Low rank updated LS-SVM classifiers for fast variable selection

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

Department of Electrical Engineering (ESAT - SCD division), Katholieke Universiteit Leuven, B-3001 Leuven, Belgium

Abstract

Least squares support vector machine (LS-SVM) classifiers are a class of kernel methods whose solution follows from a set of linear equations. In this work we present low rank modifications to the LS-SVM classifiers that are useful for fast and efficient variable selection. The inclusion or removal of a candidate variable can be represented as a low rank modification to the kernel matrix (linear kernel) of the LS-SVM classifier. In this way, the LS-SVM solution can be updated rather than being recomputed, which improves the efficiency of the overall variable selection process. Relevant variables are selected according to a closed form of the leave-one-out (LOO) error estimator, which is obtained as a by-product of the low rank modifications. The proposed approach is applied to several benchmark data sets as well as two microarray data sets. When compared to other related algorithms used for variable selection, simulations applying our approach clearly show a lower computational complexity together with good stability on the generalization error.

Key words:

Low rank matrix modifications, least-squares support vector machines, leave-one-out error, variable selection.

1 Introduction

The task of variable selection consist of selecting a reduced set of variables that is most predictive for a given outcome. Such type of problem is found in almost all areas of machine learning research, namely, classification, regression, and time series prediction. The goal of variable selection is that of building predictive models with less number of variables while providing, at least the same, or even better performance than the full initial set. Advantages of using variable selection are, among others, related to: (i) improve the prediction performance of the predictors, (ii) reduce storage requirements, (iii) provide 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. Such areas mainly refer to text processing (e.g. web mining) and genomics, particularly gene selection in microarray data. Variable selection algorithms are generally grouped into two main categories (Kohavi & John, 1997): (i) *filter* methods, which select variables independently to the predictor and (ii) *wrapper* methods, for which the way of selecting variables is related to the predictor performance. The main drawback in variable selection resides in its combinatorial nature, turning 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 the data analysis workflow.

Email addresses: fabian.ojeda@esat.kuleuven.be, johan.suykens@esat.kuleuven.be, bart.demoor@esat.kuleuven.be (Fabian Ojeda, Johan A.K. Suykens and Bart De Moor).

Support vector machines (SVM) (Vapnik, 1998) were initially introduced for solving binary classification problems. Starting from the statistical learning theory, the solution to SVM is given by a convex quadratic programming (QP) problem, which guarantees a global minimum of the cost function. LS-SVM classifiers (Suykens & Vandewalle, 1999) are a modified version of the standard SVM, whose solution results into a set of linear equations. Successful applications using these two techniques can be extensively found in the machine learning and data mining domains. Particularly, the recursive feature elimination (RFE) algorithm introduced by (Guyon, Weston, Barnhill, & Vapnik, 2002) has been widely referenced in the variable selection context. In such procedure, variables are successively eliminated during the training of a sequence of SVM classifiers. Applications for variable selection using the LS-SVM classifiers have been also recently reported by (Xiong, Fang, & Zhao, 2001; Zhou & Mao, 2005; Tang, Suganthan, & Yao, 2006).

Our study focuses in the use of LS-SVM classifiers for variable selection. Although the structure of the LS-SVM solution allows for a fast method to compute the leave-one-out error (Van Gestel et al., 2003; Cawley & Talbot, 2004), the number of calculations involved during the variable selection procedure still requires repetitive evaluations of the LS-SVM classifier. Since the solution to the LS-SVM is a system of linear equations, we would then need to compute matrix inversions upon any change on the system. For a large number of variables this approach quickly becomes an expensive computational task. Fortunately, for the special case of linear kernels it is possible to reduce computations by using *low rank* modifications (Golub & Van Loan, 1989). Adding or removing a candidate variable can be expressed as a *rank-one* modification into the kernel matrix (linear kernel) of the LS-SVM classifier and therefore the LS-SVM solution can be *updated* rather that being recomputed from scratch.

We are particularly interested in the system of equations of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is $n \times n$ matrix and $\boldsymbol{b}$ is a n-vector. Once $\boldsymbol{x}$ has been computed, it is often necessary to solve the modified system $\tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$. Clearly, we should be able to modify $\boldsymbol{A}$ to obtain factors for $\tilde{\boldsymbol{A}}$, from which $\tilde{\boldsymbol{x}}$ may be computed as before. In this paper we are particularly interested in one type of modification, in which $\tilde{\boldsymbol{A}}$ has the form $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{u}\boldsymbol{u}^T$, where $\boldsymbol{u}$ is a n-vector representing one of the input variables in the learning problem. The matrix $\boldsymbol{u}\boldsymbol{u}^T$ is a matrix of *rank-one* (Golub & Van Loan, 1989). The rank-one modifications to the LS-SVM provide with a methodology for speeding up the task of variable selection. Efficient versions of forward and backward variable selection algorithms are presented here, in which the solution of the LS-SVM is updated at each step, rather than being recomputed.

The work presented here is an extension of (Ojeda, Suykens, & De Moor, 2007). This paper is organized as follows. The problem of variable selection is shortly described in section 2. Section 3 provides an overview of the LS-SVM classifiers (Suykens, Van Gestel, De Brabanter, De Moor, & Vandewalle, 2002), while a methodology for efficient evaluation of the leave-one-out (LOO) error for the LS-SVM is given in section 4. Descriptions on low rank modifications are given in section 5 while their application to the LS-SVM classifiers are described in section 6. The overall strategy presented in this work, helps to tackle the combinatorial characteristic of the variable selection problem. Experimental results are presented in section 7 and in section 8 concluding comments are given.

2 Variable selection

Variable or feature [1] selection is one of the main problems in machine learning and statistics (Bishop, 1995). It usually refers to the task of extracting relevant information present in the data. In the context of classification, variable selection aims at finding a subset of variables that results in more accurate and compact classifiers. Typically, the error of the designed classifier will drop as more variables are added and, after reaching some optimal number (for a given sample size), it will begin to rise. Therefore, the variable selection method must filter out those inputs that are irrelevant to the specific model. Moreover, the selection made should capture the relevant information as well as avoid overfitting the data. In the ideal case, this resorts in determining the single best subset of variables. However, there is no single definition for *the best* subset as different algorithms will lead to different subsets. In general, variable selection is comprised of three elements, namely: *(i)* the search procedure, *(i)* the learning algorithm and *(iii)* the variable selection criterion. Without loss of generality, the problem of variable selection can be stated as:

Given a set of training data $\mathcal{D} = \left\{(\boldsymbol{x}_i, y_i)_{i=1}^n \,\middle|\, \boldsymbol{x}_i \in \mathbb{R}^d,\ y_i \in \{-1, +1\}\right\}$, where the pattern $\boldsymbol{x}_i \in \mathbb{R}^d$, is described by d variables $\mathcal{S} = \{X_1, \ldots, X_k, \ldots, X_d\}$, and the corresponding outputs $y_i \in \{1, -1\}$ are the binary class labels. The goal of variable selection is to choose from the full set of all variables $\mathcal{S}$, a small subset $\mathcal{S}^* \subset \mathcal{S}, \mathcal{S}^* \in \mathbb{R}^p,\ p < d$, such that an objective function $\mathcal{J}_{\mathcal{S}^*} \leq \mathcal{J}_{\mathcal{S}}$ defined over the set of variables is minimized.

In the literature, two major types of methods for variable selection can be distin-

[1] Both terms are often used interchangeably in literature, although in the machine learning community the term feature is used in the context of a induced transformation or mapping.

guished (Kohavi & John, 1997): *(i) filter* and *(ii) wrapper* based methods. The role of the objective function $\mathcal{J}$, is often defined in the context of the specific variable selection method. The *filter* model, considers the variables independently of the learning algorithm. An initial analysis is performed on the training data and later the selected variables are provided to the classifier. A simple filtering technique ranks or scores each variable based on some measures like information gain criterion, mutual information, or t-tests. Since the learning algorithm (e.g. the classifier) is not involved, the main advantage of the filter model is its low computational cost (Kohavi & John, 1997). Ignoring the impact of the learning algorithm, however, constitutes the weakest point of the filter method. In the *wrapper* technique (Kohavi & John, 1997), different subsets of features are tested by means of the learning algorithm. The performance of each subset is evaluated using a *scoring function* like n-fold cross-validation or leave-one-out cross-validation and the best subset is kept. Since the size of the search space scales with the number of variables as 2^d, exhaustive search through the input space is no longer feasible. Therefore, heuristic search methods using *backward* or *forward* search are often used for variable selection. In (Kohavi & John, 1997), it is concluded that *wrapper* models result in an increased accuracy due to the consideration of the interaction between the algorithm and the training set. The disadvantage of *wrapper* methods is the high computational cost of the search. Such aspect constitutes, a key point in this study.

This work focuses on the second class of variable selection methods, namely the *wrapper* technique. We will use the prediction performance of a given learning machine technique to assess the relative usefulness of subsets of variables. The learning machine, in fact, is considered as the black box scoring subsets of variables according to their predictive power. In practice, we need to define (i) how to search in the space of all possible variable subsets and (ii) how to assess the pre-

diction performance of the learning machine. Sequential algorithms are the most common choice of search strategies as they cover efficiently a large portion of the whole space search. These greedy search strategies are particularly simple and universal. They come in two main types: *forward selection* and *backward elimination*. Forward selection refers to a search starting at the empty set of variables, while the backward elimination refers to a search that begins at the full set of variables. When the number of variables is sufficiently large, however, they demand massive amount of computations.

