
7      $b \leftarrow 0$;
8      foreach $i \in \{1,\ldots,n\} \setminus \mathcal{S}$ do
9          $\mathbf{u} \leftarrow \mathbf{C}_{:,i}(1 + \mathbf{X}_i\mathbf{C}_{:,i})^{-1}$;
10         $\tilde{\mathbf{a}} \leftarrow \mathbf{a} - \mathbf{u}(\mathbf{X}_i\mathbf{a})$;
11         $e_i \leftarrow 0$;
12         foreach $j \in \{1,\ldots,m\}$ do
13             $\tilde{\mathbf{d}}_j \leftarrow \mathbf{d}_j - \mathbf{u}_j\mathbf{C}_{j,i}$;
14             $p \leftarrow \mathbf{y}_j - (\tilde{\mathbf{d}}_j)^{-1}\tilde{\mathbf{a}}_j$;
15             $e_i \leftarrow e_i + (p - \mathbf{y}_j)^2$;
16         if $e_i < e$ then
17             $e \leftarrow e_i$;
18             $b \leftarrow i$;
19     $\mathbf{u} \leftarrow \mathbf{C}_{:,b}(1 + \mathbf{X}_b\mathbf{C}_{:,b})^{-1}$;
20     $\mathbf{a} \leftarrow \mathbf{a} - \mathbf{u}(\mathbf{X}_b\mathbf{a})$;
21     foreach $j \in \{1,\ldots,m\}$ do
22         $\mathbf{d}_j \leftarrow \mathbf{d}_j - \mathbf{u}_j\mathbf{C}_{j,b}$;
23     $\mathbf{C} \leftarrow \mathbf{C} - \mathbf{u}(\mathbf{X}_b\mathbf{C})$;
24     $\mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$;
25 $\mathbf{w} \leftarrow \mathbf{X}_{\mathcal{S}}\mathbf{a}$;
```

Figure 2. Greedy RLS algorithm proposed by us

the selection of large feature sets is made possible. However, feature selection with large training sets is still infeasible because of the quadratic complexity with respect to m. The space complexity of the low-rank updated LS-SVM algorithm is $O(nm + m^2)$, because it stores the matrices $\mathbf{X}$ and $\mathbf{G}$ requiring $O(nm)$ and $O(m^2)$ space, respectively. Due to the quadratic dependence of m, this space complexity is worse than that of the standard wrapper approach.

B. Greedy Regularized Least-Squares

Here, we present our novel algorithm for greedy forward selection for RLS with LOO criterion. We refer to our algorithm as greedy RLS, since in addition to feature selection point of view, it can also be considered as a greedy algorithm for learning sparse RLS predictors. Pseudo code of greedy RLS is presented in Fig. 2.

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

$$\begin{aligned}\mathbf{a} &= \mathbf{G}\mathbf{y},\\ \mathbf{d} &= \text{diag}(\mathbf{G}),\\ \mathbf{C} &= \mathbf{G}\mathbf{X}^{\mathrm{T}},\end{aligned} \tag{6}$$

where

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

and diag$(\mathbf{G})$ denotes a vector consisting of the diagonal entries of $\mathbf{G}$.

In the initialization phase of the greedy RLS algorithm (lines 1-4 in Algorithm 2) the set of selected features is empty, and hence the values of $\mathbf{a}$, $\mathbf{d}$, and $\mathbf{C}$ are initialized to $\lambda^{-1}\mathbf{y}$, $\lambda^{-1}\mathbf{1}$, and $\lambda^{-1}\mathbf{X}^{\mathrm{T}}$, respectively, where $\mathbf{1} \in \mathbb{R}^m$ is a vector having every entry equal to 1. The time complexity of the initialization phase is dominated by the $O(mn)$ time required for initializing $\mathbf{C}$. Thus, the initialization phase is no more complex than one pass through the features.

