2010 IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2010)
August 29 – September 1, 2010, Kittilä, Finland

FEATURE SELECTION FOR REGULARIZED LEAST-SQUARES: NEW COMPUTATIONAL SHORT-CUTS AND FAST ALGORITHMIC IMPLEMENTATIONS

Tapio Pahikkala, Antti Airola, and Tapio Salakoski

University of Turku and Turku Centre for Computer Science,
Turku, Finland, e-mail: firstname.lastname@utu.fi.

ABSTRACT

We propose novel computational short-cuts for constructing sparse linear predictors with regularized least-squares (RLS), also known as the least-squares support vector machine or ridge regression. The short-cuts make it possible to accelerate the search in the power set of features with leave-one-out criterion as a search heuristic. Our first short-cut finds the optimal search direction in the power set. The direction means either adding a new feature into the set of selected features or removing one of the previously added features. The second short-cut updates the set of selected features and the corresponding RLS solution according to a given direction. The computational complexities of both short-cuts are $O(mn)$, where m and n are the numbers of training examples and features, respectively. The short-cuts can be used with various different feature selection strategies. As case studies, we present efficient implementations of greedy and floating forward feature selection algorithm for RLS.

1. INTRODUCTION

In this paper, we consider machine learning methods for constructing sparse linear predictors. In the literature, the task is often referred to as feature selection (see e.g. [1]). The feature selection methods are usually divided into three classes, namely, to the so called filter, wrapper (see e.g. [2]), and embedded methods. In particular, we consider the wrapper type of feature selection in which the features are selected through interaction with a machine learning method.

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 more easily be deployed in the instrument 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 dimensions, it makes it easier for a human expert to explain the underlying concept. For example, in life sciences the aim is often to find genes relevant to the problem under consideration.

The number of possible feature sets grows exponentially with the number of available features. Therefore, feature selection methods need a search strategy over the power set of features. In addition to the search strategy, a heuristic for assessing the goodness of the feature subsets is required. As suggested by [3], we use the leave-one-out (LOO) cross-validation approach, where each example in turn is left out of the training set and used for testing, as a search heuristic.

The most popular search strategy in wrapper type of feature selection is so-called greedy forward selection which starts from the empty feature set, and on each iteration adds the feature whose effect on the heuristic is the most beneficial. Another popular strategy is greedy backward elimination which starts from the complete set and keeps removing the features one by one. Sometimes, both approaches suffer from the so-called nesting effect. With this, we refer to the fact that the best subset of size k, for example, does not necessarily cover the features included in the best subset of size $k-1$. It has been shown (see e.g. [4,5]) that floating search methods, which use both forward and backward steps work better than the two above ones, because they are able to correct some of the mistakes made in the previous steps. See also [6] and references therein for some alternative search strategies.

Our method is built upon the regularized least-squares (RLS), also known as the least-squares support vector machine (LS-SVM) and ridge regression, which is a state-of-the art machine learning method suitable both for regression and classification [7–9], and it has also been extended for ranking [10, 11]. An important property of the 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

This work was supported by the Academy of Finland.

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. [12, 13] and references therein) which in turn enables the fast computation of LOO-based performance estimates. Analogously to removing the effect 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. In addition, the computation of LOO performance without computational short-cuts requires m trainings. Furthermore, in each search iteration we have $O(n)$ search directions in the space of feature subsets, because there are $O(n)$ possibilities to add a new feature to or remove a feature from the current set of selected features. Putting everything together, the time complexity of a single iteration step is $O(\min\{k^2m^2n, km^3n\})$ 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, as well as other types of cross-validation, have been used to speed up the evaluation of feature subset quality for RLS (see e.g. [14, 15]). In [16] we proposed an algorithm for greedy forward selection according to the LOO criterion. The algorithm, which we call greedy RLS, can be carried out in $O(kmn)$ time, where k is the number of selected features, which is computationally faster than the previously proposed implementations. In particular, the time complexity of a single iteration step of greedy RLS is $O(mn)$.

