Variable selection by rank-one updates for least squares support vector machines

Fabian Ojeda, Johan A.K. Suykens and Bart De Moor

***Abstract*— Least squares support vector machines (LS-SVM) classifiers are a class of simple, yet powerful, kernel methods whose solution follows from a set of linear equations. Here, forward and backward algorithms, based on this technique, are proposed for fast and efficient variable selection. By exploiting the structure of the LS-SVM solution a closed form expression for the leave-one-out (LOO) estimator, useful for selecting variables, is obtained. For inclusion or removal of a new variable, rank-one adjustments in the kernel matrix (linear kernel) allow for updating, rather than recomputing, the LS-SVM solution. The proposed approach is applied to microarray data for gene selection. Simulations clearly show lower computational complexity along with good stability on the generalization performance when compared to other related algorithms.**

I. Introduction

The task of variable selection refers to the problem of selecting a subset of the input variables that are most predictive of a given outcome. Such type of problems are found in almost all areas of machine learning research: classification, regression, time series prediction. The importance resides in the possibility of building predictive models from a relatively few number of variables that summarizes the data with at least the same, or even better, performance than the full initial set. Advantages of using variable selection are among others (i) improving the prediction performance of the predictors, (ii) reducing storage requirements while providing a better understanding of the underlying concept that generated the data.

In recent years, variable selection has become the focus of numerous research studies in areas where datasets with tens or hundreds of thousands of variables are available. These areas mainly refer to text processing (e.g. web mining) and genomics, particularly gene selection in microarray data. In the latter case, the variables are gene expression coefficients, which correspond for example to the abundance of mRNA in a sample (e.g. tissue biopsy) from a number of patients. In cancer research, a typical application aims at separating healthy patients from affected patients based on their gene expression profile. Both unsupervised [1], [2] and supervised approaches [3], have been used to analyze microarray data. In gene selection, the construction of predictive models with fewer number of genes could give a deeper insight into the nature of a particular study and the role that those genes play on it.

The authors are with the Department of Electrical Engineering (ESAT - SCD division), Katholieke Universiteit Leuven, B-3001 Leuven, Belgium (email: fabian.ojeda@esat.kuleuven.be, johan.suykens@esat.kuleuven.be, bart.demoor@esat.kuleuven.be).

Variable selection algorithms are generally grouped into two main categories [4]: *filter* methods, which select variables independently to the predictor and *wrapper* methods for which the way of selecting variables is related to the predictor performance. The main drawback in variable selection, however, resides in its combinatorial nature, which turns into a NP hard problem. The use of exhaustive search methods is unpractical for large numbers of variables, therefore the design of efficient methods and reliable criteria becomes crucial in data analysis.

Support vector machines (SVM) [5] were initially introduced for solving binary pattern recognition problems. Having strong mathematical foundations, their solution translates into quadratic programming (QP) problem, which is convex and guarantees a global minimum of the error function. Successful applications in different domains have converted them into one of the most promising techniques. In the recursive feature elimination (RFE) algorithm introduced by [6], variables are successively eliminated during training of a sequence of support vector machine classifiers. Least squares support vector machines (LS-SVM) [7] are a modified version of the standard SVM algorithm, whose solution results into a set of linear equations instead of a QP problem. In the context of variable selection, they have been also recently used [8], [9], [10].

Our study focuses in the use of LS-SVM for variable selection, particulary in high dimensional data. The structure of the LS-SVM solution allows for a fast method to compute the leave-one-out error [11], [12]. However, the number of calculations during the selection procedure requires a repetitive evaluation of the LS-SVM classifier, i.e. a matrix inversion. For a large number of variables this approach quickly becomes an expensive computational task. The use of rank-one updates [13] in combination with LS-SVM provides a simpler methodology for speeding up this task. Efficient versions of forward and backward variable selection algorithms are presented. They allow for updating the solution at each step rather than recomputing it.

This paper is organized as follows. Section II gives an overview of wrapper based algorithms for variable selection. In section III the LS-SVM algorithm is briefly reviewed. The leave-one-out computation for LS-SVM is given in section IV. The use of rank-one updates in combination with LS-SVM is described in section V. Experimental results are presented in section VI and in section VII concluiding comments are given.

II. Wrapper based variable selection

Variable selection algorithms require a criterion and/or an evaluation measure guiding the selection of potential variables. Selected variables should guarantee, under some conditions, better or at most equal performance to the full set of variables.