We now consider finding the optimal update direction given a set of $n - |\mathcal{S}|$ of available directions, that is, the direction having the lowest LOO error. Computing the LOO performance with formula (5) for the modified feature set $\mathcal{S} \cup \{i\}$, where i is the index of the feature to be tested, requires the vectors $\tilde{\mathbf{a}} = \widetilde{\mathbf{G}}\mathbf{y}$ and $\tilde{\mathbf{d}} = \text{diag}(\widetilde{\mathbf{G}})$, where

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

Let us define a vector

$$\mathbf{u} = \mathbf{C}_{:,i}(1 + \mathbf{X}_i\mathbf{C}_{:,i})^{-1} \tag{7}$$

which is computable in $O(m)$ time provided that $\mathbf{C}$ is available. Then, the matrix $\widetilde{\mathbf{G}}$ can be rewritten as

$$\begin{aligned}\widetilde{\mathbf{G}} &= \mathbf{G} - \mathbf{G}(\mathbf{X}_i)^{\mathrm{T}}(1 + \mathbf{X}_i\mathbf{G}(\mathbf{X}_i)^{\mathrm{T}})^{-1}\mathbf{X}_i\mathbf{G}\\ &= \mathbf{G} - \mathbf{u}\mathbf{X}_i\mathbf{G}\end{aligned} \tag{8}$$

where the first equality is due to the well-known Sherman-Morrison-Woodbury formula. Accordingly, the vector $\tilde{\mathbf{a}}$ can be written as

$$\begin{aligned}\tilde{\mathbf{a}} &= \widetilde{\mathbf{G}}\mathbf{y}\\ &= (\mathbf{G} - \mathbf{u}\mathbf{X}_i\mathbf{G})\mathbf{y}\\ &= \mathbf{a} - \mathbf{u}(\mathbf{X}_i\mathbf{a})\end{aligned} \tag{9}$$

in which the lowermost expression is also computable in $O(m)$ time. Further, the entries of $\tilde{\mathbf{d}}$ can be computed from

$$\begin{aligned}\tilde{\mathbf{d}}_j &= \widetilde{\mathbf{G}}_{j,j}\\ &= (\mathbf{G} - \mathbf{u}\mathbf{X}_i\mathbf{G})_{j,j}\\ &= (\mathbf{G} - \mathbf{u}(\mathbf{C}_{:,i})^{\mathrm{T}})_{j,j}\\ &= \mathbf{d}_j - \mathbf{u}_j\mathbf{C}_{j,i}\end{aligned} \tag{10}$$

in a constant time, the overall time needed for computing $\tilde{\mathbf{d}}$ again becoming $O(m)$. Thus, provided that we have all the necessary caches available, evaluating each feature requires $O(m)$ time, and hence one pass through the whole set of n features needs $O(mn)$ floating point operations.

Thus, provided that we have all the necessary caches available, evaluating each feature requires $O(m)$ time, and hence one pass through the whole set of n features needs

$O(mn)$ time. But we still have to ensure that the caches can be initialized and updated efficiently enough.

When a new feature is added into the set of selected features, the vector $\mathbf{a}$ is updated according to (9) and the vector $\mathbf{d}$ according to (10). Putting together (6), (7), and (8), the cache matrix $\mathbf{C}$ can be updated via

$$\mathbf{C} - \mathbf{u}(\mathbf{X}_b\mathbf{C}),$$

where b is the index of the feature having the best LOO value, requiring $O(mn)$ time. This is equally expensive as the above introduced fast approach for trying each feature at a time using LOO as a selection criterion.

Finally, if we are to select altogether k features, the overall time complexity of greedy RLS becomes $O(kmn)$. The space complexity is dominated by the matrices $\mathbf{X}$ and $\mathbf{C}$ which both require $O(mn)$ space.

IV. Experimental results

We recall that our greedy RLS algorithm leads to results that are equivalent to those of the algorithms proposed by [9] and by [10], while being computationally much more efficient. The aforementioned authors have in their work shown that the greedy forward selection with LOO criterion approach compares favorably to other commonly used feature selection techniques. Hence, in our experiments, we focus on the scalability of our algorithm implementation and compare it with the best previously proposed algorithm, namely to the low-rank updated LS-SVM introduced by [10]. We will not compare greedy RLS to the implementation proposed by [9], because it was by [10] already shown to be slower than the low-rank updated LS-SVM algorithm.