In this paper, we consider our computational short-cuts as building blocks and show that they can be used to construct feature selecting algorithms based also on other search strategies than the greedy forward selection. As case studies, we show how to implement both the greedy and the floating search strategies. In our floating search strategy, we follow the idea proposed by [5] in which the algorithm may perform a series of corrective backward elimination steps after each forward selection step. In particular, a backward step is performed when the value of the LOO heuristic deteriorates less than half of the amount it was enhanced in earlier forward steps. Accordingly, the algorithm can correct some of the mistakes it made in the earlier steps but still it eventually keeps enhancing the value of the heuristic. For more in depth analysis of this strategy, we refer to [5].

2. 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 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:

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

where $\lambda > 0$ is a regularization parameter. The first term, 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.

To support the following considerations, we introduce the dual form of the prediction function and some extra notation. According to [17], the prediction function (1) can be

[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: Floating search
   Input: $\mathbf{X} \in \mathbb{R}^{n\times m}, \mathbf{y} \in \mathbb{R}^m, \epsilon, \lambda$
   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  $c \leftarrow 0$;
6  $e_c \leftarrow \sum_{j=1}^{m}(\mathbf{y}_j)^2$;
7  while true do
8      $c \leftarrow c+1$;
9      $\mathcal{D} \leftarrow \{1,\ldots,n\} \setminus \mathcal{S}$
       $b, e_c \leftarrow \mathrm{FOU}(\mathbf{X},\mathbf{C},\mathbf{y},\mathbf{a},\mathbf{d},\mathcal{D},1)$;
10     $\delta_c \leftarrow e_{c-1} - e_c$;
11     if $\delta_c < \epsilon$ then
12         break
13     $\mathbf{C},\mathbf{a},\mathbf{d} \leftarrow \mathrm{PU}(\mathbf{X},\mathbf{C},\mathbf{a},\mathbf{d},b,1)$;
14     $\mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$;
15     while true do
16         $b, e' \leftarrow \mathrm{FOU}(\mathbf{X},\mathbf{C},\mathbf{y},\mathbf{a},\mathbf{d},\mathcal{S},-1)$;
17         $\delta' \leftarrow e' - e_c$;
18         if $\delta' > 0.5\delta_c$ then
19             break
20         $\mathbf{C},\mathbf{a},\mathbf{d} \leftarrow \mathrm{PU}(\mathbf{X},\mathbf{C},\mathbf{a},\mathbf{d},b,-1)$;
21         $\mathcal{S} \leftarrow \mathcal{S} \setminus \{b\}$;
22         $c \leftarrow c-1$;
23         $e_c \leftarrow e'$;
24 $\mathbf{w} \leftarrow \mathbf{X}_{\mathcal{S}}\mathbf{a}$;
```

Fig. 1. The floating search algorithm.

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

Next, we consider a well-known efficient approach for evaluating the LOO performance of a trained RLS predictor (see e.g. [13]). 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

$$\mathbf{y}_j - (\mathbf{G}_{j,j})^{-1}\mathbf{a}_j. \tag{2}$$

3. NOVEL COMPUTATIONAL SHORT-CUTS

Here, we present our novel computational short-cuts enabling the construction of various types of feature subset search strategies with LOO error as a heuristic. In addition, we construct implementations of greedy and floating forward feature selection algorithms for RLS with LOO criterion, which act as case studies about how the computational short-cuts can be employed. We refer to the algorithms as greedy RLS and floating RLS. Both algorithms start from an empty feature set $\mathcal{S} = \emptyset$, and on each iteration add the feature whose addition provides the best LOO performance. Greedy RLS only adds features to the set of selected features and never removes any from it. In contrast, floating RLS may perform a series of corrective backward steps after each addition of a new feature during which it removes some of the previously added features if the removal does not increase the LOO error too much. Both algorithms stop if the decrease of LOO error gained by adding a new feature would be smaller than a given threshold $\epsilon > 0$.

Pseudo code of floating RLS is presented in Fig. 1. The greedy RLS algorithm is obtained if the inner loop, starting from line 15 in the pseudo code, is removed (see [16] for a more detailed description of greedy RLS). In addition, we present the pseudo codes of our two computational short-cuts (Fig. 2 and Fig. 3) which the algorithms take advantage of.

In order to take advantage of the computational short-cuts, feature selection algorithm 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

$$\mathbf{a} = \mathbf{G}\mathbf{y}, \tag{3}$$
$$\mathbf{d} = \mathrm{diag}(\mathbf{G}), \tag{4}$$
$$\mathbf{C} = \mathbf{G}\mathbf{X}^{\mathrm{T}}, \tag{5}$$

where

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

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

In the initialization phase of the floating RLS algorithm (lines 1-4 in Fig. 1) 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 computational 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.

Let us now consider the function FOU described in Fig. 2, which finds the optimal update direction given a set $\mathcal{D} \subseteq \{1,\ldots,n\}$ of available directions, that is, the update direction that has the lowest LOO error. The update may refer to either adding a new feature into the set of selected features (the argument t has the value 1) or removing one of the features from the set (the value of t is -1). In addition to $\mathcal{D}$ and t, the function FOU gets as arguments the matrices $\mathbf{X},\mathbf{C} \in \mathbb{R}^{n\times m}$ and the vectors $\mathbf{y},\mathbf{a},\mathbf{d} \in \mathbb{R}^m$. The function

Algorithm 2: Find optimal update (FOU)