2.1 Sequential Forward Selection (SFS)

In forward selection, variables are progressively incorporated into increasingly larger subsets. At each step the best candidate, in terms of the learning machine criterion, is added to the selected set. This procedure continues until all candidate variables, or a maximum number of them fixed a priori, are selected. Building classifiers in this scheme is often preferred for computational considerations. The algorithms is much faster when there are few variables in the data to search for.

2.2 Sequential Backward Elimination (SBE)

On the other hand, backward elimination 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. Although in theory, searching backwards from the full set of variables may capture interacting features more easily than the forward

scheme, the method is often more expensive in terms of computational cost.

In the following sections, both search algorithms are explored. The use of LS-SVM classifiers as the learning algorithm is motivated on the the basis of its special structure. The leave-one-out (LOO) estimator, directly obtained as a by-product of the basic training algorithm (Cawley & Talbot, 2004; Van Gestel et al., 2003; Cawley, 2006), is taken as the basis of the variable selection criterion. Moreover, considerable savings in terms of computational complexity can be achieved with a particular case of low rank modifications on the LS-SVM solution. Simplifications that eventually allow the practical implementation of forward and backward search strategies even for a relatively high number of variables.

3 Least squares support vector machine classifiers (LS-SVM)

In standard two-class classification problems, we are given a set of training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where the input $\boldsymbol{x}_i \in \mathbb{R}^d$, is described by d variables, and the corresponding output $y_i \in \{1, -1\}$ is binary. We seek to construct a classification rule from the training data, so that given a new input $\boldsymbol{x}$, we can assign a class to it. We consider a discriminant function f of the form $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\varphi}(\boldsymbol{x}) + b$, in a space $\mathcal{F}$, defined by a fixed transformation of the input space $\boldsymbol{\varphi}(\cdot) : \mathbb{R}^d \mapsto \mathcal{F}$.

Support vector machines (SVM)(Vapnik, 1998), provide a solution to the binary classification above-mentioned problem from statistical learning theory. The 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 global solution. In the SVM algorithm, the inputs $\boldsymbol{x}_i$, are first

mapped to a high (and possibly) infinite dimensional feature space by means of the mapping $\boldsymbol{\varphi}(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}^{d_\varphi}$, and the separating hyperplane is constructed in this feature space. The constrained optimization problem is given, in primal form, by (Vapnik, 1998) as

$$\begin{aligned} &\min_{\boldsymbol{w},b,\xi} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n}\xi_i \\ &\text{s.t. } y_i(\boldsymbol{w}^T\boldsymbol{\varphi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \\ &\quad \xi_i \geq 0,\ i = 1, \ldots, n. \end{aligned} \tag{1}$$

where $C \in \mathbb{R}^+$ is a penalty (cost) parameter for the training error. The slack variables $\xi \geq 0$, represent penalization for the misclassified samples. Given a positive definite kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ with $K(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\varphi}(\boldsymbol{x})^T\boldsymbol{\varphi}(\boldsymbol{z})$ and penalty term $C \in \mathbb{R}^+$, the dual problem to (1) is given by the following quadratic programming (QP) problem (Vapnik, 1998) :

$$\begin{aligned} &\max_{\alpha_i} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &\text{s.t. } \sum_{i=1}^{n}\alpha_i y_i = 0,\ C \geq \alpha_i \geq 0,\ i = 1, \ldots, n. \end{aligned} \tag{2}$$

In (2), α_i are the corresponding dual variables, or the associated Lagrange multipliers, to the constraints $y_i(\boldsymbol{w}^T\boldsymbol{\varphi}(\boldsymbol{x}_i)+b) \geq 1-\xi_i$. The feature map $\boldsymbol{\varphi}(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}^{d_\varphi}$, implicitly known by means of Mercer's theorem, $\boldsymbol{\varphi}(\boldsymbol{x}_i)^T\boldsymbol{\varphi}(\boldsymbol{x}_j) = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and the use of a positive definite kernel. The classifier for a new input $\boldsymbol{x}$, takes now the form

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{n}\alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b\right),$$

Typical kernel functions are the linear kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T\boldsymbol{x}_j$ and the radial basis function (RBF) kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-||\boldsymbol{x}_i - \boldsymbol{x}_j||^2/\sigma^2)$. Although SVM were initially introduced for solving binary pattern recognition problems, a large

number of different-domain applications have confirmed their superiority over classical approaches and neural networks-like methods.

Another class of SVM, are that of the LS-SVM classifiers (Suykens et al., 2002; Suykens & Vandewalle, 1999). Such learning algorithms are modifications of the standard support vector machine algorithm. Differences are, specifically, the use of equality constraints and an explicit squared loss function in the primal objective. As a result, the solution follows directly from solving a set of linear equations, instead of quadratic programming. While in classical SVM many support values are zero (nonzero values correspond to support vectors) in LS-SVM, the support values are proportional to the errors. In (Saunders, Gammerman, & Vovk, 1998), a support vector interpretation of ridge regression (Golub & Van Loan, 1989) was proposed for the function estimation problem considering equality type constraints.

The LS-SVM algorithm considers the following constrained optimization problem (Suykens & Vandewalle, 1999)

$$\begin{aligned} &\min_{\boldsymbol{w},b,e} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \gamma\frac{1}{2}\sum_{i=1}^{n} e_i^2 \\ &\text{s.t. } y_i = \boldsymbol{w}^T\boldsymbol{\varphi}(\boldsymbol{x}_i) + b + e_i,\ i = 1,\ldots,n. \end{aligned} \tag{3}$$

where $\gamma \in \mathbb{R}^+$ is a regularization parameter controlling the bias-variance trade-off. The variables $e_i \in \mathbb{R}$, are viewed as normal distributed errors of the outputs y_i. The solution is considered in its dual form. Given a positive definite kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ with $K(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\varphi}(\boldsymbol{x})^T\boldsymbol{\varphi}(\boldsymbol{z})$ and the regularization term $\gamma \in \mathbb{R}^+$, the dual problem to (3) is given by the following set of linear equations

((Suykens & Vandewalle, 1999)):

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

In (4), $\boldsymbol{y}=[y_1;\ldots;y_n]$ and $\boldsymbol{\alpha}=[\alpha_1;\ldots;\alpha_n]$ and $\boldsymbol{K}_{ij}=K(\boldsymbol{x}_i,\boldsymbol{x}_j)$. For a new given input $\boldsymbol{x}$, the least-squares support vector machine classifier is given by

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

The optimal parameters, $\boldsymbol{\alpha}, b$, can be obtained with a computational complexity of $\mathcal{O}(n^3)$ operations ((Golub & Van Loan, 1989)). Alternatively, Eq. (4) can be factorized into the form a positive definite system

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

with $\boldsymbol{H}=\boldsymbol{K}+\gamma^{-1}\boldsymbol{I}_n$. 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 (Suykens et al., 2002). Model parameters of the LS-SVM system are now given solely in terms of $\boldsymbol{H}^{-1}$ by

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

$$\boldsymbol{\alpha}=\boldsymbol{H}^{-1}\left(\boldsymbol{y}-b\mathbf{1}\right) \tag{8}$$

An empirical study by (Van Gestel et al., 2004) showed that the LS-SVM classifiers achieve comparable performance as the standard SVM on a series of benchmark data sets with less overall complexity.

At this point two of the elements in the variable selection process have been de-

scribed: the search method (forward and backward search) and the learning algorithm used to evaluate the candidate subsets, the LS-SVM classifier. The third element, the selection criterion, is given in the next section in the form of the leave-one-out (LOO) estimator. Special attention is given to the efficient implementation for the case of LS-SVM classifiers.

4 Leave-One-Out estimator

The leave-one-out error is an important statistical estimator of the performance of a learning algorithm. It is referred to as an almost unbiased estimator of the generalization error and has been frequently used for model selection ((Lachenbruch, 1967)). 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.

4.1 Criteria

The leave-one-out error can be defined as

$$\text{error} = \frac{1}{n}\sum_{i=1}^{n} \ell(f^{(-i)}(\boldsymbol{x}_i), y_i) \tag{9}$$

where $f^{(-i)}(\boldsymbol{x}_i) = \hat{y}_i$ 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. A typical loss used in classification settings is the so-called *zero-one* function

$$\text{error}_{0/1} = \frac{1}{n}\sum_{i=1}^{n} \Psi\left(y_i - \hat{y}_i^{(-i)}\right) \tag{10}$$

where $\Psi(x) = 1, x \geq 0$ is the step function. Alternatively to the step function for the leave-one-out error, continuous functions or upper bounds on the error rate appear to be more tractable for numerical routines. The simplest criterion is the leave-one-out estimate of the sum of squares error, or PRESS statistic, as suggested in (Cawley, 2006)

$$\text{error}_{\text{press}} = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{y}_i^{(-i)} \right)^2 \tag{11}$$

However, the exact computation of (9) requires n runs of the learning algorithm. In the case of LS-SVM classifiers, the overall complexity is thus $\mathcal{O}(n^4)$. Such approach becomes quickly prohibitive for a large number of samples. In the context of SVM, particularly, many works have focused in developing approximations or deriving upper bounds for the leave-one-out error (Vapnik & Chapelle, 2000). The main reason is the high computational cost involved when solving n-QP problems. Fortunately, for LS-SVM classifiers, the leave-one-out (LOO) error can be exactly obtained in closed form with a computational complexity of only $\mathcal{O}(n^3)$ operations (Van Gestel et al., 2003; Cawley & Talbot, 2004; Cawley, 2006).