The wrapper methodology for variable selection [4], in general, consists in using the prediction performance of a given learning machine to assess the relative usefulness of subsets of variables. The learning machine, in fact, is considered as a black box to score subsets of variables according to its predictive power. In practice, one needs to define (i) how to search in the space of all possible variable subsets and (ii) how to assess the prediction performance of the learning machine to guide the search. Sequential algorithms are the most common choice of search strategies as they cover efficiently a large portion of the whole space search. If the number of variables is sufficiently large, however, they demand massive amount of computations. These greedy search strategies are particulary simple and universal. They come in two main types: *forward selection* and *backward elimination*.

In forward selection, variables are progressively incorporated into increasingly larger subsets. At each step the best canditate, in terms of the learning machine criteria, is added to the selected set. This procedure continues until all candidate variables, or a maximum number of them fixed apriori, are selected. Backward elimination, on the other hand, starts with the full set of variables and progressively eliminates the less promising ones. In other words, those subsets that do not degrade the performance of the learning machine are removed. Usually the search ends when is not possible to remove another variable without decreasing the learning machine performance. Both of these alternatives are explored in this work. The sequential mechanism inside these two procedures is combined with the least-square support vector machine (LS-SVM) classifier. The selection criteria guiding the search is explained in terms of the leave-one-out error estimator.

III. Least squares support vector machines

Given a set of training examples $\{\boldsymbol{x}_i, y_i\} \in X \times Y$, $y_i \in \{-1, +1\}$, the objective in classification is to learn a mapping from an input set X to an output set Y. Ideally, the function f chosen should minimize a generalization error, that is to say, it should predict, the most accurately as possible, the correct y associated to an unseen $\boldsymbol{x}$ [14]. Learning a classifier involves inducing a model from the training data set that can be used to classify a new sample vector into one of the existing classes. This new data is often referred to as the test data.

Support vector machines were initially introduced for solving binary pattern recognition problems [5]. This algorithm separates the two classes by a hyperplane that maximizes the distance between the hyperplane and the nearest sample of each class. The determination of such hyperplane involves the solution of a convex quadratic programming (QP) problem with unique solution. A large number of different-domain applications have confirmed the superiority of SVM based methods over classical approaches and neural networks. Least squares support vector machines (LS-SVM) classifiers [15] are a modification of the standard support vector machine algorithm. By using equality constraints and an explicit squared loss function, the solution to LS-SVM is a system of linear equations. This is far simpler and less computational expensive than the QP problem in the standard SVM.

A. Primal formulation

In LS-SVMs for classification, the inputs $\{\boldsymbol{x}_i\}_{i=1}^n, \boldsymbol{x}_i \in \mathbb{R}^d$, are first mapped to a high (and possibly) infinite dimensional feature space via $\boldsymbol{\varphi}(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}^{d_\varphi}$. In this high dimensional feature space, the LS-SVM classifier takes the form

$$\hat{y} = f(\boldsymbol{x}) = \operatorname{sign}\big(\boldsymbol{w}^T \boldsymbol{\varphi}(\boldsymbol{x}) + b\big) \tag{1}$$

The LS-SVM classifier can be expressed in the form of the following constrained optimization problem

$$\min_{\boldsymbol{w},b,e} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \gamma\frac{1}{2}\sum_{i=1}^{n} e_i^2 \tag{2}$$

$$\text{s.t.} \quad e_i = y_i - (\boldsymbol{w}^T\boldsymbol{\varphi}(\boldsymbol{x}_i) + b)\ i = 1, \ldots, n \tag{3}$$

The second term corresponds to the sum of squared errors term, while the first term can be interpreted as the large margin in support vector machines (SVM). The parameter $\gamma > 0$, determines the trade-off between regularization (γ small) and error minimization (γ large). A correct choice of γ prevents from overfitting.

B. Dual formulation

The solution for LS-SVM in the primal space follows from a linear set of equations of dimension $(n_\varphi + 1) \times (n_\varphi + 1)$, but, as in the case of SVM and kernel methods in general, the feature vector $\boldsymbol{\varphi}(\boldsymbol{x})$ is only implicitly known from Mercer's theorem: $\boldsymbol{\varphi}(\boldsymbol{x}_i)^T\boldsymbol{\varphi}(\boldsymbol{x}_j) = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. For a given positive definite kernel K, the solution of (2), is given by the dual problem in the form of the system of linear equations

$$\left[\begin{array}{c|c} \boldsymbol{\Omega} + \gamma^{-1}\boldsymbol{I}_n & \mathbf{1} \\ \hline \mathbf{1}^T & 0 \end{array}\right]\left[\begin{array}{c} \boldsymbol{\alpha} \\ \hline b \end{array}\right] = \left[\begin{array}{c} \boldsymbol{y} \\ \hline 0 \end{array}\right], \tag{4}$$

with $\boldsymbol{y} = [y_1; \ldots; y_n]$ and $\boldsymbol{\alpha} = [\alpha_1; \ldots; \alpha_n]$ and $\boldsymbol{\Omega}_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. For a new given input $\boldsymbol{x}$, the least-squares support vector machine classifier is given by

$$\hat{y} = f(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b\right), \tag{5}$$