```
Input: $\mathbf{X}, \mathbf{C} \in \mathbb{R}^{n\times m}$, $\mathbf{y}, \mathbf{a}, \mathbf{d} \in \mathbb{R}^m$,
       $\mathcal{D} \subseteq \{1,\ldots,n\}$, $t \in \{-1,1\}$
Output: $b$, $e$
1  $e \leftarrow \infty$;
2  $b \leftarrow 0$;
3  foreach $i \in \mathcal{D}$ do
4      $\mathbf{v} \leftarrow (\mathbf{X}_i)^{\mathrm{T}}$;
5      $\mathbf{u} \leftarrow \mathbf{C}_{:,i}(t + \mathbf{v}^{\mathrm{T}}\mathbf{C}_{:,i})^{-1}$;
6      $\tilde{\mathbf{a}} \leftarrow \mathbf{a} - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{a})$;
7      $e_i \leftarrow 0$;
8      foreach $j \in \{1,\ldots,m\}$ do
9          $\tilde{\mathbf{d}}_j \leftarrow \mathbf{d}_j - \mathbf{u}_j\mathbf{C}_{j,i}$;
10         $p \leftarrow \mathbf{y}_j - (\tilde{\mathbf{d}}_j)^{-1}\tilde{\mathbf{a}}_j$;
11         $e_i \leftarrow e_i + l(\mathbf{y}_j, p)$;
12     if $e_i < e$ then
13         $e \leftarrow e_i$;
14         $b \leftarrow i$;
```

Fig. 2. Find optimal update (FOU) function.

returns the index b corresponding the optimal update direction and the value of the LOO error e the predictor would have in case the update would be performed.

Computing the LOO performance with the formula (2) for the modified feature set $\mathcal{S} \cup \{i\}$ or $\mathcal{S} \setminus \{i\}$, where i is the index of the feature to be added or removed, requires the vectors $\tilde{\mathbf{a}} = \widetilde{\mathbf{G}}\mathbf{y}$ and $\tilde{\mathbf{d}} = \text{diag}(\widetilde{\mathbf{G}})$, where

$$\widetilde{\mathbf{G}} = ((\mathbf{X}_{\mathcal{S}})^{\mathrm{T}}\mathbf{X}_{\mathcal{S}} + \mathbf{v}^{\mathrm{T}}t\mathbf{v} + \lambda\mathbf{I})^{-1} \qquad (7)$$

and $\mathbf{v} = (\mathbf{X}_i)^{\mathrm{T}}$. Let us define a vector

$$\mathbf{u} = \mathbf{C}_{:,i}(t + \mathbf{v}^{\mathrm{T}}\mathbf{C}_{:,i})^{-1} \qquad (8)$$

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

$$\begin{aligned}\widetilde{\mathbf{G}} &= \mathbf{G} - \mathbf{G}\mathbf{v}(t + \mathbf{v}^{\mathrm{T}}\mathbf{G}\mathbf{v})^{-1}\mathbf{v}^{\mathrm{T}}\mathbf{G} \\ &= \mathbf{G} - \mathbf{u}\mathbf{v}^{\mathrm{T}}\mathbf{G} \qquad (9)\end{aligned}$$

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

$$\begin{aligned}\tilde{\mathbf{a}} &= \widetilde{\mathbf{G}}\mathbf{y} \\ &= (\mathbf{G} - \mathbf{u}\mathbf{v}^{\mathrm{T}}\mathbf{G})\mathbf{y} \\ &= \mathbf{a} - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{a}) \qquad (10)\end{aligned}$$

in which the lowermost expression is also computable in

Algorithm 3: Perform update (PU)