To test the scalability of the method, we use randomly generated data from two normal distributions with 1000 features of which 50 are selected. The number of examples is varied. In the first experiment we vary the number of training examples between 500 and 5000, and provide a comparison to the low-rank updated LS-SVM method [10]. In the second experiment the training set size is varied between 1000 and 50000. We do not consider the baseline method in the second experiment, as it does not scale up to the considered training set sizes. The runtime experiments were run on a modern desktop computer with 2.4 GHz Intel Core 2 Duo E6600 processor, 8 GB of main memory, and 64-bit Ubuntu Linux 9.10 operating system.

We note that the running times of these two methods are not affected by the choice of the regularization parameter, or the distribution of the features or the class labels. This is in contrast to iterative optimization techniques commonly used to train, for example, support vector machines [16]. Thus we can draw general conclusions about the scalability of the methods from experiments with a fixed value for the regularization parameter, and synthetic data.

The runtime experiments are presented in Fig. 3. The results are consistent with the algorithmic complexity analysis of the methods. The the low-rank updated LS-SVM method of [10] shows quadratic scaling with respect to the number of training examples, while the proposed method scales linearly. Moreover, the figure shows the running times of greedy RLS for up to 50000 training examples, in which case selecting 50 features out of 1000 took a bit less than twelve minutes.

V. Conclusion

We propose greedy regularized least-squares, a novel training algorithm for sparse linear predictors. The predictors learned by the algorithm are equivalent with those obtained by performing a greedy forward feature selection with leave-one-out (LOO) criterion for regularized least-squares (RLS), also known as the least-squares support vector machine or ridge regression. That is, the algorithm works like a wrapper type of feature selection method which starts from the empty feature set, and on each iteration adds the feature whose addition provides the best LOO performance. Training a predictor with greedy RLS requires $O(kmn)$ time, where k is the number of non-zero entries in the predictor, m is the number of training examples, and n is the original number of features in the data. This is in contrast to the computational complexity $O(\min\{k^3m^2n, k^2m^3n\})$ of using the standard wrapper method with LOO selection criterion in case RLS is used as a black-box method, and the complexity $O(km^2n)$ of the method proposed by [10], which is the most efficient of the previously proposed speed-ups. Hereby, greedy RLS is computationally more efficient than the previously known feature selection methods for RLS. We demonstrate experimentally the computational efficiency of greedy RLS compared to the best previously proposed implementation.

We have made freely available a software package called RLScore out of our previously proposed RLS based machine learning algorithms[2]. An implementation of the greedy RLS algorithm is also available as a part of this software.

The study presented in this paper opens several directions for future research. For example, greedy RLS can quite straightforwardly be generalized to use different types of cross-validation criterion, such as N-fold or repeated N-fold. These are motivated by their smaller variance compared to the leave-one-out and they have also been shown to have better asymptotic convergence properties for feature subset selection for ordinary least-squares [17]. This generalization can be achieved by using the short-cut methods developed by us [18] and by [19]. In [13] we investigated this approach for the RankRLS algorithm by developing a selection criteria based on leave-query-out cross-validation.

Acknowledgments

This work has been supported by the Academy of Finland.

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

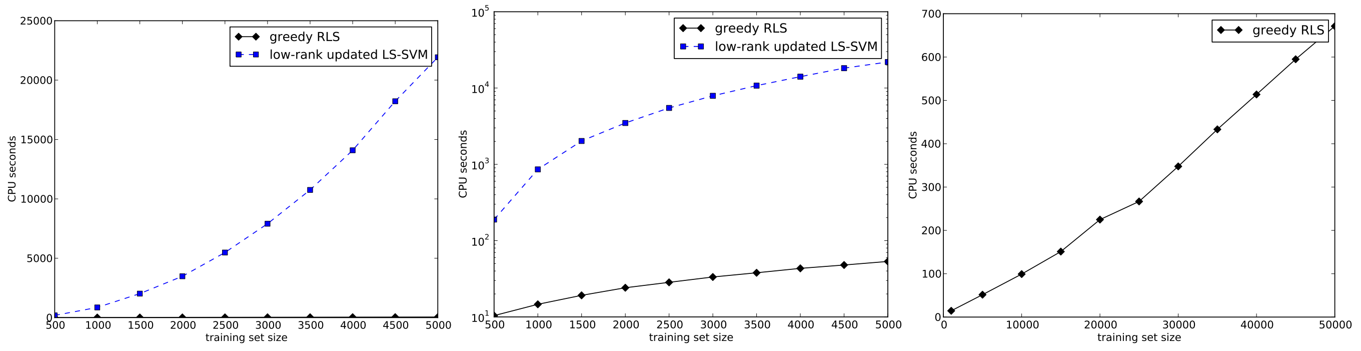

Figure 3. Running times in CPU seconds for the proposed greedy RLS method and the and the low-rank updated LS-SVM of [10]. Linear scaling on y-axis (left). Logarithmic scaling on y-axis (middle). Running times in CPU seconds for the proposed greedy RLS method (right).