4.2 Efficient leave-one-out computation for LS-SVM

In this section a review of the efficient leave-one-out computation for LS-SVM classifiers is presented. More details can be found in (Van Gestel et al., 2003; Cawley & Talbot, 2004; Cawley, 2006). The systems of linear equations given by (4) can be represented in the form of an arbitrary block structured matrix $\boldsymbol{A}$ as

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

where $\boldsymbol{a}_{12}$, $\boldsymbol{a}_{21}$ are column vectors and a_{22} is a scalar. According to (Van Gestel et al., 2003) and (Cawley, 2006), by exploiting the block structure on (4), the residual of the leave-one-out prediction for the i-th training pattern can be expressed in a closed form. Given the inverse of $\boldsymbol{A}$ and the model parameters $\boldsymbol{\alpha}$ solving (4) for the full training set are known, the residual of the leave-one-out prediction for the i-th training pattern is (Van Gestel et al., 2003; Cawley, 2006):

$$r_i^{(-i)} = y_i - \hat{y}_i^{(-i)} = \frac{\alpha_i}{(\boldsymbol{A}^{-1})_{ii}} \tag{13}$$

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 given in (4). Given the solution to the basic training algorithm, the complexity of this efficient leave-one-out computation reduces to $\mathcal{O}(n)$.

Moreover, we can readily notice that only the diagonal elements of $\boldsymbol{A}^{-1}$ need to be determined. Using the matrix inversion lemma of a block structure matrix (Golub & Van Loan, 1989), the inverse of $\boldsymbol{A}$ can be expressed in terms of $\boldsymbol{H}^{-1}$. The block inverse reads

$$\boldsymbol{A}^{-1} = \left[\begin{array}{c|c} \boldsymbol{H}^{-1} + \frac{1}{s}\boldsymbol{H}^{-1}\mathbf{1}\mathbf{1}^T\boldsymbol{H}^{-1} & -\frac{1}{s}\boldsymbol{H}^{-1}\mathbf{1} \\ \hline -\frac{1}{s}\mathbf{1}^T\boldsymbol{H}^{-1} & \frac{1}{s} \end{array}\right] \tag{14}$$

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 (13) and given that $\boldsymbol{H}^{-1} \in \mathbb{R}^{n\times n}$ is already known, then using the upper left block in (14) we get

$$(\boldsymbol{A}^{-1})_{ii} = (\boldsymbol{H}^{-1})_{ii} + \frac{\nu_i^2}{s},\ i = 1, \ldots, n \tag{15}$$

with $\boldsymbol{\nu} = \boldsymbol{H}^{-1}\mathbf{1},\ \boldsymbol{\nu} \in \mathbb{R}^{n\times 1}$. Alternatively, Eq. (15) can be defined in terms of the Cholesky factorization of $\boldsymbol{H} = \boldsymbol{L}\boldsymbol{L}^T$ (Cawley, 2006). In summary, the procedure to compute the efficient leave-one-out error is presented in Algorithm (1).

Up to this point, almost all the necessary elements to perform variable selection in our classification problem [2] have been clearly established. The heuristic search methods, forward and backward search, seem to fulfill the requirements of an efficient search space exploration. Yet their greedy scheme provides with a finite number of steps and the required termination condition for the search. Our learning algorithm, that is the LS-SVM classifiers, can be efficiently trained by solving just one system of linear equations. Additionally, a simple training of the LS-SVM is sufficient to obtain the leave-one-out (LOO) estimator with less computational burden than a naive implementation. Such leave-one-out criterion is used to evaluate whether potential variables must be included or discarded.

At every stage of the search (a node in Kohavi's tree (Kohavi & John, 1997)) or in other words, every single time a candidate variable is tested to be included (or removed) to (from) the set of variables, the learning algorithm must be solved. Consider that there are at most t nodes (stages) in the search space, then for the LS-SVM classifier the approximate complexity is $\mathcal{O}(tn^3)$. In fact, this is what makes wrapper based approaches computationally expensive.

In the work by (Guyon et al., 2002), the SVM is solved only once with the full set of variables. Although the selection criterion used is the change on the norm weight cost function, the support vectors $\boldsymbol{\alpha}$ remain fixed throughout the whole search. Such strategy reduces the complexity of such approach from n-QP problems to just one. Although efficient, such approach violates the optimality conditions and relies in a

[2] Extension to regression tasks is straightforward.

bound on the generalization performance.

In the next sections, the use of low rank modifications to the LS-SVM system is motivated as to exploit its structure and therefore reduce the complexity of the sequential variable selection.

5 Low rank modifications

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 $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is known, it may also be useful to know how the inverse changes upon addition of a matrix of small rank (Golub & Van Loan, 1989). This section motivates that in this particular situation, it is much more efficient to *update* the inverse of $\boldsymbol{A}$ than to generate it again from *scratch*.

We are particularly interested in the system of equations of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is $n \times n$ matrix and $\boldsymbol{b}$ is a n-vector. Once $\boldsymbol{x}$ has been computed, it is often necessary to solve the modified system $\tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$. Clearly, we should be able to modify $\boldsymbol{A}$ to obtain factors for $\tilde{\boldsymbol{A}}$, from which $\tilde{\boldsymbol{x}}$ may be computed as before. In this paper, special attention is devoted to one type of modification, in which $\tilde{\boldsymbol{A}}$ has the form $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{u}\boldsymbol{u}^T$, where $\boldsymbol{u}$ is a n-vector. The matrix $\boldsymbol{u}\boldsymbol{u}^T$ is a matrix of *rank-one* (Golub & Van Loan, 1989), and the problem is usually described as that of *updating* the factors of $\boldsymbol{A}$ following a *rank-one* modification. Such type of modifications have been extensively studied in (Gill, Golub, Murray, & Saunders, 1974).

There are mainly two methods that efficiently solve this problem. The first approach is based on the Sherman-Morrison-Woodbury (SMW) formula (Golub &

Van Loan, 1989) for a low rank update of the inverse of a matrix. The second approach is based on the *Cholesky* factorization of the system $\boldsymbol{Ax} = \boldsymbol{b}$. The Sherman-Morrison-Woodbury formula has been used widely in the context of interior point methods (IPM) for linear programming (LP). In that context, however, the method may run into numerical difficulties. The Cholesky factorization is somewhat more expensive than the SMW method, with two times the workload and about three times storage requirement of the SMW method (Dongarra, Bunch, Moler, & Stewart, 1979; Seeger, 2005). On the other hand, the second method is said to be numerically more stable (experimentally and theoretically) in the context of interior point methods for LP and quadratically constrained quadratic problems. The Cholesky product-form has been used also in the context of SVM by (Fine & Scheinberg, 2001) to reduce the computational complexity in the interior point method solving the SVM QP optimization and by (Shilton, Palaniswami, Ralph, & Chung, 2005) for incremental training of SVM. Also in (Seeger, 2005), the use of Cholesky decompositions is motivated for low rank modifications. The LINPACK (Dongarra et al., 1979) library provides with implementations for such type of updates.

5.1 SMW formula and rank-one modifications

It is of particular interest to update the representation of $\boldsymbol{M} \in \mathbb{R}^{n \times n}$, if it is modified by adding a symmetric low rank matrix $\boldsymbol{VV}^T, \boldsymbol{V} \in \mathbb{R}^{n \times q}, q < n$. This can be carried out by a classical method for a low-rank update of an inverse of a matrix, referred as the Sherman-Morrison-Woodbury (SMW) formula.

Suppose that a nonsingular matrix $\boldsymbol{H} \in \mathbb{R}^{n \times n}$, has a known inverse $\boldsymbol{H}^{-1}$. Consider also a modification of the form $\boldsymbol{M} = \boldsymbol{H} + \boldsymbol{URV}^T$, where $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{n \times q}$, $\boldsymbol{R} \in \mathbb{R}^{q \times q}$. If $\boldsymbol{M}$ is nonsingular, that is, both $\boldsymbol{H}$ and $\boldsymbol{URV}^T$ are also nonsingular, then

(Golub & Van Loan, 1989)

$$(\boldsymbol{H} + \boldsymbol{U}\boldsymbol{R}\boldsymbol{V}^T)^{-1} = \boldsymbol{H}^{-1} - \boldsymbol{H}^{-1}\boldsymbol{U}\boldsymbol{Z}^{-1}\boldsymbol{V}^T\boldsymbol{H}^{-1} \tag{16}$$

where $\boldsymbol{Z} = \boldsymbol{R}^{-1} + \boldsymbol{V}^T\boldsymbol{H}^{-1}\boldsymbol{U}$.

The motivation for this formula resides in the fact that if q is much smaller than n, then $\boldsymbol{R}$ and $\boldsymbol{Z}$ may be much easier to invert than $\boldsymbol{M}$. A more common case is the symmetric modification $\boldsymbol{M} = \boldsymbol{A} + \rho\boldsymbol{V}\boldsymbol{R}\boldsymbol{V}^T$ where either $\rho = +1$ (corresponding to an update) or $\rho = -1$ (corresponding to a downdate).