```
Input: $\mathbf{X}, \mathbf{C} \in \mathbb{R}^{n\times m}$, $\mathbf{a}, \mathbf{d} \in \mathbb{R}^m$, $b \in \{1,\ldots,n\}$,
       $t \in \{-1,1\}$
Output: $\mathbf{C}, \mathbf{a}, \mathbf{d}$
1  $\mathbf{v} \leftarrow (\mathbf{X}_b)^{\mathrm{T}}$;
2  $\mathbf{u} \leftarrow \mathbf{C}_{:,b}(t + \mathbf{v}^{\mathrm{T}}\mathbf{C}_{:,b})^{-1}$;
3  $\mathbf{a} \leftarrow \mathbf{a} - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{a})$;
4  foreach $j \in \{1,\ldots,m\}$ do
5      $\mathbf{d}_j \leftarrow \mathbf{d}_j - \mathbf{u}_j\mathbf{C}_{j,b}$;
6  $\mathbf{C} \leftarrow \mathbf{C} - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{C})$;
```

Fig. 3. Perform update (PU) function.

$O(m)$ time. Further, the entries of $\mathbf{d}$ can be computed from

$$\begin{aligned}\mathbf{d}_j &= \widetilde{\mathbf{G}}_{j,j} \\ &= (\mathbf{G} - \mathbf{u}\mathbf{v}^{\mathrm{T}}\mathbf{G})_{j,j} \\ &= (\mathbf{G} - \mathbf{u}(\mathbf{C}_{:,i})^{\mathrm{T}})_{j,j} \\ &= \mathbf{d}_j - \mathbf{u}_j\mathbf{C}_{j,i} \qquad (11)\end{aligned}$$

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 $|\mathcal{D}|$ features needs $O(m|\mathcal{D}|)$ floating point operations.

Let us next consider the function PU described in Fig. 3, which performs the required updates to the matrix $\mathbf{C}$ and to the vectors $\mathbf{a}$ and $\mathbf{d}$ corresponding to either addition or removal of a feature indicated by the argument b. Whether the feature is added or removed is again indicated by the argument t. The updates to the vectors $\mathbf{a}$ and $\mathbf{d}$ are performed in $O(m)$ time in a way analogous to the temporary updates done in function FOU. Updating the cache matrix $\mathbf{C}$ is also required when the set of selected features is updated permanently. Putting together (5), (8), and (9), $\mathbf{C}$ can be updated via

$$\mathbf{C} - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{C}),$$

which requires $O(mn)$ time which dominates the computational complexity of the function PU.

Back to the main algorithm, the loop starting from line 7 in Fig. 1 first performs one forward selection step, that is, it adds one new feature to the set of selected features $\mathcal{S}$ by first calling the function FOU in order to find the optimal feature to be added and then calling the function PU which performs the actual update operation. In addition, the algorithm performs book keeping by storing the the value of LOO error e_c corresponding to the updated feature set and the difference $\delta_c = e_{c-1} - e_c$ of the new LOO error and the LOO error before the update. The LOO error corresponding to the empty feature set is computed in the beginning of the algorithm. The inner loop may perform corrective backward elimination steps after each forward selection step. Here, we follow the idea proposed by [5]. Namely, a backward step is performed when the increase (denoted by δ' in Fig. 1) of LOO error is no more than half of the decrease of the LOO error in earlier forward steps.

Since the computational complexities of both FOU and PU functions are of the order $O(mn)$, the overall time complexities of floating and greedy RLS are $O(kmn)$, where k is the number of feature additions or removals. For greedy RLS, k is also the number of selected features (see [16]).

