# SMO Algorithm for Least Squares SVM Formulations

*S.S. Keerthi* *and* *S.K. Shevade*

Dept. of Mechanical Engineering
National University of Singapore
Singapore 117576

Genome Institute of Singapore
National University of Singapore
Singapore 117528

June 11, 2002

**Abstract**

*This paper extends the well-known SMO algorithm of Support Vector Machines (SVMs) to Least Squares SVM formulations which include LS-SVM classification, Kernel Ridge Regression and a particular form of regularized Kernel Fisher Discriminant. The algorithm is shown to be asymptotically convergent. It is also extremely easy to implement. Computational experiments show that the algorithm is fast and scales efficiently (quadratically) as a function of the number of examples.*

## 1 Introduction

This paper concerns least squares formulations of SVMs. These formulations, which provide interesting and useful alternatives to SVMs, have been proposed both for regression as well as classification. Saunders et al. (1998) proposed the kernel version of ridge regression and empirically demonstrated its usefulness. Suykens and Vandewalle (1999) gave an interesting and useful extension of the idea to classification and called their method as Least Squares SVM (LS-SVM). A careful empirical study by Van Gestel et al. (2000) has shown that LS-SVM is comparable to SVM in terms of generalization performance. Recently Van Gestel et al. (2002) have also established the equivalence of LS-SVM with a particular form of regularized Kernel Fisher Discriminant (KFD) method (Mika et al. 2001).

While SVMs are characterized by convex quadratic programming problems involving inequality constraints, kernel ridge regression and LS-SVMs on the other hand, are formulated using equality constraints only. Hence, their solution is obtained by solving a system of linear equations (of the order of the number of training examples) involving the dense kernel matrix. To tackle large scale problems, Suykens et al (1999) and Van Gestel et al. (2000) effectively employed the conjugate-gradient method. In this paper we propose a new iterative algorithm for these problems. It is based on the solution of the Wolfe dual problem using ideas similar to those of the SMO (Sequential Minimal Optimization) algorithm for SVMs (Platt, 1999, Keerthi et al., 2001). The algorithm is extremely easy to implement and scales efficiently (quadratically) as a function of the number of examples. The scaling is much better than that of the CG algorithm.

The paper is organized as follows. In section 2 we formulate the problem and develop the Wolfe dual. Optimality conditions for the dual are derived in section 3. A useful condition for terminating numerical algorithms solving the least squares formulations is also given. These conditions form the basis for the SMO algorithm developed in section 4. In this section we also prove that the algorithm is asymptotically convergent. In section 5 we discuss the conjugate gradient method of Suykens et al (1999) and show how the termination criterion derived in section 3 can be easily applied to this method. Computational experiments describing the effectiveness of the algorithm are given in section 6. Section 7 briefly discusses the solution of KFD. Section 8 extends the SMO algorithm to least square formulations without the threshold parameter, followed by some concluding remarks in section 9. The appendix gives a pseudocode for the SMO algorithm.

## 2 Problem Formulation

Throughout we will use $x$ to denote the input vector of the classification/regression problem and $z$ to denote the feature space vector which is related to $x$ by the transformation, $z = \varphi(x)$. As in all kernel designs, we do not assume $\varphi$ to be known; all computations will be done using only the kernel function,

$$k(x, \hat{x}) = \varphi(x) \cdot \varphi(\hat{x}) \tag{2.1}$$

where "$\cdot$" denotes inner product in the $z$ space. Let $\{(x_i, y_i)\}_{i=1}^m$ denote the training set, where $x_i$ is the $i$-th input pattern and $y_i$ is the corresponding target value. For classification problems (two classes) $y_i$ takes only two possible values (say, 1 if $x_i$ is in class 1 and $-1$ if $x_i$ is in class 2) while, for regression $y_i$ takes any real value. Let $z_i = \varphi(x_i)$. In kernel ridge regression[1] and

[1]Saunders et al. (1998) leave out $b$ from their formulation. This actually leads to a simpler problem. We will discuss its solution in section 8.

classification by LS-SVMs the following optimization problem is solved:

$$\begin{array}{rcl} \min E & = & \frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_i \xi_i^2 \\ \text{s.t.} & & y_i - (w \cdot z_i - b) = \xi_i \;\; \forall i \end{array} \tag{2.2}$$