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

$$\left(\boldsymbol{H} + \rho\boldsymbol{u}\boldsymbol{v}^T\right)^{-1} = \boldsymbol{H}^{-1} - \rho\frac{\boldsymbol{H}^{-1}\boldsymbol{u}\boldsymbol{v}^T\boldsymbol{H}^{-1}}{1 + \boldsymbol{v}^T\boldsymbol{H}^{-1}\boldsymbol{u}} \tag{17}$$

These last two equations (16) and (17) show that the inverse of $\boldsymbol{M}$ can be computed from the inverse of $\boldsymbol{H}$. As above mentioned, this is simpler and cheaper to compute than starting from scratch. Particularly, Eq. (17) requires only simple matrix vector operations given that $\boldsymbol{H}^{-1}$ is already known. Therefore in the simplest case, $\tilde{\boldsymbol{H}} = \boldsymbol{H} + \boldsymbol{v}\boldsymbol{v}^T$ (positive rank-1, update) or $\tilde{\boldsymbol{H}} = \boldsymbol{H} - \boldsymbol{v}\boldsymbol{v}^T$ (negative rank-1, downdate). These cases are fundamentally different in that for a positive update $\tilde{\boldsymbol{H}} \succ \boldsymbol{0}$, as long as $\boldsymbol{H}$ is well-conditioned. On the other hand, negative downdates break down if $\boldsymbol{H}$ is not positive definite, and can result in large errors if $\boldsymbol{H}$ is close to singularity (Dongarra et al., 1979; Golub & Van Loan, 1989). Low rank updates of the form $\tilde{\boldsymbol{H}} = \boldsymbol{H} \pm \boldsymbol{V}\boldsymbol{V}^T, \boldsymbol{V} \in \mathbb{R}^{n\times q}$, can always be done by applying q updates/downdates sequentially (Seeger, 2005).

Let us to consider a general matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, being symmetric and positive definite. Then, for $\boldsymbol{A}$ exists a lower triangular matrix $\boldsymbol{L}$ with positive diagonal elements such that $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$. The matrix $\boldsymbol{L}$ is unique and is known as the Cholesky factorization of $\boldsymbol{A}$ (Golub & Van Loan, 1989). Several computations involving $\boldsymbol{A}^{-1}$ can be done equally fast using $\boldsymbol{L}$, and usually more numerically stable. For instance, the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, can be solved by means of two triangular systems as $\boldsymbol{L}\boldsymbol{t} = \boldsymbol{b}$, $\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{t}$. In fact, this is the best way of inverting $\boldsymbol{A}$.

Suppose that now we use a representation based on a Cholesky decomposition for the matrix $\boldsymbol{H} \in \mathbb{R}^{n \times n}$ as $\boldsymbol{H} = \boldsymbol{L}\boldsymbol{L}^T$, and we further consider the case where $\boldsymbol{H}$ is modified by a symmetric matrix of rank-one, i.e., we have $\tilde{\boldsymbol{H}} = \boldsymbol{H} + \boldsymbol{u}\boldsymbol{u}^T$. Assuming that the Cholesky factors of $\boldsymbol{H}$ are known, we are now interested in the factors of

$$\tilde{\boldsymbol{H}} = \tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^T \tag{18}$$

The method that calculates $\tilde{\boldsymbol{L}}$ from $\boldsymbol{L}$ and $\boldsymbol{u}$ is called a *rank-one* update algorithm. One way of performing such update is by applying a series of orthogonal *Givens* rotations of the form

$$\boldsymbol{G}_{n-1} \dots \boldsymbol{G}_k \dots \boldsymbol{G}_1 \left[\boldsymbol{u}, \boldsymbol{L}\right]^T = \left[\boldsymbol{0}, \tilde{\boldsymbol{L}}\right]^T, \tag{19}$$

leading to $\tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^T = \boldsymbol{L}\boldsymbol{L}^T + \boldsymbol{u}\boldsymbol{u}^T$. The algorithm proceeds $k = 1, \dots, n-1$. The matrix $\boldsymbol{G}_k$ is chosen based on the k-th column, namely $\boldsymbol{G}_k[\boldsymbol{u}_k, \boldsymbol{L}_{kk}]^T = [0, \tilde{\boldsymbol{L}}_{kk}]^T$. Appropriate algorithms can be found in standard texts (Golub & Van Loan, 1989; Gill et al., 1974) and software packages such as the LINPACK library (Dongarra et al., 1979). Additionally, the routine `cholupdate` in MATLAB also performs such operation. Factorizations of this form are also valid for the QR decomposition.

In the next section we will concentrate our attention to the use of the SMW method and its application to the LS-SVM classifiers. The reason behind this lies on the simplicity of the algorithms to form and consequently update the LS-SVM classifier. The Cholesky update counterparts are more complex, but are equally applicable to the LS-SVM classifier with respect to a rank-one modification. Moreover the matrix to be modified, *i.e.* $\boldsymbol{H}^{-1} = (\boldsymbol{K} + \gamma^{-1}\boldsymbol{I}_n)^{-1}$ can be computed upon adequate selection of γ. Therefore, $\boldsymbol{H}$ is well conditioned and positive definite (Golub & Van Loan, 1989).

6 Low rank updated LS-SVM

In this section, we describe how to perform low rank modifications to the system of linear equations (4) for the LS-SVM classifier. In fact, we refer to modifications involving $\boldsymbol{H}$ or its corresponding inverse $\boldsymbol{H}^{-1}$. Given that $\boldsymbol{H}$ is positive-definite and therefore well conditioned, we may initially not run into any numerical instability. To ensure positive-definiteness of $\boldsymbol{H}$, we rely on an adequate selection of the hyper-parameter γ.

Recalling the basic mechanism of *wrapper* search methods described earlier on section 2, it is clear that a repetitive evaluation of the learning machine would be required each time a new variable is tested. For instance, in the LS-SVM classifier, the calculation of $\boldsymbol{H}^{-1}$ for each candidate would be demanded. Such alternative becomes prohibitive for large data sets with high number of variables. However, there exists a way to circumvent such repetitive matrix inversions in the case of a *linear kernel*.

Let the given training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n},\ \boldsymbol{x}_i \in \mathbb{R}^d$ be described by the set of

variables $\mathcal{U} = \{\boldsymbol{u}_k\}_{k=1}^{d}, \boldsymbol{u}_k \in \mathbb{R}^{n\times 1}$, and given that $\boldsymbol{H}^{-1}$ is known and positive definite, $\boldsymbol{H} = \boldsymbol{K} + \gamma^1 \boldsymbol{I}_n$, for a linear kernel we can write

$$\begin{aligned}\boldsymbol{H} &= \boldsymbol{K} + \gamma^{-1}\boldsymbol{I}_n \\ &= \sum_{k=1}^{d} \boldsymbol{u}_k \boldsymbol{u}_k^T + \gamma^{-1}\boldsymbol{I}_n \end{aligned} \tag{20}$$

Splitting the sum at the level of variable k, and grouping back the first two terms, then we have

$$\begin{aligned}\boldsymbol{H}_k &= \sum_{j=1}^{k-1} \boldsymbol{u}_j \boldsymbol{u}_j^T + \gamma^{-1}\boldsymbol{I}_n + \boldsymbol{u}_k \boldsymbol{u}_k^T \\ \boldsymbol{H}_k &= \boldsymbol{H}_{k-1} + \boldsymbol{u}_k \boldsymbol{u}_k^T \end{aligned} \tag{21}$$

Direct application of Eq. (17) to Eq. (21) results in

$$\boldsymbol{H}_k^{-1} = \boldsymbol{H}_{k-1}^{-1} - \frac{\boldsymbol{H}_{k-1}^{-1}\boldsymbol{u}_k \boldsymbol{u}_k^T \boldsymbol{H}_{k-1}^{-1}}{1 + \boldsymbol{u}_k^T \boldsymbol{H}_{k-1}^{-1}\boldsymbol{u}_k}. \tag{22}$$

The previous set of equations explain us that, for a linear kernel and a given variable $\boldsymbol{u}_k$ to be tested for inclusion, the inverse of $\boldsymbol{H}^{-1}$ can be updated from the inverse of the previous step. Combination of this rank-one update procedure with the leave-one-out algorithm given in section 4, provides an efficient algorithm for variable selection. With the use of such *low rank* modifications, algorithms for *forward* and *backward* search can be devised by direct application of *updates* and *downdates* operations respectively. There is no computation cost incurred other than matrix vector multiplications. Moreover, the updated matrix at each step, can be immediately used to compute the efficient leave-one-out error outlined in Algorithm 1.

6.1 Forward update

In forward selection, we start from the empty set of variables. In the case of LS-SVM this translates into $\boldsymbol{H}_0 = \gamma^{-1}\boldsymbol{I}_n$, and so the initial inverse being $\boldsymbol{H}_0^{-1} = \gamma\boldsymbol{I}_n$. Therefore, we keep updating this matrix when testing the variable $\boldsymbol{u}_k$ for inclusion. Notice that no matrix inversions are required at all since the by-product of the rank-one update is directly used to compute the leave-one-out error. In short, the goal is to compute $\boldsymbol{H}_k^{-1}$ at selection step k from $\boldsymbol{H}_{k-1}^{-1}$ at step $k-1$, upon addition of variable u_k.

$$\boldsymbol{H}_k^{-1} = \boldsymbol{H}_{k-1}^{-1} - \frac{\boldsymbol{H}_{k-1}^{-1}\boldsymbol{u}_k\boldsymbol{u}_k^T\boldsymbol{H}_{k-1}^{-1}}{1 + \boldsymbol{u}_k^T\boldsymbol{H}_{k-1}^{-1}\boldsymbol{u}_k} \tag{23}$$

6.2 Backward downdate

Along the same lines of the *forward update* algorithm, and using the elements we have described in the previous sections, we can come up with an efficient algorithm for backward elimination of variables. Since we start from the full set of all variables, the inverse of the matrix $\boldsymbol{H}$ is needed just once to initialize the algorithm. Afterwards, at every elimination step the inverse $\boldsymbol{H}^{-1}$, is *downdated*. The basic idea is to compute $\boldsymbol{H}_k^{-1}$ at elimination step k, from $\boldsymbol{H}_{k+1}^{-1}$ at step $k+1$, upon removal of variable u_k, that is

$$\boldsymbol{H}_k^{-1} = \boldsymbol{H}_{k+1}^{-1} + \frac{\boldsymbol{H}_{k+1}^{-1}\boldsymbol{u}_k\boldsymbol{u}_k^T\boldsymbol{H}_{k+1}^{-1}}{1 - \boldsymbol{u}_k^T\boldsymbol{H}_{k+1}^{-1}\boldsymbol{u}_k} \tag{24}$$

However, the formulations given in this section are restricted to the case of the *linear kernel*. The rank-one modifications expressions are only valid in such situation. The use of a non-linear kernel strictly requires of the computation of $\boldsymbol{H}^{-1}$ at each selection step.