4. EXPERIMENTAL RESULTS

In the experiments we test the greedy and floating feature selection strategies on the MNIST handwritten digit database[2]. We consider an artificial regression task, where all the examples corresponding to the digit 5 have the label 1, and all the rest of the examples the label -1. We use the standard training/test split with 60000 training examples, and 10000 test examples. The data set has 780 features, of which we select up to 100. Mean squared error is used to measure performance. The regularization parameter is set to 1. In the experiments we compare the greedy RLS and the floating RLS. Both were implemented in Python, as part of the RLScore[3] open source machine learning software framework. The experiments were run on a modern desktop computer with 2.4 GHz processor and 8 GB of main memory.

First we study the computational efficiency of floating RLS. In Fig 4 (top) are the running times in CPU seconds for greedy RLS and floating RLS for varying sparsity levels. Because floating RLS on each iteration calculates the optimal direction for backward update, and sometimes takes this step, its running time is a between two to three times that of the greedy RLS. Still, even when selecting 100 features out of 780 on a data set of 60000 examples, the running time is less than half an hour. Thus already with a Python implementation the method scales well to large data sets, optimized implementation in a low-level language could be expected to provide further increase in efficiency.

In Fig. 4 (middle) are the LOO results for varying sparsity levels, and in Fig. 4 (bottom) the corresponding test errors. The results for the greedy and floating search are quite close to each other. The correcting steps taken by the floating search lead to slightly lower LOO-errors, and also to lower test errors. Thus with the floating strategy one can with a smaller feature set reach the same performance than with the greedy search. When selecting close to 100 features the test error for the floating search becomes slightly worse than that of the greedy search, probably due to overfitting.

[2]Available at `http://yann.lecun.com/exdb/mnist/`
[3]Available at `http://www.tucs.fi/RLScore`

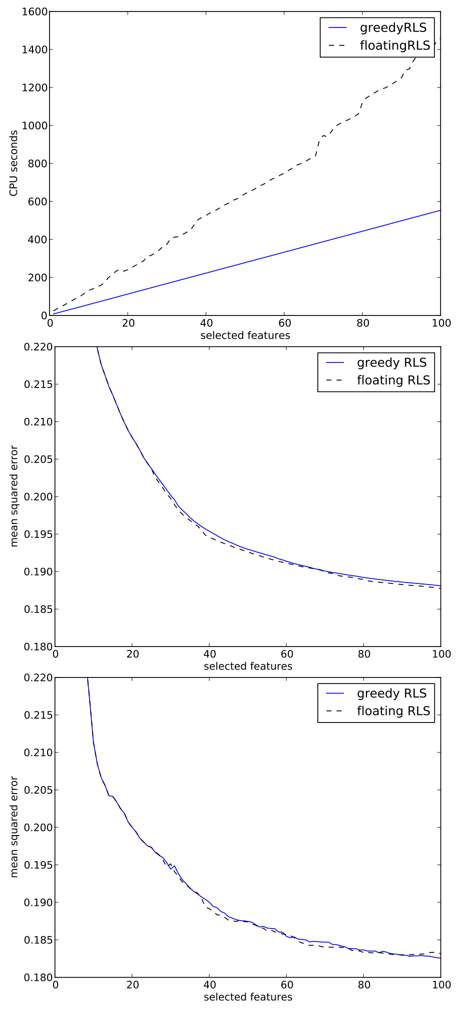

Fig. 4. Runtimes on the MNIST data for varying sparsity levels (top). LOO-error on the MNIST data for varying sparsity levels (middle). Test error on the MNIST data for varying sparsity levels (bottom).

5. CONCLUSIONS

In this paper, we have proposed two computational short-cuts which can be used as building blocks in feature selection algorithms for regularized least-squares (RLS), also known as the least-squares support vector machine or ridge regression. The first short-cut finds the optimal direction in the space of feature subsets, the optimality being determined by the leave-one-out (LOO) criterion. Given an update direction which refers to either adding a new feature into the set of selected features or removing one of the previously added features from the set, the second short-cut updates the set of selected features and the corresponding RLS solution accordingly. The computational complexity of both short-cuts is $O(mn)$, where m is the number of training examples and n is the overall number of features.