Define:

$$\tilde{w} = \left( \begin{array}{c} w \\ \sqrt{C}\xi \end{array} \right); \quad \tilde{z}_i = \left( \begin{array}{c} z_i \\ \frac{1}{\sqrt{C}} e_i \end{array} \right) \;\; \forall i \tag{2.3}$$

where $\xi$ is a $m$ dimensional vector containing the $\xi_i$'s and, for every $i$, $e_i$ is the $m$ dimensional vector in which the $i$-th component is 1 and all other components are 0. Define $\tilde{k}(x_i, x_j) = \tilde{z}_i \cdot \tilde{z}_j$. By (2.1) and (2.3),

$$\tilde{k}(x_i, x_j) = k(x_i, x_j) + \frac{1}{C}\delta_{ij} \tag{2.4}$$

where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. It is easy to see that (2.2) transforms to

$$\min E = \frac{1}{2}\|\tilde{w}\|^2 \;\; \text{s.t. } y_i - (\tilde{w} \cdot \tilde{z}_i - b) = 0 \; \forall i \tag{2.5}$$

The Lagrangian for this problem is:

$$L = \frac{1}{2}\|\tilde{w}\|^2 + \sum_i \alpha_i [y_i - (\tilde{w} \cdot \tilde{z}_i - b)] \tag{2.6}$$

The KKT optimality conditions are given by:

$$\nabla_{\tilde{w}} L = \tilde{w} - \sum_i \alpha_i \tilde{z}_i = 0 \tag{2.7}$$

$$\frac{\partial L}{\partial b} = \sum_i \alpha_i = 0 \tag{2.8}$$

Note that, because (2.5) has only equality constraints, the $\alpha_i$'s are not constrained to be non-negative. Using (2.7) $\tilde{w}$ can be expressed as a function of the $\alpha_i$'s:

$$\tilde{w}(\alpha) = \sum_i \alpha_i \tilde{z}_i \tag{2.9}$$

Let us now apply Wolfe duality theory to (2.5). The Wolfe dual corresponds to the maximization of $L$ subject to (2.7) and (2.8), with $\tilde{w}$, $b$ and $\alpha_i$'s as variables. Using (2.8) and

(2.9) we can simplify the Wolfe dual as

$$\max f(\alpha) = -\frac{1}{2}\|\tilde{w}(\alpha)\|^2 + \sum_i \alpha_i y_i \quad \text{s.t.} \quad \sum_i \alpha_i = 0 \tag{2.10}$$

Note that $\|\tilde{w}\|^2 = \sum_i \sum_j \alpha_i \alpha_j \tilde{k}(x_i, x_j)$. Hence (2.10) is a simple convex quadratic programming problem involving a single equality constraint and no nonnegativity constraints. Once the $\alpha_i$'s are obtained by solving (2.10), the primal variables, $w$ and $\xi_i$'s can be determined using (2.9) and (2.3). The determination of $b$ will be addressed in the next section.

# 3 Optimality Conditions for Dual

To derive proper stopping conditions for algorithms which solve the dual and also determine the threshold parameter $b$, it is important to write down the optimality conditions for the dual. The Lagrangian for (2.10) is:

$$\bar{L} = -\frac{1}{2}\|\tilde{w}(\alpha)\|^2 + \sum_i \alpha_i y_i + \beta \sum_i \alpha_i \tag{3.1}$$

Define

$$F_i = -\frac{\partial f}{\partial \alpha_i} = \tilde{w}(\alpha) \cdot \tilde{z}_i - y_i = \sum_j \alpha_j \tilde{k}(x_i, x_j) - y_i \tag{3.2}$$

where $f$ is defined in (2.10). The KKT conditions for the dual problem are:

$$\frac{\partial \bar{L}}{\partial \alpha_i} = (\beta - F_i) = 0 \ \ \forall\, i \tag{3.3}$$

Define:

$$b_{\text{up}} = \max_i F_i \qquad i\_up = \arg\max_i F_i \quad \text{and} \quad b_{\text{low}} = \min_i F_i \qquad i\_low = \arg\min_i F_i \tag{3.4}$$

Then optimality conditions will hold at a given $\alpha$ iff

$$b_{\text{low}} = b_{\text{up}} \tag{3.5}$$

In the above, note that $F_i$, $b_{\text{up}}$, $i\_up$, $b_{\text{low}}$ and $i\_low$ are all functions of $\alpha$. The functional dependancies have not been put down to avoid notational clutter.