Typical kernel functions are the linear kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T\boldsymbol{x}_j$ and the RBF kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-||\boldsymbol{x}_i - \boldsymbol{x}_j||^2/\sigma^2)$. Alternatively, Eq. (4) can be factorized into

$$\left[\begin{array}{c|c} \boldsymbol{H} & 0 \\ \hline 0 & \mathbf{1}^T\boldsymbol{H}^{-1}\mathbf{1} \end{array}\right]\left[\begin{array}{c} \boldsymbol{\alpha} + \boldsymbol{H}^{-1}\mathbf{1}b \\ \hline b \end{array}\right] = \left[\begin{array}{c} \boldsymbol{y} \\ \hline \mathbf{1}\boldsymbol{H}^{-1}\boldsymbol{y} \end{array}\right], \tag{6}$$

with $\boldsymbol{H} = \boldsymbol{\Omega} + \gamma^{-1}\boldsymbol{I}_n$. Since $\mathbf{1}^T\boldsymbol{H}^{-1}\mathbf{1} > 0$ is positive and $\boldsymbol{H}$ is positive definite, the overall matrix is also positive definite. This form is very suitable for efficient iterative algorithms involving positive definite matrices [15]. Model parameters of the LS-SVM system are now given in terms of $\boldsymbol{H}^{-1}$ by

$$\begin{aligned} b &= \boldsymbol{y}^T\boldsymbol{H}^{-1}\mathbf{1}\left(\boldsymbol{y}^T\boldsymbol{H}^{-1}\boldsymbol{y}\right)^{-1} \\ \boldsymbol{\alpha} &= \boldsymbol{H}^{-1}\left(\boldsymbol{y} - b\mathbf{1}\right) \end{aligned} \tag{7}$$

As mentioned earlier, the main differences with respect to the standard SVM are the equality constraints and the squared loss function. In [15], an empirical study showed that the LS-SVM can achieve comparable performance as the standard SVM on different benchmark data sets. Also in [16], LS-SVM were applied to microarray data.

IV. Leave-One-Out error

The leave-one-out error is an important statistical estimator of the performance of a learning algorithm. It is refferred as an almost unbiased estimator of the generalization error and has been frequently used for model selection [17].

The basic procedure consists in training the learning algorithm from $n-1$ examples, testing the remaining one (leave-out sample) and repeating until all elements have served as test example. The leave-one-out error is defined as

$$\text{error} = \frac{1}{n}\sum_{i=1}^{n}\ell(f^{(-i)}(\boldsymbol{x}_i), y_i) \tag{8}$$

where $f^{(-i)}(\boldsymbol{x}_i)$ is the prediction made for $\boldsymbol{x}_i$ by the classifier when trained on all the examples except $\boldsymbol{x}_i$. The loss function $\ell(\cdot)$ weights the cost (error) when $f(\boldsymbol{x})$ is predicted instead of y. The exact computation of (8), however, requires n runs of the learning algorithm. Previous works have focused in approximating it by deriving upper bounds on the leave-one-out error.

A. Leave-one-out computation for LS-SVMs

The systems of linear equations given by Eq. (4) can be represented in the form of a block structured matrix

$$\left[\begin{array}{c|c} \boldsymbol{H} & \mathbf{1} \\ \hline \mathbf{1}^T & 0 \end{array}\right] = \left[\begin{array}{c|c} \boldsymbol{A}_{11} & \boldsymbol{a}_{12} \\ \hline \boldsymbol{a}_{12}^T & a_{22} \end{array}\right] = \boldsymbol{A} \tag{9}$$

where $\boldsymbol{a}_{12}$, $\boldsymbol{a}_{21}$ are column vectors and a_{22} is a scalar. According to [11] and [18], by exploiting the block structure on Eq. (4), the residual of the leave-one-out prediction for the i-th training pattern can be expressed in a closed form as

$$r_i^{(-i)} = y_i - \hat{y}_i^{(-i)} = \frac{\alpha_i}{\boldsymbol{A}_{ii}^{-1}} \tag{10}$$

The misclassification error is expressed then as

$$\text{error} = \frac{1}{n}\sum_{i=1}^{n}\theta(y_i r_i^{(-i)} - 1) \tag{11}$$

with the step function $\theta(x) = 1, x \geq 0$.