The algorithm for forward updates is summarized in Algorithm 2. The backward downdate version possesses (see Algorithm 3) a similar structure. In the following section, we present results of these two algorithms when applied on a series of benchmark data sets provided in (Rätsch, Onoda, & Müller, 2001). The computational efficiency is compared with respect to existing approaches for variable selection.

7 Empirical Results

In this section, experimental results demonstrate the benefits on the use of low rank updates for LS-SVM classifiers. This approach overcomes, and reduces, the high computational cost incurred during variable selection. Table (1) shows a selection of seven public domain benchmark data sets used in (Rätsch et al., 2001). For six out of seven data sets, one hundred random partitions of training and test sets are available. The `Splice` data set contains only twenty. Linear and RBF kernels are considered in combination with the two search algorithms: forward and backward search. The use of low rank updates/downdates is only applicable for the case of linear kernel. The selection criterion is based on the PRESS statistic derived from the efficient leave-one-out estimator. Additionally, results in two public available microarray data are also included.

7.1 Computational performance

First, we compare the computational cost incurred when using either of the two proposed algorithms on the same data set. We compare Algorithms 2 and 3 on a balanced two-class synthetic data set with five hundred samples and one thousand

variables, that is $\{n = 500, d = 1000\}$. The main computational difference between both approaches is the *initial inverse matrix* $\boldsymbol{H}_0^{-1}$. Therefore, this $n \times n$, $n = 500$, kernel matrix needs to be first computed and then inverted in the *rank-one downdates* algorithm (Refer to line 3 in Algorithm 3). Clearly, such situation harms the performance of the rank-one *downdate* algorithm over its *update* counterpart. Fig. 1 shows the computational time for both algorithms with respect to the number of variables. The gap between both algorithms clearly appears in the very first iteration, which represents the effect of the initialization scheme.

Additionally, to assess the computational complexity of the variable selection problem and to show the improvements especially in the use of low rank updates, for this we consider two experiments. First, the `Splice` data set is used to see the impact of having a moderate number or samples (1000 training) and a reasonable number of variables (60 variables). The second experiment constitutes the opposite case, that is a very small number of samples and a large number of variables. For this purpose the Colon cancer data set (Alon et al., 1999) is employed. It is clear that the use of a wrapper method pose difficulties in both the number of samples and the number of variables of the data set under consideration. The computational performance on each of the examples mentioned is shown in Fig. 2 and Fig. 3. In the `Splice` data set we want to see the effect of inverting a reasonable big matrix, therefore we consider a naive implementation where all the required matrix inversions are performed. The inversion at the first iteration already pushes up the computational load. A dramatic reduction is immediately appreciated when using the low rank updated algorithm even for selecting a small number of variables. In the Colon cancer data set, the efficiency of the rank-update algorithm, is compared with other two variants that also employ the LS-SVM classifier. The first algorithm presented in (Tang et al., 2006), uses a closed-form to compute the leave-one-out

error. Matrix inversions are, however, still required to solve the LS-SVM system for each new potential subset of variables. The second approach, given in (Zhou & Mao, 2005), makes use of a span-based (Vapnik & Chapelle, 2000) upper bound on the leave-one-out error. The expensive part of the algorithm in (Zhou & Mao, 2005) is mainly the computation of a $n \times n$ distance matrix each time a new subset of variables is tested. We stop the three algorithms when 200 out of the 2000 variables have been selected in the Colon cancer dat set and compare the computational time. Clearly, it can be drawn that the proposed algorithm scales very nicely even for high dimensional data sets. The time gap with respect to the approach in (Tang et al., 2006) is readily appreciable (around two orders of magnitude). When compared to the bound approach (Zhou & Mao, 2005), our proposed method is still faster, despite we compute an exact form of leave-one-out error.

We also test the *downdate* version algorithm with the Colon data. The comparison is made with respect to the two variants of the SVM-RFE algorithm (Guyon et al., 2002). The first variant (SVM-RFE1), computes the support vectors at each step of the elimination phase (retraining), whereas in the second variant (SVM-RFE2) they are fixed as from the first iteration (no retraining) . Fig. 4 shows the computational required in the elimination of 500 variables out of 2000. The quadratic programming problem (QP) involved in the standard SVM, as mentioned earlier in section 3, is more computationally expensive than the LS-SVM solution. The sequential nature of the backward search increases the volume of computations. The proposed backward approach is even faster than the SVM-RFE2, which avoids a considerable number of operations by keeping support vectors fixed throughout the whole elimination phase.

The combination of the low rank updates and the closed-form expression of the leave-one-out error clearly benefits the use of the LS-SVM classifiers as a potential

algorithm for the task of variable selection. It is important to emphasize that no matrix inversions are required when the linear kernel is used in combination with the low rank update modification. The whole solution simply uses matrix vector operations. Similar results are obtained for the linear kernel in combination with the backward downdate.

7.2 Prediction performance

7.2.1 Benchmarking data

The final performance of any of the possible strategies (and combinations) is heavily subjected to the hyperparameter selection, i.e. a correct choice for $\theta = (\gamma, \sigma)$ when using an RBF kernel or simply γ for the linear kernel. In this work, the hyperparameters are tuned as to minimize the PRESS statistic. For each of the 100 realizations of the data sets (20 for the `Splice` data set), we select the pair (γ^*, σ^*) minimizing the leave-one-out PRESS statistic. Such pair is then used during the variable selection procedure. Tables 2 and 3 show the performance of the LS-SVM with respect to the error rate over the corresponding number of realization per data set. The basic LS-SVM classifiers with no variable selection, but appropriate hyperparameter tuning, constitutes the baseline classifier. Linear kernel and RBF kernel are used.

In general terms, there seems to be little difference in the performances regardless the search methodology chosen. Both forward and backward search methods provide very similar results across all data sets when combined either with the linear kernel or the RBF kernel. The number of selected variables tends to be the same number as well. The computational cost, however, when combined with lin-

ear kernels is significantly several orders of magnitude lower thanks to the *low rank* modifications introduced. Additionally, in three of the data sets, namely: `Breast`, `Diabetes` and `Flare solar`, there is apparently no gain in performance with respect to the baseline LS-SVM. There is, however, a consistent tendency of the other LS-SVM variants to select a lower number of variables with no substantial increase of the test error. Since the number of parameters is relatively small in all of these three data sets and the sample size is not that large, the search space appears to be well covered. We show in Fig. 5 the test error on the `Flare` data set. Boxplots are generated from the available 100 realizations, explaining an almost constant variance remains with median value clearly low at two variables. The low rank downdate algorithm was used.

On the other hand, for the `Splice` data set, the RBF kernel performs significantly better than any other alternative. Additionally, the RBF kernel tends to select less variables. Such an improvement, however, is attained at a much higher computational cost mainly because of the larger number of patterns (1000 training patterns). The low ranked LS-SVM with linear kernel is obviously faster than the RBF kernel setting, but produces slightly higher error when using the top-selected 28 variables. Indeed, Fig. 6 explains, in terms of the test error, our previous comments on the `Splice` data set.

Another interesting example is the `Heart` data set. This data set tends to produce the same results regardless of the method used. In fact the basic LS-SVM, with no variable selection, appears more consistent for this data set. A plausible explanation could be the low sample size of this data set, thus a high risk of overfitting is present.

7.2.2 Microarray data

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 (Ambroise & McLachaln, 2002), is applied to the training set. 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 consistently selected in different folds comprise the set used to compute the test error.

Fig. 7, shows the test error (with respect to the number of selected variables) of the proposed low rank update algorithm when applied to the Leukemia data set. Additionally, we make comparisons with respect to the methods for variable selection cited earlier in (Zhou & Mao, 2005) and (Tang et al., 2006). Our proposed algorithm exhibits better stability than the bound (Zhou & Mao, 2005) approach. The tightness of such bound is heavily dependant on the stability of the LS-SVM to perturbations in the training data. The exact computation of the leave-one-out has therefore a direct 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.

The low rank downdate algorithm is compared using the Colon cancer data set. We again use the two SVM-RFE algorithms (Guyon et al., 2002). The test error for 1000 out of 2000 selected variables is shown in Fig. 8. The use of the leave-one-out error is clearly superior to the margin criterion used in the SVM-RFE algorithm for eliminating variables. This was expected since the margin is a bound on the generalization error. In addition, the test error is remarkably lower around 200 variables. The latter observation suggest of a more compact and accurate representation of

the Leukemia data.