Using (3.3), (3.2) and (2.5), it is easy to see the close relationship between the threshold parameter $b$ in the primal problem and the multiplier, $\beta$. *In particular, at optimality, $\beta$ and $b$ are identical.* Therefore, in the rest of the paper $\beta$ and $b$ will denote one and the same quantity.

We will say that an index pair $(i, j)$ defines a *violation* at $\alpha$ if

$$F_i \neq F_j \tag{3.6}$$

*Thus, optimality conditions will hold at $\alpha$ iff there does not exist any index pair $(i, j)$ that defines a violation.*

Suppose $(i, j)$ satisfies (3.6) at some $\alpha$. Then it is possible to achieve an increase in $f$ (while maintaining the equality constraint, $\sum \alpha_k = 0$) by adjusting $\alpha_i$ and $\alpha_j$ only. To see this, let us define the following:

$$\tilde{\alpha}_i(t) = \alpha_i - t\ ; \quad \tilde{\alpha}_j(t) = \alpha_j + t\ ; \quad \text{and} \quad \tilde{\alpha}_k(t) = \alpha_k \ \forall k \neq i, j, \tag{3.7}$$

and

$$\phi(t) = f(\tilde{\alpha}(t)) \tag{3.8}$$

Then it is easy to verify that

$$\phi'(t) = F_i - F_j \tag{3.9}$$

where $F_i$ and $F_j$ are evaluated at $\tilde{\alpha}(t)$. Since, by (3.6), $F_i - F_j \neq 0$ at $t = 0$, an increase in $\phi$ is possible by choosing $t$ suitably away from 0.

Since, in numerical solution, it takes too much computing time to compute the exact optimal solution, there is a need to define approximate stopping conditions that guarantee a specified closeness to the solution of the primal problem, (2.2). We consider a stopping condition based on duality gaps. This has been used effectively by Smola and Schölkopf (1998) in SVM regression.

Suppose $\alpha$ is any vector that satisfies $\sum_i \alpha_i = 0$, $b$ is any real number, $w = \sum_i \alpha_i z_i$ and

$$\xi_i = y_i - (w \cdot z_i - b) = b + \frac{\alpha_i}{C} - F_i \tag{3.10}$$

where $F_i$ is as defined by (3.2). Note that $(w, b, \xi)$ satisfies the constraints of the primal, (2.2) and $\alpha$ satisfies the constraints of the dual, (2.10). Let: $E$ denote the primal cost corresponding to $(w, b, \xi)$; $E^\star$ be the optimal primal cost of (2.2); $f$ denote the dual objective function value given by $\alpha$; and $f^\star$ be the optimal dual objective function value. By Wolfe duality (Fletcher, 1984) we have:

$$E \geq E^\star = f^\star \geq f \tag{3.11}$$

Eventhough $E^\star (= f^\star)$ is unknown, (3.11) allows us to obtain guaranteed bounds using duality gap (a computable quantity) defined by

$$Dgap = E - f = \sum_i [\alpha_i (F_i - \frac{\alpha_i}{2C}) + \frac{C}{2}\xi_i^2] \tag{3.12}$$

$Dgap$ is a nonnegative quantity that becomes zero iff optimality is reached. Thus, if $\{\alpha^r\}$ is a sequence that converges to $\alpha^\star$, the solution of (2.10), then the continuity of the $Dgap$ function implies that $\{Dgap^r\}$ goes to zero, where $Dgap^r$ is the value of $Dgap$ for $\alpha^r$. Also, for any non-trivial problem, $E^\star = f^\star > 0$. Thus, it is appropriate to choose $\epsilon > 0$ and use the stopping criterion,

$$Dgap \leq \epsilon f \tag{3.13}$$

More importantly, if (3.13) is satisfied, then we can obtain a guaranteed bound for the primal problem using (3.11):

$$E - E^\star \leq Dgap \leq \epsilon f \leq \epsilon E \tag{3.14}$$

Thus, if $\epsilon = 10^{-p}$ and (3.13) is satisfied, then the closeness of $E$ to $E^\star$ has more than $p$ decimal digits of accuracy. These ideas are used in interior point methods of linear programming (Vanderbei, 1994) and have also been employed in SVMs (Smola and Schölkopf, 1998).