Given that the solution to $\boldsymbol{\alpha}$ is obtained via explicit inversion of $\boldsymbol{A}$, the leave-one-out error can now be computed using the information already available from the training phase of the least-squares support vector machine. This is far more efficient than solving n times the system of linear equations in (4). However, one can readily notice that only the diagonal elements of $\boldsymbol{A}^{-1}$ need to be determined. Using the matrix inversion lemma [13] the inverse of $\boldsymbol{A}$ can be expressed in terms of $\boldsymbol{H}^{-1}$

$$\boldsymbol{A}^{-1} = \left[\begin{array}{c|c} \boldsymbol{H}^{-1} + \frac{1}{s}\boldsymbol{H}^{-1}\mathbf{1}\mathbf{1}^T\boldsymbol{H}^{-1} & -\frac{1}{s}\boldsymbol{H}^{-1}\mathbf{1} \\ \hline -\frac{1}{s}\mathbf{1}^T\boldsymbol{H}^{-1} & \frac{1}{s} \end{array}\right] \tag{12}$$

where the Schur complement $s = -\mathbf{1}^T\boldsymbol{H}^{-1}\mathbf{1}$ is a scalar. Since only the first n diagonal elements of $\boldsymbol{A}^{-1} \in \mathbb{R}^{n+1\times n+1}$ are relevant in computing (10) and given that $\boldsymbol{H}^{-1} \in \mathbb{R}^{n\times n}$ is known, then, using the upper left block in (12) one gets

$$\boldsymbol{A}_{ii}^{-1} = \boldsymbol{H}_{ii}^{-1} + \frac{\nu_i^2}{s},\ i = 1, \ldots, n \tag{13}$$

with $\boldsymbol{\nu} = \boldsymbol{H}^{-1}\mathbf{1},\ \boldsymbol{\nu} \in \mathbb{R}^{n\times 1}$. Algorithm 1 summarizes the procedure for computing the leave-one-error.

Algorithm 1 Efficient leave-one-out for LS-SVM

1: Construct kernel matrix and obtain $\boldsymbol{H}^{-1}$.
2: Solve for b and $\boldsymbol{\alpha}$ using Eq. (7).
3: Compute diagonal elements $\boldsymbol{A}_{ii}^{-1}$ with Eq. (13).
4: Compute prediction residuals using Eq. (10).
5: Finally use Eq. (11) to obtain the leave-one-out error.

V. Rank-one updates

In many applications it is necessary to update a given matrix once it has been altered in some minimal sense. For instance, if the inverse of a given matrix $\boldsymbol{H} \in \mathbb{R}^{n\times n}$ is known, it may also be useful to know how the inverse changes upon addition of a matrix of small rank [13]. In this section we show that in this kind of situations, it is much more efficient to *update* the inverse of $\boldsymbol{H}$ than to generate it from scratch.

Suppose that a nonsingular matrix $\boldsymbol{H}$ has a known inverse $\boldsymbol{H}^{-1}$. Consider also $\boldsymbol{M} = \boldsymbol{H} + \boldsymbol{U}\boldsymbol{R}\boldsymbol{V}^T$ where $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{n\times q}$, $\boldsymbol{R} \in \mathbb{R}^{q\times q}$. If $\boldsymbol{M}$ is nonsingular then by using the Sherman-Woodbury-Morrison formula [13] we have

$$(\boldsymbol{H} + \boldsymbol{U}\boldsymbol{R}\boldsymbol{V}^T)^{-1} = \boldsymbol{H}^{-1} - \boldsymbol{H}^{-1}\boldsymbol{U}\boldsymbol{Z}^{-1}\boldsymbol{V}^T\boldsymbol{H}^{-1} \tag{14}$$

where $\boldsymbol{Z} = \boldsymbol{R}^{-1} + \boldsymbol{V}^T\boldsymbol{H}^{-1}\boldsymbol{U}$. If q is much smaller than n, then $\boldsymbol{R}$ and $\boldsymbol{Z}$ may be much easier to invert than $\boldsymbol{M}$. If the modification to $\boldsymbol{H}^{-1}$ has rank $q = 1$, i.e. $\boldsymbol{U} = \boldsymbol{u}, \boldsymbol{V} = \boldsymbol{v} \in \mathbb{R}^{n\times 1}$ and $\boldsymbol{R} = 1$, the formula becomes

$$\left(\boldsymbol{H} + \boldsymbol{u}\boldsymbol{v}^T\right)^{-1} = \boldsymbol{H}^{-1} - \frac{\boldsymbol{H}^{-1}\boldsymbol{u}\boldsymbol{v}^T\boldsymbol{H}^{-1}}{1 + \boldsymbol{v}^T\boldsymbol{H}^{-1}\boldsymbol{u}} \tag{15}$$

These last two equations show that the inverse of $\boldsymbol{M}$ can be computed from the inverse of $\boldsymbol{H}$. As above mentioned, this is simpler to compute than starting from scratch. Particularly, Eq. (15) only requires simple matrix vector operations given that $\boldsymbol{H}^{-1}$ is already known.