7.3 Model selection

In the experiments reported in earlier sections, we tuned the hyperparameters with the full set of variables and leave them unmodified during the variable selection process. The main reason for this was, in principle, to avoid more degrees of freedom within the variable selection loop, as well as to reduce the computational load. However, performing a careful adjustment of the hyperparameters, once a variable is definitely added or removed to the final set, may result in a more robust and consistent set of variables. Instead of considering a grid search approach, we chose for a direct optimization of the PRESS statistic with respect to the LS-SVM parameters $\theta = (\gamma, \sigma)$ (or simply γ for a linear kernel). Since the PRESS statistic is a continuous function, we can make use of standard simplex-search algorithm, such as the MATLAB routine `fminunc`. An adequate practice when using unconstrained optimization routines, is that of transforming the hyperparameters into a logarithmic scale. Another alternative to ensure correct values for the parameters, e.g. $\gamma \geq 0$, is the `fmincon` function with the additional non-negative constraint for θ. Additionally, extreme values for the hyperparameters may introduce instabilities and singularities into the kernel matrix, and consequently numerical problems in the low rank updates/downdates. Alternative to the use of the PRESS statistic, (Cawley & Talbot, 2007) propose to add a regularization/penalization term to prevent overfitting and perform a subsequent optimization within a Bayesian framework. Instead, we simply add a second term to the PRESS statistic that accounts for the norm of the hyperparameters

$$\mathrm{error}_{\mathrm{pressreg}} = \frac{1}{n}\sum_{i=1}^{n}\left(r_i^{(-i)}\right)^2 + \frac{1}{2}\sum_{j=1}^{n_\theta}\theta_j^2 \tag{25}$$

where n_θ denotes the number of parameters. We test the modified PRESS criterion in the `Heart` data set. The reduced number of training patterns of the `Heart` data set supports it as a good example. In Fig. 9 and 10, the test error box plots for the `Heart` data with the PRESS statistic and the modified version are shown respectively. Both results were obtained using the LS-SVM with RBF kernel and the forward search. Although the true error seems to increase as we add more and more variables, the modified PRESS estimator did capture the trend of the true error. The sharp point, in Fig. 10, of the error is a clear indicator of a good candidate model with less number of variables.

8 Conclusion

In this paper, the application of *low rank updates/downdates* to the structure (linear kernel) of the LS-SVM classifiers was found to be very suitable for variable selection wrapper-like algorithms. By exploiting the inherent block structure of the LS-SVM solution, an efficient method for computing the leave-one-out enabled us to evaluate the performance of potentially relevant subsets of variables. Furthermore, in the special case of the *linear kernel*, algorithms for fast *forward* and *backward search* are devised such that *matrix inversions* are at most completely avoided. Our proposed methodology proceeds then by *low ranking* the LS-SVM classifiers every time a variable is tested for inclusion (or removal) into the set of candidate variables. Such strategy constitutes a major improvement in the efficiency of the algorithms presented here, bringing in addition good numerical stability. In particular matrix inversions are avoided at any cost in the low *rank update* algorithm, while the *rank downdate* version requires only one. Experiments on microarray data provide us with evidence of the particular usefulness of our methods

in high dimensional scenarios. Moreover, the use of a continuous function in the leave-one-out estimator, the PRESS statistic, enables us to use optimization routines in the model selection, contrary to grid search approaches. Yet, alternatives to the leave-one-out estimator based on continuous functions may provide more refined sets of variables, especially for data sets with reduced number of samples. Comparison with other related algorithms delivered satisfactory results in terms of efficiency and generalization performance. Finally, we also give some hints on how traditional matrix factorization techniques of positive-definite matrices may further be considered within our framework.

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

Alon, U., Barkai, N., Notterman, D. A., Gish, K., Ybarra, S., Mack, D., et al. (1999). Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proc Natl Acad Sci*, *96*(12), 6745–6750.

Ambroise, C., & McLachaln, G. J. (2002). Selection bias in gene extraction on the basis of microarray gene-expression data. *PNAS*, *99*(10), 6562–6566.

Bishop, C. (1995). *Neural networks for pattern recognition*. Oxford UP.

Cawley, G. C. (2006). Leave-one-out cross-validation based model selection criteria for weighted LS-SVMs. In *Proc of the international joint conference on computational neural networks (ijcnn'06).*

Cawley, G. C., & Talbot, N. L. (2004). Fast exact leave-one-out cross-validation of sparse least-squares support vector machines. *Neural Networks*, *17*, 1467–1475.

Cawley, G. C., & Talbot, N. L. C. (2007). Preventing over-fitting in model selection via bayesian regularisation of the hyper-parameters. *Journal of Machine Learning Research*, *8*, 841-861.

Dongarra, J., Bunch, J., Moler, C., & Stewart, G. (1979). Linpack users' guide. *SIAM, Philadelphia*.

Fine, S., & Scheinberg, K. (2001). Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, *2*, 243–264.

Gill, P. E., Golub, G. H., Murray, W., & Saunders, M. A. (1974, Apr.). Methods for modifying matrix factorizations. *Mathematics of Computation*, *28*(126), 505–535.

Golub, G. H., & Van Loan, C. F. (1989). *Matrix computations* (2nd. ed.). The

Jonhs Hopkins University Press.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, *46*.

Kohavi, R., & John, G. (1997). Wrappers for feature selection. *Artificial Intelligence*, *97*, 273–324.

Lachenbruch, P. (1967). An almost unbiased method for the probability of misclassification in discriminant analysis. *Biometrics*, *23*, 639-645.

Ojeda, F., Suykens, J. A. K., & De Moor, B. (2007). Variable selection by rank-one updates for least squares support vector machines. In *Proceedings of the international joint conference on neural networks (ijcnn'07).*

Rätsch, G., Onoda, T., & Müller, K.-R. (2001). Soft margins for AdaBoost. *Machine Learning*, *42*(3), 287–320.

Saunders, G., Gammerman, A., & Vovk, V. (1998). Ridge regression learning algorithm in dual variables. In *Proc. 15th international conf. on machine learning* (pp. 515–521). Morgan Kaufmann, San Francisco, CA.

Seeger, M. (2005). *Low rank updates for the cholesky decomposition.* Technical Report. Max Planck Society, Tuebingen, Germany.

Shilton, A., Palaniswami, M., Ralph, D., & Chung, A. (2005). Incremental training of support vector machines. *IEEE Transactions on Neural Networks*, *16*(1), 114–131.

Suykens, J. A. K., Van Gestel, T., De Brabanter, J., De Moor, B., & Vandewalle, J. (2002). *Least squares support vector machines*. Singapore: World Scientific.

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

Tang, E. K., Suganthan, P., & Yao, X. (2006). Gene selection for microarray data based on least squares support vector machine. *BMC Bioinformatics*, *7*(95).

Van Gestel, T., Baesens, B., Suykens, J. A. K., Espinoza, M., Baestaens, D., Van-

thienen, J., et al. (2003). Bankruptcy prediction with least squares support vector machine classifiers. In *International conference on computational intelligence for financial enigeering (cifer'03)* (Vol. 1).

Van Gestel, T., Suykens, J. A. K., Baesens, B., Viaene, S., Vanthienen, J., Dedene, G., et al. (2004). Benchmarking least squares support vector machine classifiers. *Machine Learning*(1), 5–32.

Vapnik, V. (1998). *Statistical learning theory*. Wiley.

Vapnik, V., & Chapelle, O. (2000). Bounds on error expectation for support vector machines. *Neural Computation*, *12*(9), 2013-2036.

Xiong, M., Fang, X., & Zhao, J. (2001). Biomarker identification by feature wrappers. *Genome Research*, *11*, 1878–1887.

Zhou, X., & Mao, K. (2005). LS bound based gene selection for DNA microarray data. *Bioinformatics*, *21*(8).

Captions

Algorithm 1 Efficient leave-one-out for LS-SVM classifiers. The leave-one-out (LOO) error is obtained as a by-product of just one run of the LS-SVM classifier.

Algorithm 2 Algorithm for fast variable selection using low rank updates. We apply *rank-one* updates into the linear kernel matrix, to obtain a fast sequential forward selection (SFS) algorithm. Such modification avoids completely the use of matrix inversion type of operations.

Algorithm 3 Algorithm for fast variable selection using low rank downdates. We apply *rank-one* downdates into the linear kernel matrix, to obtain a fast sequential backward elimination (SBE) algorithm. In this algorithm we required of the

inverse of the matrix $\boldsymbol{H}$ just one time. Downdates are performed iteratively on this matrix.

Table 1 Data sets used for the benchmarking study.

Table 2 Error rates of LS-SVM classifier with a linear kernel. We apply *rank-one updates* into the linear kernel matrix to perform fast sequential forward selection (SFS) of variables. Similarly *rank-one downdates* into the linear kernel matrix allows for fast sequential backward elimination (SBE) of variables. The results for each method correspond to the mean error rate over test data for 100 realizations of each data set (20 for the splice data set). Whenever a lower number variables can be selected without degrading the performance of the classifier, is indicated between squared brackets.

Table 3 Error rates of LS-SVM classifier with RBF kernel. The purpose of this experiment is for completeness. Although we can not apply low rank modifications as in the case of the linear kernel, it is interesting to see how does the RBF kernel behaves in variable selection problems. To implement sequential forward selection (SFS) and sequential backward elimination (SBE) with the RBF kernel we do require of matrix inversions at every stage. The results for each method correspond to the mean error rate over test data for 100 realizations of each data set (20 for the splice data set). Whenever a lower number variables can be selected without degrading the performance of the classifier, is indicated between squared brackets.