In our implementation of the SMO algorithm derived in the next section, we employ (3.13) as the stopping criterion. In section 5 we explain how to use (3.13) to efficiently stop the conjugate gradient algorithm for LS-SVM proposed by Suykens et al. (1999).

At a given $\alpha$, we need a method for determining a good value of $b$ for use in the above calculations as well as for use in the classifier function defined by

$$o(x) = w \cdot \varphi(x) - b = \sum_i \alpha_i k(x, x_i) - b \tag{3.15}$$

One possibility is to take

$$b = \frac{b_{\text{low}} + b_{\text{up}}}{2} \tag{3.16}$$

where $b_{\text{up}}$ and $b_{\text{low}}$ are as defined in (3.4). An alternative method is to choose $b$ to minimize $Dgap$ after fixing $w$ at $w(\alpha)$. This is equivalent to minimizing $\sum_i \xi_i^2$, which can be done in closed form to get

$$b = \frac{1}{m}\sum_i (F_i - \frac{\alpha_i}{C}) \tag{3.17}$$

# 4 SMO Algorithm

In this section we give the SMO algorithm for solving (2.10). A basic step consists of starting with a point $\alpha$ and optimizing only two variables $\alpha_i$ and $\alpha_j$ to form the new point $\alpha_{\text{new}}$. Consider (3.7) and (3.8). Given (3.9), the natural choice is $i = i\_up$ and $j = i\_low$ so as to make $|\phi'(0)|$ as large as possible. Using the notations of section 3, we can write the optimization problem and the resulting solution as

$$t^\star = \arg\max_t \phi(t) \quad \text{and} \quad \alpha_{\text{new}} = \tilde{\alpha}(t^\star) \tag{4.1}$$

The function $\phi(t)$ is a simple quadratic function of $t$:

$$\phi(t) = \phi(0) + t\phi'(0) + \frac{t^2}{2}\phi''(0) \tag{4.2}$$

where $\phi'$ is given by (3.9) and

$$\phi''(0) = \eta = 2\tilde{k}(x_i, x_j) - \tilde{k}(x_i, x_i) - \tilde{k}(x_j, x_j) \tag{4.3}$$

Thus (4.1) can be easily solved in closed form:

$$t^\star = -\frac{\phi'(0)}{\phi''(0)} = -\frac{F_i - F_j}{\eta} \tag{4.4}$$

The SMO algorithm can now be described. It is very much along the lines of the simplified version of the SMO algorithm for SVMs given by Chang and Lin (2001).

**SMO Algorithm for (2.10).**

1. Choose any $\alpha^0$ and set $r = 0$.

2. If $\alpha^r$ satisfies (3.5), stop. If not, set $\alpha = \alpha^r$, choose $i = i\_up$, $j = i\_low$ and solve (4.1).

3. Let $\alpha^{r+1} = \alpha_{\text{new}}$, $r := r + 1$ and go back to step 2.

We now establish asymptotic convergence of the above algorithm. The absence of 'hard boundaries' in the optimization makes the proof of convergence much simpler than corresponding proofs for SMO algorithm for SVMs. Putting (4.4) in (4.2) and simplifying, we get

$$f(\alpha^{r+1}) - f(\alpha^r) = \phi(t^\star) - \phi(0) = -\frac{(t^\star)^2}{2}\eta \tag{4.5}$$

Using (2.4) we see that $\eta < -2/C$. Also, $|t^\star| = \|\alpha^{r+1} - \alpha^r\|/\sqrt{2}$. Thus (4.5) implies a *sufficient increase in* $f$ *in each basic step of the SMO algorithm:*

$$f(\alpha^{r+1}) - f(\alpha^r) \geq \frac{1}{2C}\|\alpha^{r+1} - \alpha^r\|^2 \tag{4.6}$$

Because of (2.4), $-f$ is a positive definite quadratic form. Hence the solution of (2.10) is unique. Call this point as $\alpha^\star$. We show the following result.