Previously on Section II, the basic mechanism of sequential variable selection algorithms was described. Essentially, a repetitive evaluation of the learning machine is required each time a new variable is tested. For instance, in the LS-SVM classifier, the calculation of $\boldsymbol{H}^{-1}$ for each candidate is required. This becomes prohibitive for large data sets. Let X be described by the set of variables $\mathcal{U} = \{\boldsymbol{u}_k\}_{k=1}^{m}, \boldsymbol{u}_k \in \mathbb{R}^{n\times 1}$, then for a linear kernel we can write

$$\begin{aligned} \boldsymbol{H} &= \boldsymbol{\Omega} + \gamma^{-1}\boldsymbol{I}_n \\ &= \sum_{k=1}^{m} \boldsymbol{u}_k\boldsymbol{u}_k^T + \gamma^{-1}\boldsymbol{I}_n \end{aligned} \tag{16}$$

At the level of variable k we have

$$\begin{aligned} \boldsymbol{H}_k &= \sum_{j=1}^{k-1} \boldsymbol{u}_j\boldsymbol{u}_j^T + \gamma^{-1}\boldsymbol{I}_n + \boldsymbol{u}_k\boldsymbol{u}_k^T \\ \boldsymbol{H}_k &= \boldsymbol{H}_{k-1} + \boldsymbol{u}_k\boldsymbol{u}_k^T \end{aligned} \tag{17}$$

Direct application of Eq. (15) to Eq. (17) results in

$$\boldsymbol{H}_k^{-1} = \boldsymbol{H}_{k-1}^{-1} - \frac{\boldsymbol{H}_{k-1}^{-1}\boldsymbol{u}_k\boldsymbol{u}_k^T\boldsymbol{H}_{k-1}^{-1}}{1 + \boldsymbol{u}_k^T\boldsymbol{H}_{k-1}^{-1}\boldsymbol{u}_k} \tag{18}$$

which suggests that we just need to update $\boldsymbol{H}^{-1}$ from the previous step rather than to recompute it. Combination of this rank-one update procedure with the leave-one-out algorithm given in section IV, provides an efficient algorithm for variable selection in a forward scheme. No matrix inversions are required since the byproduct of the rank-one update is directly used to compute the leave-one-out error. Algorithm 2 summarizes the procedure. The backward version slightly

Algorithm 2 Forward rank-update algorithm

1: $\mathcal{U} = \{\boldsymbol{u}_k\},\ k = 1, \ldots, m$. {full set of variables}
2: Initialize inverse $\boldsymbol{H}_0^{-1} = \gamma \times \boldsymbol{I}_n$ {no variables}
3: **for** $k = 1$ to m **do**
4: Update inverse for each candidate variable as in Eq. (18) $\boldsymbol{H}_k^{-1} = \boldsymbol{H}_{k-1}^{-1} - (\boldsymbol{H}_{k-1}^{-1}\boldsymbol{u}_k\boldsymbol{u}_k^T\boldsymbol{H}_{k-1}^{-1})/(1 + \boldsymbol{u}_k^T\boldsymbol{H}_{k-1}^{-1}\boldsymbol{u}_k)$
5: Compute leave-one-out from $\boldsymbol{H}_k^{-1}$ using Algorithm 1.
6: **end for**

differs from the previous algorithm. Since one starts from the full set of variables, the inverse of the full kernel matrix is required. This computation, however, needs to be carried out only once. Similar to Eq. (15), each time a variable is tested for removal the matrix $\boldsymbol{H}^{-1}$ is downdated as

$$\left(\boldsymbol{H} - \boldsymbol{u}\boldsymbol{u}^T\right)^{-1} = \boldsymbol{H}^{-1} + \frac{\boldsymbol{H}^{-1}\boldsymbol{u}\boldsymbol{u}^T\boldsymbol{H}^{-1}}{1 - \boldsymbol{u}^T\boldsymbol{H}^{-1}\boldsymbol{u}}\ . \tag{19}$$

The formulation given in this section is restricted to the case of having a linear kernel. The rank-one updates expressions are only valid in this situation. The use of non-linear kernel demands the computation of $\boldsymbol{H}^{-1}$ at each selection step.