Figure 1 Computational time for the *update* (Solid line) and *downdate* (Dashed line) algorithms on a balanced two-class synthetic data set with five hundred samples and one thousand variables, that is $\{n = 500, d = 1000\}$. The main computational difference between both approaches is the *initial inverse matrix*

$\boldsymbol{H}_0^{-1}$. The gap between both algorithms clearly appears in the very first iteration, which represents the effect of the initialization scheme. In the *downdate* algorithm we require of computing first the matrix $\boldsymbol{H} = \boldsymbol{K} + \boldsymbol{I}_n$, and its associated inverse; whereas in the *update* algorithm we start simply from the diagonal matrix $\boldsymbol{H} = \gamma^{-1}\boldsymbol{I}_n$.

Figure 2 Computational time of the low *rank update* (Solid line) algorithm when compared to a naive implementation (Dashed line) where all matrix inversions are performed. The simulations are done with the `Splice` data set. Although the number of variables is 60, we can immediately see a dramatic reduction in the time needed to select even a small number of variables.

Figure 3 Computational time for the low *rank update* algorithm ($\star$) when compared to other two variable selection algorithms also based on the LS-SVM classifier. The first algorithm ($\circ$) by (Tang et al., 2006) requires of matrix inversions during the variable selection phase. The second variant ($\square$) proposed by (Zhou & Mao, 2005) makes use of an upper bound on the leave-one-out error but requires of an $n \times n$ distance matrix to be computed at every selection step. We test the three algorithms in the Colon cancer data set and show their performance to select 200 out of a total of 2000 variables. The gap with respect to the approach in (Tang et al., 2006) is readily appreciable (around two orders of magnitude). When compared to the bound approach (Zhou & Mao, 2005), our proposed method is still faster, despite we compute an exact form of leave-one-out error.

Figure 4 Computational time for the low *rank downdate* algorithm ($\star$) when compared with two SVM-RFE ((Guyon et al., 2002)) based algorithms. The first variant, SVM-RFE1 ($\circ$), computes the support vectors at each step of the elimination phase (retraining), whereas in the second variant,SVM-RFE2 ($\square$),

support vectors are fixed as from the first iteration (no retraining). We stop the algorithm when 500 variables of a total of 2000 are removed in the Colon cancer data set. Our proposed approach is even faster than the SVM-RFE2, which avoids a considerable number of operations by keeping fixed the support vectors through the whole elimination phase.

Figure 5 `Flare` data set. Boxplots of test error averaged over 100 realizations. The low rank downdate algorithm was used.

Figure 6 `Splice` data set test error averaged over 20 realizations. (*Solid*) Linear kernel with low rank downdates. (*Dashed*) RBF kernel with explicit matrix inversion.

Figure 7 Test error on the Colon data set. We compare the performance of our low rank update algorithm (Solid) with the algorithm by (Zhou & Mao, 2005) (Dashed). Error bars shows the variance of the external 10-fold cross-validation. We stop both algorithms when 100 variables out of 2000 were selected.

Figure 8 Test error on the Leukemia data set. We compare the low *rank downdate* algorithm ($\star$) with the two SVM-RFE ((Guyon et al., 2002)) based algorithms: SVM-RFE1 ($\circ$) and SVM-RFE2 ($\square$). Test error is shown when thousand variables (out of 2000) have been removed. The use of the leave-one-out error is clearly superior to the margin criterion used in the SVM-RFE algorithm for eliminating variables.

Figure 9 Boxplot of the `Heart` data set. Test error is averaged over 100 realizations. We use the LS-SVM with RBF kernel in a forward selection search. Hyperparameters (σ, γ) are tuned with the PRESS statistic.

Figure 10 Boxplot of the `Heart` data set. Test error (solid line) is averaged over

100 realizations. We use the LS-SVM with RBF kernel. Hyperparameters (σ, γ) are tuned with the modified PRESS statistic. We additionally show the Error estimation (*Dashed*).

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

1: Construct kernel matrix and obtain $\boldsymbol{H}^{-1}$.

2: Solve for b and $\boldsymbol{\alpha}$ using Eq. (7).

3: Compute diagonal elements $(\boldsymbol{A}^{-1})_{ii}$ with Eq. (15).

4: Compute prediction residuals using Eq. (13).

Algorithm 2 Algorithm for fast variable selection using low rank updates

Require: X, Y: training set; n: number of samples; d: number of variables.; p: number of variables to select.

Ensure: Selected variables $S \neq \emptyset$.

$S = \emptyset$ {final set of selected variables}
$P = \{X_1, \ldots, X_k, \ldots, X_d\},\ k = 1, \ldots, d$. {full set of variables}
$\boldsymbol{H}^{-1} \leftarrow \gamma \times \boldsymbol{I}_n$ {Initial inverse when no variables}
for $k = 1$ to t **do**
  $loo \leftarrow \infty$
  $c = \mathrm{card}\{P\}$ {Variables yet to evaluate}
  for $l = 1$ to c **do**
    $\boldsymbol{H}_l^{-1} \leftarrow \boldsymbol{H}^{-1} - (\boldsymbol{H}^{-1} X_l X_l^T \boldsymbol{H}^{-1})/(1 + X_l^T \boldsymbol{H}^{-1} X_l)$ {Update inverse}
    Solve for b_l , $\boldsymbol{\alpha}_l$ using $\boldsymbol{H}_l^{-1}$ and (7). {Solve LS-SVM system}
    Compute loo_l using first (15), then (13) and (9). {Fast leave-one-out}
    if $loo_l < loo$ **then**
      {Save the loo_l, variable l and $\boldsymbol{H}_l^{-1}$ for next comparison}
      $loo \leftarrow loo_l$
      $l^* \leftarrow l$
      $\boldsymbol{H}^{-1*} \leftarrow \boldsymbol{H}_l^{-1}$
    end if
  end for
  $\boldsymbol{H}^{-1} \leftarrow \boldsymbol{H}^{-1*}$ {Store inverse for next selection procedure}
  $S \leftarrow S \cup X_{l^*}$ {Update selected variables}
  $P \leftarrow P \setminus X_{l^*}$ {Remove selected variable from the full set}
end for
return S

Algorithm 3 Algorithm for fast variable selection using low rank downdates

Require: X, Y: training set; n: number of samples; d: number of variables.

Ensure: $S \neq \emptyset$: selected variables.

$S = \{X_1, \ldots, X_k, \ldots, X_d\},\ k = 1, \ldots, d$. {full set of candidates}

$P = \emptyset$. {variables to remove}

Compute $\boldsymbol{H}^{-1} = (\boldsymbol{K} + \gamma^{-1}\boldsymbol{I}_n)^{-1}$ with all variables. {Initial inverse}

while $S \neq \emptyset$ **do**

  $c = \mathrm{card}\{S\}$ {Variables yet to evaluate}

  for $l = 1$ to c **do**

    $\boldsymbol{H}_l^{-1} \leftarrow \boldsymbol{H}^{-1} + (\boldsymbol{H}^{-1}X_l X_l^T \boldsymbol{H}^{-1})/(1 - X_l^T \boldsymbol{H}^{-1} X_l)$ {Downdate inverse}

    Solve for b_l , $\boldsymbol{\alpha}_l$ using $\boldsymbol{H}_l^{-1}$ and (7). {Solve LS-SVM system}

    Compute loo_l using first (15), then (13) and (9). {Fast leave-one-out}

  end for

  $loo^* \leftarrow \min loo_l, l = 1, \ldots, c$ {Lowest LOO error}

  $X_* \leftarrow \arg\min loo_l,\ l = 1, \ldots, c$ {Variable to be definitely removed}

  $\boldsymbol{H}^{-1} \leftarrow \boldsymbol{H}^{-1} + (\boldsymbol{H}^{-1}X_* X_*^T \boldsymbol{H}^{-1})/(1 - X_*^T \boldsymbol{H}^{-1} X_*)$ {Downdate inverse with respect to X_*}

  $S \leftarrow S \setminus X_*$ {Remove selected variable from the full set}

  $P \leftarrow P \cup X_*$ {Add removed variable to set of removed variables}

end while

return P {Sequence of removed variables}

Tables

Table 1

Data set	Training samples	Test samples	Number of realizations	Input variables
Breast cancer	200	77	100	9
Diabetes	468	300	100	8
Flare solar	666	400	100	9
German	700	300	100	20
Heart	170	100	100	13
Splice	1000	2175	20	60
Waveform	400	4600	100	21

Table 2

Data set	LS-SVM	LS-SVM+SFS	LS-SVM+SBE
Breast cancer	0.268 (0.044)	0.268 (0.052) [2/9]	0.264 (0.049) [2/9]
Diabetes	0.221 (0.013)	0.234 (0.017) [6/8]	0.234 (0.017) [6/8]
Flare solar	0.334 (0.014)	0.329 (0.018) [2/9]	0.326 (0.019) [2/9]
German	0.252 (0.020)	0.245 (0.021)	0.242 (0.022) [16/20]
Heart	0.151 (0.026)	0.157 (0.032)	0.157 (0.032)
Splice	0.161 (0.007)	0.161 (0.006) [27/60]	0.161 (0.007) [28/60]
Waveform	0.133 (0.006)	0.132 (0.006)	0.147 (0.009)

Table 3

Data set	LS-SVM	LS-SVM+SFS	LS-SVM+SBE
Breast cancer	0.258 (0.064)	0.250 (0.047) [2/9]	0.258 (0.047) [2/9]
Diabetes	0.235 (0.015)	0.233 (0.019) [7/8]	0.233 (0.019) [7/8]
Flare solar	0.344 (0.011)	0.326 (0.019) [2/9]	0.326 (0.019) [2/9]
German	0.232 (0.025)	0.235 (0.022)	0.235 (0.022)
Heart	0.155 (0.034)	0.167 (0.034)	0.167 (0.034)
Splice	0.107 (0.007)	0.075 (0.006) [15/60]	0.077 (0.006) [14/60]
Waveform	0.096 (0.004)	0.099 (0.005) [16/21]	0.098 (0.004)

Figures

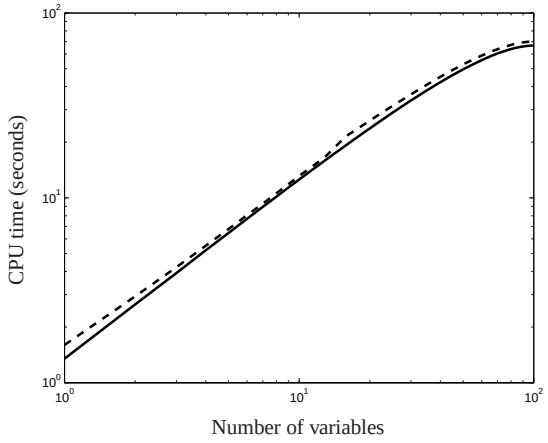

Fig. 1.