**Lemma 1.** *$\{\alpha^r\}$ converges to $\alpha^\star$.*

**Proof.** Since the algorithm increases $f$ at each step and $f$ is bounded above, $\{f(\alpha^r)\}$ is a convergent sequence. By (4.6) we get that $\{\alpha^{r+1} - \alpha^r\}$ converges to 0.

Because $-f$ is a positive definite quadratic form, the set $\{\alpha | f(\alpha) \geq f(\alpha^0\}$ is a compact set. Since $\{\alpha^r\}$ lies in this set, it is a bounded sequence and therefore it has at least one limit point. Let $\{\alpha^{r(s)}\}_{s\geq 0}$ denote a convergent subsequence and $\bar{\alpha}$ denote the limit point to which it converges. For any $r \geq 0$, let $i(r) = i\_up(\alpha^r)$ and $j(r) = i\_low(\alpha^r)$, the two indices chosen for optimization at the $r$-th step. Since $\phi'(t^\star) = 0$ for $t^\star$ given by (4.1), we get from (3.9) that

$$F_{i(r)}(\alpha^{r+1}) - F_{j(r)}(\alpha^{r+1}) = 0 \tag{4.7}$$

Since there are only a finite number of indices, there exists at least one pair $(i_1, j_1)$ such that $i_1 = i(r(s))$ and $j_1 = j(r(s))$ for infinitely many $s$. Let us restrict ourselves to only such a subsequence. To keep notations simple, let us rename the subsequence and take that

$$i_1 = i(r(s)) = i\_up(\alpha^{r(s)}) \qquad and \qquad j_1 = j(r(s)) = i\_low(\alpha^{r(s)}) \qquad \forall\, s \geq 0$$

Since $b_{\text{up}}$ and $b_{\text{low}}$ are continuous functions of $\alpha$, we also get

$$\begin{aligned} b_{\text{up}}(\bar{\alpha}) - b_{\text{low}}(\bar{\alpha}) &= \lim_{s\to\infty}[b_{\text{up}}(\alpha^{r(s)}) - b_{\text{low}}(\alpha^{r(s)})] \\ &= \lim_{s\to\infty}[F_{i_1}(\alpha^{r(s)}) - F_{j_1}(\alpha^{r(s)})] \\ &= \lim_{s\to\infty}[P(s) + Q(s) + R(s)] \end{aligned} \tag{4.8}$$

where

$$\begin{aligned} P(s) &= [F_{i_1}(\alpha^{r(s)}) - F_{i_1}(\alpha^{r(s)+1})] \\ Q(s) &= [F_{i_1}(\alpha^{r(s)+1}) - F_{j_1}(\alpha^{r(s)+1})] \\ R(s) &= [F_{j_1}(\alpha^{r(s)+1}) - F_{j_1}(\alpha^{r(s)})] \end{aligned} \tag{4.9}$$

Since $\{\alpha^{r(s)+1} - \alpha^{r(s)}\}$ converges to 0, $\lim_{s\to\infty} P(s) = 0$ and $\lim_{s\to\infty} R(s) = 0$. By (4.7), $Q(s) = 0 \;\; \forall s$. Thus (4.8) yields $b_{\text{up}}(\bar{\alpha}) - b_{\text{low}}(\bar{\alpha}) = 0$. By (3.5), $\bar{\alpha}$ is a solution of (2.10), i.e., $\bar{\alpha} = \alpha^\star$. Strict convexity of $-f$ implies that the sequence $\{\alpha^r\}$ itself converges to $\alpha^\star$. This completes the proof. ■

The implementation of the SMO algorithm is extremely simple. A pseudocode is given in the appendix. For efficiency, a cache is maintained for all $F_i$'s. At each basic step, when only two indices, say $i1$ and $i2$ change their values from $\alpha_{i1}^{old}$ and $\alpha_{i2}^{old}$ to $\alpha_{i1}^{new}$ and $\alpha_{i2}^{new}$, the $F_i$'s can be updated as

$$F_i^{new} = F_i^{old} + (\alpha_{i1}^{new} - \alpha_{i1}^{old})\tilde{k}(x_{i1}, x_i) + (\alpha_{i2}^{new} - \alpha_{i2}^{old})\tilde{k}(x_{i2}, x_i) \;\; \forall i \tag{4.10}$$