VI. Empirical Results

In this section, experimental results of the proposed approach are reported on two public available microarray data sets. The colon cancer data set described in [2] and the acute leukemia data set presented in [3]. Both data sets are considered here as two-class problems and were standarized to zero mean and unit variance.

A. Computational time

The efficiency of the forward rank-update algorithm, is compared with other two variants that also employ LS-SVM classifier. The first algorithm presented in [10], uses a closed-form to compute the leave-one-out error. Matrix inversions, however, are still required to solve the LS-SVM system for each new potential subset of variables. The second approach given in [9], makes use of an upper bound in the leave-one-out. Since heavy computations for the error estimation are avoided, this approach provides a very fast algorithm. Additionally, rank-updates are also employed. The expensive part of this algorithm is mainly the computation of a $n \times n$ distance matrix each time a new subset of variables is tested. For this comparison we took the colon cancer data set. The computational times required to select 200 out of 2000 variables are shown in Fig. 1.

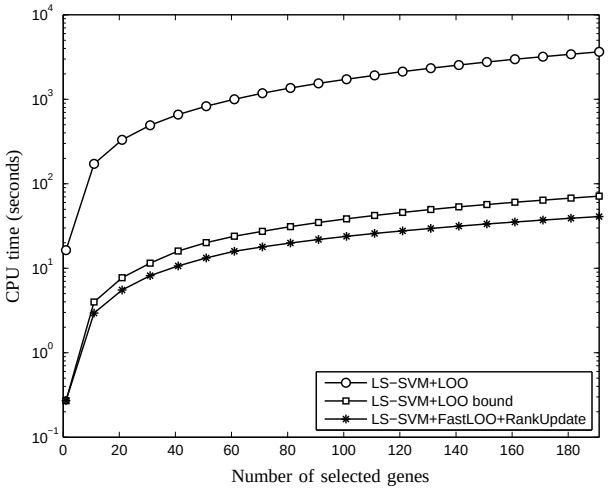

Fig. 1. Computational time for forward selection algorithms. ○: LS-SVM LOO. □: LS-SVM LS Bound. ⋆: LS-SVM Forward rank-one-update.

Clearly, it can be drawn that the proposed algorithm scales very nicely even for high dimensional data sets. The gap with respect to the approach in [10] is readily appreciable (around two orders of magnitude). Compared to the bound approach [9], the proposed method is slightly faster, eventhough we compute the exact leave-one-out error. The combiantion of both, the one-rank update and the closed-form leave-one-out expression, leads to a fast strategy that may be useful for variable selection. It does avoid matrix inversions and relies on simple matrix-vector operations.

For the backward version, again the colon cancer data set is used. Comparison is made with respect to the two variants

of the SVM-RFE algorithm [6]. First variant, SVM-RFE1, consist in computing the support vectors on each step of the elimination phase (retraining), whereas fixing them (no retraining) is the second one, SVM-RFE2. Fig. 2 shows the computational time for ranking 500 variables out of 2000. The quadratic programming problem (QP) involved in the

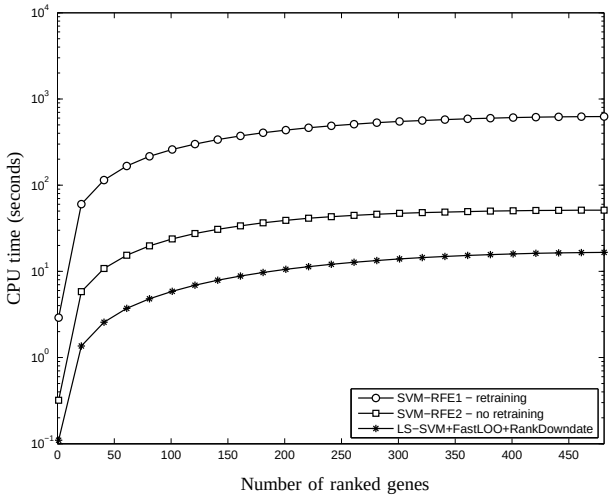

Fig. 2. Computational time for backward selection algorithms. ○: SVM-RFE with retraining. □: SVM-RFE without retraining. ⋆: LS-SVM Backward rank-one-update.

standard SVM, as mentioned earlier, is more computationally expensive than the LS-SVM solution. The sequential nature of the backward elimination increases the volume of computations. The proposed backward approach is even faster than the SVM-RFE1, which avoids a considerable number of operations by keeping fixed the support vector through the whole elimination phase.