In our experiments, we test two feature selection algorithms that take advantage of the presented short-cuts. The first is greedy RLS, a feature selection based on greedy forward selection, whose overall computational complexity is $O(kmn)$, where k is the desired number of features to be selected. The second is a method based on the floating search strategy. The floating search is computationally slightly slower than greedy RLS but it is able to correct some of its mistakes by discarding previously selected features. To conclude, the presented computational short-cuts can be used to implement various different feature selection strategies for RLS which in turn enable the fast learning of accurate but sparse linear predictors.

Acknowledgment

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

6. REFERENCES

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

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

[3] George H. John, Ron Kohavi, and Karl Pfleger, "Irrelevant features and the subset selection problem," in *Proceedings of the Eleventh International Conference on Machine Learning*, William W. Cohen and Haym Hirsch, Eds., San Fransisco, CA, 1994, pp. 121–129, Morgan Kaufmann Publishers.

[4] P. Pudil, J. Novovičová, and J. Kittler, "Floating search methods in feature selection," *Pattern Recogn. Lett.*, vol. 15, no. 11, pp. 1119–1125, 1994.

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

[6] G. Van Dijck and M.M. Van Hulle, "Speeding up the wrapper feature subset selection in regression by mutual information relevance and redundancy analysis," in *Proceedings of the 16th International Conference on Artificial Neural Networks (ICANN 2006)*, S.D. Kollias, A. Stafylopatis, W. Duch, and E. Oja, Eds. 2006, vol. 4131 of *Lecture Notes in Computer Science*, pp. 31–40, Springer.

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

[8] 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.

[9] R. Rifkin, G. Yeo, and T. Poggio, "Regularized least-squares classification," in *Advances in Learning Theory: Methods, Model and Applications*, J.A.K. Suykens, G. Horvath, S. Basu, C. Micchelli, and J. Vandewalle, Eds., vol. 190 of *NATO Science Series III: Computer and System Sciences*, chapter 7, pp. 131–154. IOS Press, Amsterdam, 2003.

[10] 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.

[11] 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.

[12] T. Pahikkala, J. Boberg, and T. Salakoski, "Fast n-fold cross-validation for regularized least-squares," in *Proceedings of the 9th Scandinavian Conference on Artificial Intelligence (SCAI 2006)*, T. Honkela, T. Raiko, J. Kortela, and H. Valpola, Eds., 2006, pp. 83–90.

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

[14] 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, pp. 95, 2006.

[15] F. Ojeda, J.A.K. Suykens, and B. De Moor, "Low rank updated LS-SVM classifiers for fast variable selection," *Neural Networks*, vol. 21, no. 2-3, pp. 437–449, 2008.

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

[17] 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, 1998, pp. 515–521, Morgan Kaufmann Publishers Inc.