Also, the updating of the dual objective function can be written, using (4.4) and (4.5) as

$$f^{new} = f^{old} - \frac{\eta}{2}(t^\star)^2 \tag{4.11}$$

where $t^\star = (\alpha_{i2}^{new} - \alpha_{i2}^{old}) = (\alpha_{i1}^{old} - \alpha_{i1}^{new})$.

The algorithm can be initialized with $\alpha = 0$. For this choice, $F_i = -y_i$ for all $i$ and $f(\alpha) = 0$. When the regularization parameter $C$ is tuned (say, using $k$-fold cross validation) we usually come across a situation in which the solution of (2.10) is available for some $C$ and (2.10) needs to be solved again for $\delta C$ where $\delta > 0$. In such a situation we can do *alpha seeding* (DeCoste and Wagstaff, 2000), i.e., utilize the solution at $C$ to initialize the variables for solving the problem at $\delta C$. We give below the details.

Suppose $\alpha$ is the solution at $C$ and, $f$, $F_i \; \forall \, i$ are available. Then, to start the algorithm for $\delta C$, we can use the same $\alpha$. For this choice, the $F_i$ for $\delta C$ can be initialized using

$$F_i := F_i - \frac{(\delta - 1)\alpha_i}{\delta C} \;\; \forall i \tag{4.12}$$

After the $F_i$ are computed, $f$ can be obtained using

$$f = \frac{1}{2} \sum_i \alpha_i (y_i - F_i) \tag{4.13}$$

An alternative is to initialize with $\delta\alpha$ for solving the problem at $\delta C$. Update formulas similar to (4.12) and (4.13) can be derived for this case too. Both the above initializations work effectively and give similar performance.

# 5 Conjugate Gradient Algorithm

Suykens et al. (1999) devised an efficient method for obtaining the solution of LS-SVM using the conjugate gradient technique (Golub and Van Loan, 1996). This method is very suited for problems with a large number of examples. Let $\tilde{K}$ be the $m \times m$ matrix whose $(i, j)$th element is $\tilde{k}(x_i, x_j)$. The steps of their method are as follows.[2]

**CG Method**

1. Solve
$$\tilde{K}\eta = e \quad \text{and} \quad \tilde{K}\nu = y \tag{5.1}$$
for $\eta$ and $\nu$ using the conjugate gradient method (twice), where $e$ is a vector with all elements equal to 1 and $y$ is a vector whose $i$th component is $y_i$.

2. Compute $b = -\eta \cdot y / \eta \cdot e$ and set $\alpha = \nu + b\eta$.

Although the conjugate gradient algorithm solves a system of $m$ linear equations in $m$ steps, for efficiency purposes it is important to stop the algorithm much earlier than $m$ steps. Hamers et al (2000) suggest stopping the algorithm when one of three criteria are satisfied; for details, see their paper. These criteria are not directly related to the solution of (2.2) and hence they cannot be used efficiently to stop the algorithm with a prescribed guarantee on the primal solution, such as satisfying (3.14). Below we explain how $Dgap$ can be computed easily in the CG method described above so that (3.14) can be satisfied via (3.13).

[2] The notations and description of the method as given here are different from those given by Suykens et al. (2000).

Although the linear systems in (5.1) are decoupled, for efficient use of kernel computations, they are solved together. At any stage of application of the conjugate gradient method to solve them, $\eta$ and $\nu$ that approximately satisfy (5.1) are available. The residue vectors, $r_\eta = e - \tilde{K}\eta$ and $r_\nu = y - \tilde{K}\nu$ play a crucial role in the computations and are also available. Step 2 of the CG method can be used to construct approximate $b$ and $\alpha$. Using that $\alpha$, it is possible to compute $F$ (the vector containing the $F_i$'s) as