References

[1] R. Rifkin, "Everything old is new again: A fresh look at historical approaches in machine learning," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2002.

[2] J. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. World Scientific Pub. Co., Singapore, 2002.

[3] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, pp. 55–67, 1970.

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

[5] T. Pahikkala, E. Tsivtsivadze, A. Airola, J. Boberg, and J. Järvinen, "An efficient algorithm for learning to rank from preference graphs," *Machine Learning*, vol. 75, no. 1, pp. 129–165, 2009.

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

[7] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[8] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Proceedings of the Eleventh International Conference on Machine Learning*, W. W. Cohen and H. Hirsch, Eds. San Fransisco, CA: Morgan Kaufmann Publishers, 1994, pp. 121–129.

[9] E. K. Tang, P. N. Suganthan, and X. Yao, "Gene selection algorithms for microarray data based on least squares support vector machine," *BMC Bioinformatics*, vol. 7, p. 95, 2006.

[10] F. Ojeda, J. A. Suykens, and B. D. Moor, "Low rank updated LS-SVM classifiers for fast variable selection," *Neural Networks*, vol. 21, no. 2-3, pp. 437 – 449, 2008, advances in Neural Networks Research: IJCNN '07.

[11] R. Rifkin and R. Lippert, "Notes on regularized least squares," Massachusetts Institute of Technology, Tech. Rep. MIT-CSAIL-TR-2007-025, 2007.

[12] T. Pahikkala, A. Airola, and T. Salakoski, "Feature selection for regularized least-squares: New computational short-cuts and fast algorithmic implementations," in *Proceedings of the Twentieth IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2010)*, S. Kaski, D. J. Miller, E. Oja, and A. Honkela, Eds. IEEE, 2010.

[13] T. Pahikkala, A. Airola, P. Naula, and T. Salakoski, "Greedy RankRLS: a linear time algorithm for learning sparse ranking models," in *SIGIR 2010 Workshop on Feature Generation and Selection for Information Retrieval*, E. Gabrilovich, A. J. Smola, and N. Tishby, Eds. ACM, 2010, pp. 11–18.

[14] S. R. Searle, *Matrix Algebra Useful for Statistics*. New York, NY: John Wiley and Sons, Inc., 1982.

[15] C. Saunders, A. Gammerman, and V. Vovk, "Ridge regression learning algorithm in dual variables," in *Proceedings of the Fifteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 515–521.

[16] L. Bottou and C.-J. Lin, "Support vector machine solvers," in *Large-Scale Kernel Machines*, ser. Neural Information Processing, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Eds. Cambridge, MA, USA: MIT Press, 2007, pp. 1–28.

[17] J. Shao, "Linear model selection by cross-validation," *Journal of the American Statistical Association*, vol. 88, no. 422, pp. 486–494, 1993.

[18] T. Pahikkala, J. Boberg, and T. Salakoski, "Fast n-fold cross-validation for regularized least-squares," in *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)*, T. Honkela, T. Raiko, J. Kortela, and H. Valpola, Eds. Espoo, Finland: Otamedia, 2006, pp. 83–90.

[19] S. An, W. Liu, and S. Venkatesh, "Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression," *Pattern Recognition*, vol. 40, no. 8, pp. 2154–2162, 2007.