B. Classification performance

To assess the performance in classification, the full data set is partitioned into (nearly balanced) training and test sets. An external 10-fold cross-validation, as mentioned in [19], is applied to the training. On each fold the variable selection is done using the leave-one-out error. This methodology aims at reducing the selection bias given the small sample size. Variables consistenly selected in differente folds conform the selected set used to compute the test error. In Fig. 3, the test error for the leukemia data set of forward algorithms is shown. The proposed algorithm exhibes better stability than the bound approach. The tightness of such bound is heavily dependant on the stability of the LS-SVM to perturbations. The exact computation of the leave-one-out has therefore an impact on the test error. The variance of the cross-validation is depicted as error bars. Differences are less remarkable as the number of variable increases.

Backward algorithms are compared using the colon cancer data set. The test error for 1000 out of 2000 selected variables is shown in Fig. 4. The use of the leave-one-out error is clearly superior to the margin criterion used in the SVM-RFE

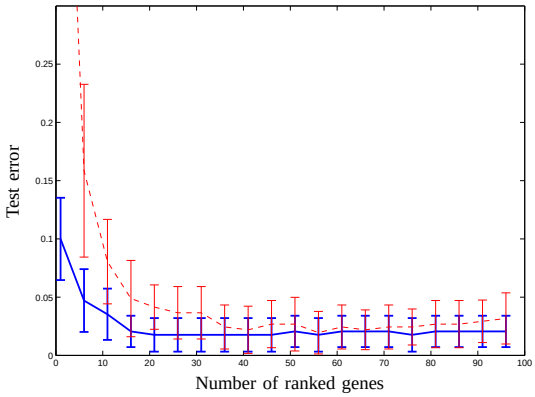

Fig. 3. Test error on the Golub data. Dashed: LS-SVM Bound. Solid: LS-SVM Forward rank-one-update

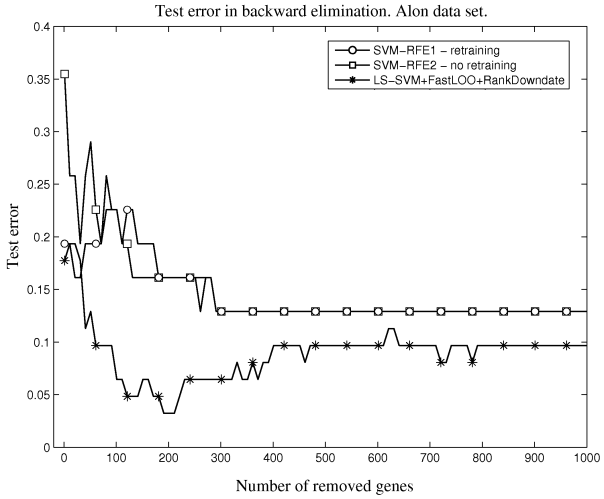

Fig. 4. Test error on the Alon data. ○: SVM-RFE with retraining. □: SVM-RFE without retraining. ⋆: LS-SVM Backward rank-one-update.

algorithm for eliminating variables. This was expected since the margin is a bound on the generalization error. The test error is remarkably lower around 200 variables. However, the small sample size in typical microarray data implies that the error estimator based on the leave-one-out can only provide very limited number of possible error values. Therefore, it is very likely that several, and possibly, different subsets of variables may attain the same minimal error during the selection procedure. A possible solution to this observation would be the use of continous functions for error estimation preserving the computational efficiency of the leave-one-out estimator. In [18] at least 4 different alternatives are presented. The use of continous estimates for the leave-one-out error may reduce the number of equally scoring candidate variables, therefore the probability of selecting irrelevant variables becomes lower. Fig. 5 shows the behavior of two other functions compared to the zero-one loss leave-one-out.

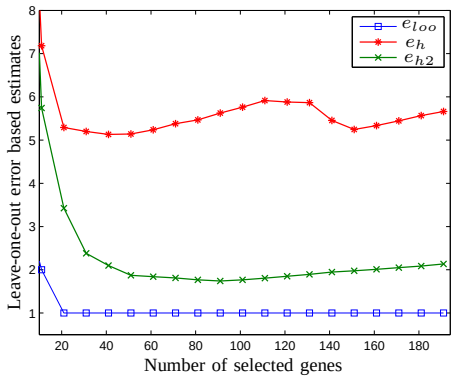

Fig. 5. Comparison of different leave-one-error estimates on the Golub data.