$$F = \tilde{K}\alpha - y = b(e - r_\eta) - r_\nu \tag{5.2}$$

Then $Dgap$ can be obtained using (3.10) and (3.12). However, there is a difficulty associated with the use of $Dgap$ computed in this way for ensuring (3.14). Recall from section 3 that, to derive (3.14), an important requirement is that the constraint $\sum_i \alpha_i = 0$ has to be satisfied. Since $\eta$ and $\nu$ are only approximate solutions of (5.1), this condition is usually not satisfied. Fortunately, this issue is easily taken care of if we replace step 2 of the CG method with this alternate implementation:

$$\text{Compute } \; b = -\nu \cdot e / \eta \cdot e \; \text{ and set } \; \alpha = \nu + b\eta \tag{5.3}$$

What we have done is replace $\eta \cdot y$ with $\nu \cdot e$ in the calculation of $b$. Note that, when $\eta$ and $\nu$ satisfy (5.1) we have $\eta \cdot y = \nu \cdot e$. Once step 2 is replaced by (5.3), application of (5.2), (3.10), (3.12) and (3.13) guarantees (3.14).

Like in the SMO algorithm, alpha seeding can be done for the CG method also. This has been pointed out by Hamers et al. (2001). Efficient update formulas for $r_\nu$, $r_\eta$ and the dual objective function can be written down similar to (4.12) and (4.13).

# 6 Numerical Experiments on Classification Problems

The SMO and CG methods were implemented in Fortran and executed on a Pentium IV 1.5 GHz machine running on Windows platform. The Gaussian kernel $K(x, \bar{x}) = \exp(-\frac{\|x-\bar{x}\|^2}{2\sigma^2})$ was used. For both methods we used the termination criterion in (3.13) with $\epsilon = 10^{-6}$.

We empirically compared the two methods in terms of computational cost. This cost is measured using the number of kernel evaluations since the bulk of the effort in both the algorithms resides in computing kernels. Four benchmark datasets were used for this purpose:

Banana, Image, Splice and Waveform. Detailed information about these datasets can be found in Rätsch (1999). For each dataset, we fixed $\sigma$ at a suitable value which gives good generalization performance and varied $C$ over a wide range.

In the first experiment we tried the following nine C values: $10^i$, $i = -4, -3, \ldots, 3, 4$. For each $C$, the methods were initialized with $\alpha = 0$. The computational costs associated with the four datasets are given in Table 1 as functions of $C$.

In the second experiment we tested the effect of alpha seeding on the computational cost. $C$ was sequentially varied from $2^{-10}$ till $2^{10}$ in multiples of 2. Thus there were twenty one $C$ values. For $C = 2^{-10}$ the methods were initialized with $\alpha = 0$. For all other $C$ values alpha seeding was used.[3] The computational costs associated with the four datasets are given in Table 2 as functions of $C$.

For the SMO algorithm the increase in cost at large $C$ values is sharp. See, for instance, the growth in cost of SMO after $C = 10^2$ for Banana and Image datasets. In the case of Image dataset, while SMO was doing quite better than CG at low and medium $C$ values, it became very expensive at large $C$ values. In most practical problems, the optimal value of $C$ where best generalization occurs (say, the value at which the $k$-fold cross validation error is smallest) is usually not high. For instance, in the case of Image dataset, 5-fold cross validation error was smallest around $C = 24.16$. Thus, when searching for the optimal $C$, if large $C$ values can be carefully avoided, then the SMO algorithm will work very efficiently. It is also worth noting that, for both SMO and CG methods, the effectiveness of the alpha seeding diminishes at large $C$ values. Since large $C$ values correspond to large costs for both methods, such values should be avoided (unless really needed) in the $C$ tuning procedure to maintain efficiency.

On the Image and Splice datasets SMO was quite faster than CG at the middle $C$ values. Since the 'optimal' $C$ usually occurs in this range, this is an advantage in favor of the SMO method. On the Banana and Waveform datasets the difference between the two methods was not much in this range of $C$ values. When $C$ is tuned using cross validation, we usually start with a low $C$ value, say $2^{-10}$, increase $C$ in steps (say in multiples of 2) and stop when the cross validation error clearly shows no further decrease. We compared the CG and the SMO methods with respect to the total cost associated with this process. To do this, we used the

[3]In section 4 we mentioned two alpha seedings: (a) set $\alpha$ to the optimal $\alpha$ of the previous $C$ value; and (b) set $\alpha$ to $\delta$ times the optimal $\alpha$ of the previous $C$ value. Our experiment showed that both initializations give nearly the same cost.