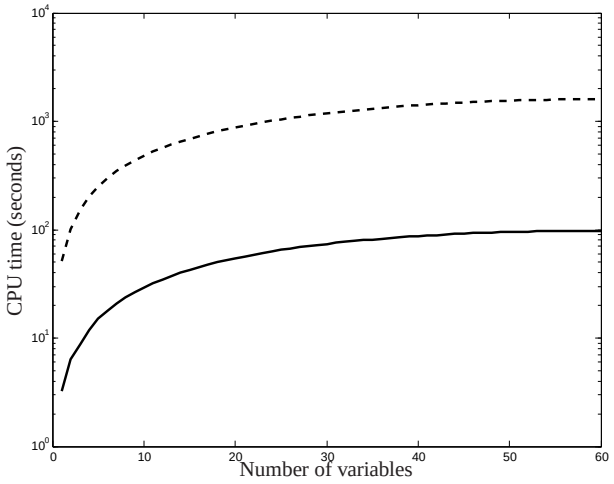
CPU time (seconds)
Number of variables

Fig. 2.

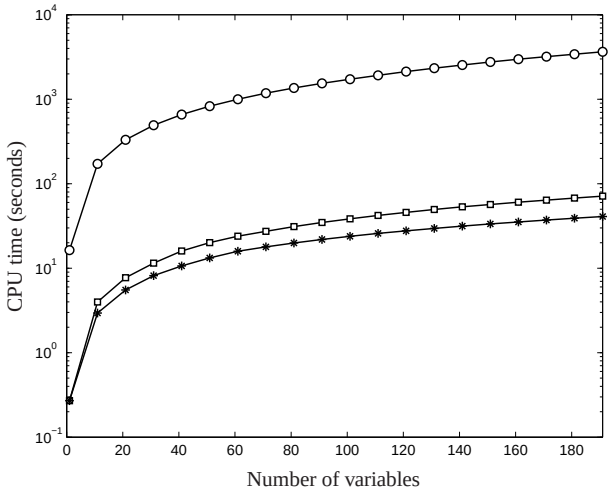
CPU time (seconds)
Number of variables

Fig. 3.

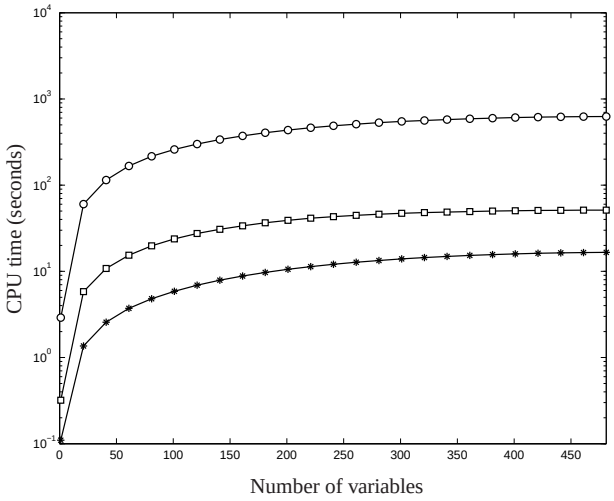

Fig. 4.

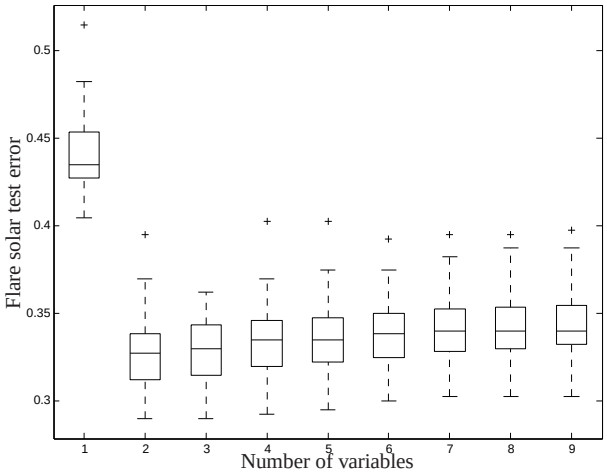

Fig. 5.

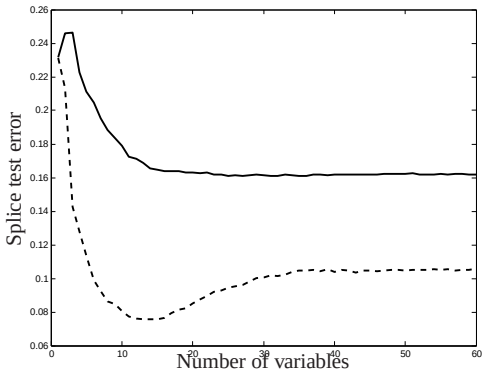

Fig. 6.

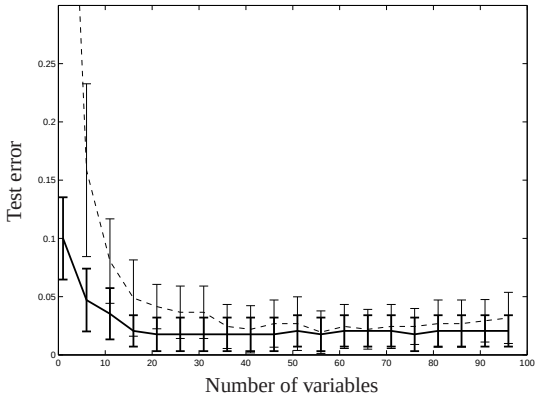

Fig. 7.

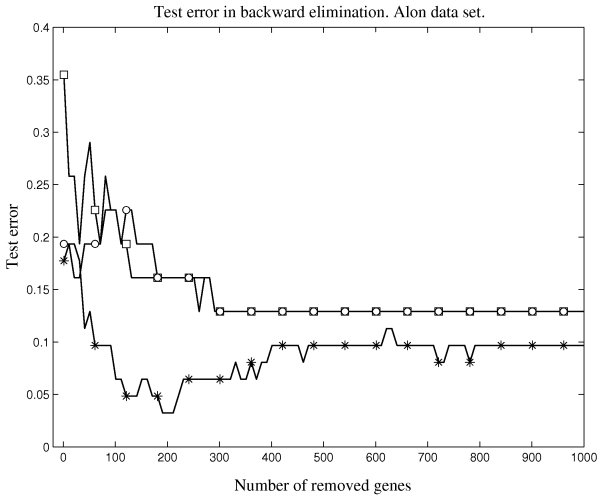

Fig. 8.

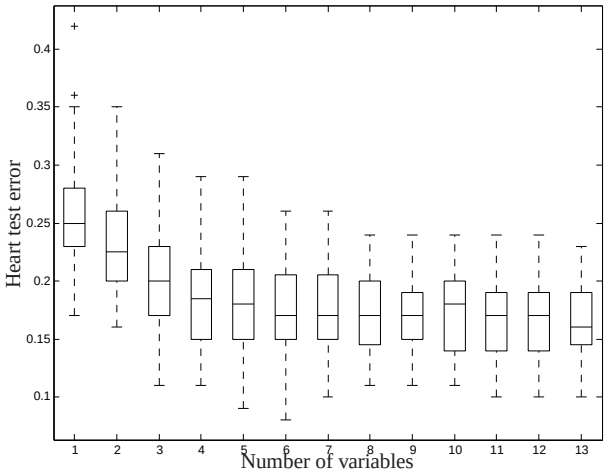

Fig. 9.

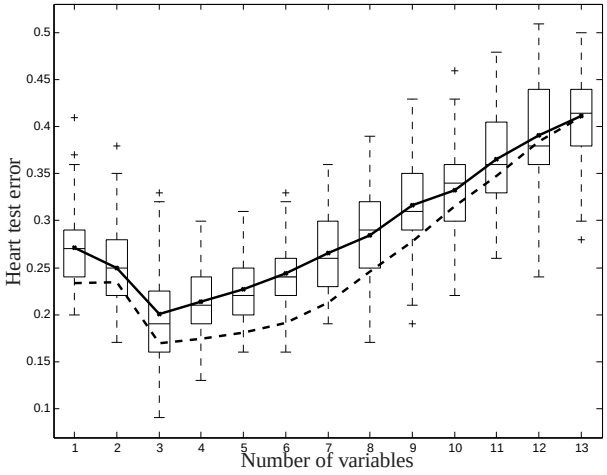
Heart test error
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
1
2
3
4
5
6
7
8
9
10
11
12
13
Number of variables

Fig. 10.