VII. Conclusions

In this paper, algorithms for variable selection by rank-one updates for least squares suppport vector machines (LS-SVM) have been presented. One of their main advantages is the possibility of a fast evaluation of LS-SVM and consequently the computation leave-one-out by the use of rank-one updates. Matrix inversions are avoided in the forward algorithm while the backward version requires only one. Therefore, they can be used even in high dimensional data. The use of an external cross-validation procedure produces a better estimate of the test error. Comparison with other related algorithms delivered satisfactory results in terms of efficiency and generalization performance. Alternatives to the leave-one-out estimator based on continous functions may provide more refined sets of variables, especially for data sets with reduced number of samples.

Acknowledgment

This work was supported by grants and projects for the Research Council K.U.Leuven (GOA AMBioRICS, CoE EF/05/006 Optimization in Engineering, several PhD/postdoc & fellow grants), the Flemish Government FWO: PhD/postdoc grants, projects, G.0407.02 (support vector machines), G.0197.02 (power islands), G.0141.03 (Identification and cryptography), G.0491.03 (control for intensive care glycemia), G.0120.03 (QIT), G.0452.04 (new quantum algorithms), G.0499.04 (Statistics), G.0211.05 (Nonlinear), G.0226.06 (cooperative systems and optimization), G.0321.06 (Tensors), G.0302.07 (SVM/Kernel, research communities (ICCoS, ANMMM, MLDM); IWT: PhD Grants, McKnow-E, Eureka-Flite2; the Belgian Federal Science Policy Office: IUAP P6/04 (Dynamical systems, control and optimization, 2007-2011); EU: ERNSI. Johan A.K. Suykens is a professor at the K.U.Leuven, Belgium. Bart De Moor is a full professor at the K.U.Leuven, Belgium. The scientific responsibility is assumed by its authors.

References

[1] M. Eisen, P. Spellman, P. Brown, and D. Botstein, "Cluster analysis and display of genome-wide expression patterns," *PNAS*, vol. 95, pp. 14 863–14 868, 1998.

[2] U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine, "Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays," *Proc Natl Acad Sci*, vol. 96, no. 12, pp. 6745–6750, 1999.

[3] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander, "Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring," *Science*, vol. 286, 1999.

[4] R. Kohavi and G. John, "Wrappers for feature selection," *Artificial Intelligence*, vol. 97, pp. 273–324, 1997.

[5] V. Vapnik, *Statistical Learning Theory*. Wiley, 1998.

[6] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, 2002.

[7] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[8] M. Xiong, X. Fang, and J. Zhao, "Biomarker identification by feature wrappers," *Genome Research*, vol. 11, pp. 1878–1887, 2001.

[9] X. Zhou and K. Mao, "LS bound based gene selection for DNA microarray data," *Bioinformatics*, vol. 21, no. 8, 2005.

[10] E. K. Tang, P. Suganthan, and X. Yao, "Gene selection for microarray data based on least squares support vector machine," *BMC Bioinformatics*, vol. 7, no. 95, 2006.

[11] T. Van Gestel, B. Baesens, J. A. K. Suykens, M. Espinoza, D.-E. Baestaens, J. Vanthienen, and B. De Moor, "Bankruptcy prediction with least squares support vector machine classifiers," in *International Conference on Computational Intelligence for Financial Enigeering (CIFER'03)*, vol. 1, no. 8, 2003.

[12] G. C. Cawley and N. L. Talbot, "Fast exact leave-one-out cross-validation of sparse least-squares support vector machines," *Neural Networks*, vol. 17, pp. 1467–1475, 2004.

[13] G. H. Golub and C. F. Van Loan, *Matrix computations*, 2nd ed. The Jonhs Hopkins University Press, 1989.

[14] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford UP, 1995.

[15] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. Singapore: World Scientific, 2002.

[16] N. Pochet, F. De Smet, J. A. K. Suykens, and B. De Moor, "Systematic benchmarking of microarrray data classification: assessing the role of non-linearity and dimensionality reduction," *Bioinformatics*, vol. 20, no. 38, pp. 3185–3195, 2004.

[17] P. Lachenbruch, "An almost unbiased method for the probability of misclassification in discriminant analysis," *Biometrics*, vol. 23, pp. 639–645, 1967.

[18] G. C. Cawley, "Leave-one-out cross-validation based model selection criteria for weighted LS-SVMs," in *Proc of te International Joint Conference on Computational Neural Networks (IJCNN'06)*, 2006.

[19] C. Ambroise and G. J. McLachaln, "Selection bias in gene extraction on the basis of microarray gene-expression data," *PNAS*, vol. 99, no. 10, pp. 6562–6566, 2002.