Table 1: Computational costs for SMO and CG algorithms ($\alpha = 0$ initialization). Each unit corresponds to $10^6$ kernel evaluations.

| | Banana $\sigma^2 = 1.8221$ | | Image $\sigma^2 = 2.7183$ | | Splice $\sigma^2 = 29.9641$ | | Waveform $\sigma^2 = 24.5325$ | |
|---|---|---|---|---|---|---|---|---|
| $\log_{10} C$ | SMO | CG | SMO | CG | SMO | CG | SMO | CG |
| -4 | 0.90 | 0.32 | 9.30 | 3.38 | 4.73 | 1.00 | 0.93 | 0.32 |
| -3 | 0.83 | 0.48 | 7.48 | 5.07 | 4.35 | 2.00 | 0.91 | 0.48 |
| -2 | 0.53 | 0.64 | 5.17 | 8.05 | 2.94 | 3.00 | 0.52 | 0.64 |
| -1 | 0.50 | 1.28 | 4.84 | 20.28 | 3.13 | 6.00 | 0.42 | 1.12 |
| 0 | 0.70 | 2.24 | 7.05 | 49.01 | 3.07 | 13.00 | 0.54 | 2.40 |
| 1 | 3.53 | 4.00 | 33.65 | 131.82 | 5.33 | 30.00 | 2.39 | 5.76 |
| 2 | 31.58 | 7.84 | 254.13 | 410.67 | 11.45 | 65.00 | 10.51 | 14.56 |
| 3 | 320.42 | 14.40 | 1908.62 | 1275.95 | 29.88 | 89.00 | 21.62 | 23.36 |
| 4 | 3317.90 | 32.32 | 11825.40 | 4096.56 | 99.90 | 109.00 | 28.33 | 27.36 |

same four datasets of Table 2, while also fixing $\sigma^2$ at the values given in that table. Five fold cross validation was employed. Table 3 gives the ratio of the coputational efforts associated with the two methods. The SMO method is clearly faster in all cases. So the SMO algorithm proposed in this paper can be viewed as an excellent alternative method for solving LS-SVMs.

To see how the SMO algorithm for LS-SVM scales to large size problems, we did a scaling experiment on the Adult and Web datasets by gradually increasing the training set size in eight steps and observing the training cost. These datasets are available from Platt's SMO page: *http://www.research.microsoft.com/˜jplatt/smo.html*. The Adult dataset was trained with $C = 1.0$ and $\sigma^2 = 10$ while the Web dataset was trained with $C = 5.0$ and $\sigma^2 = 10$. Figures 1 and 2 give plots of the computational costs of the two algorithms as functions of the problem size. The SMO algorithm for LS-SVM scales well on both the datasets, with scaling exponents of 2.017 and 2.006 respectively for the Adult and Web datasets. For the CG algorithm the corresponding scaling exponents were 2.385 and 2.496. This is due to the fact that the number of conjugate gradient iterations needed to achieve (3.13) slowly rises as the number of examples increases. The above results indicate that the SMO algorithm is better suited for large problems.

Next we briefly compared the generalization performance of SVM and LS-SVM. Five-fold cross validation was used to tune the hyperparameters involved in the problem formulations (that is, $C$ and $\sigma$) and the test set error was obtained using the optimal hyperparameter values for each of the formulations. The search for optimal hyperparameters was done on a $20 \times 20$

Table 2: Computational costs for SMO and CG algorithms with alpha seeding. Each unit corresponds to $10^6$ kernel evaluations.

| | Banana $\sigma^2 = 1.8221$ | | Image $\sigma^2 = 2.7183$ | | Splice $\sigma^2 = 29.9641$ | | Waveform $\sigma^2 = 24.5325$ | |
|---|---|---|---|---|---|---|---|---|
| $\log_2 C$ | SMO | CG | SMO | CG | SMO | CG | SMO | CG |
| -10 | 0.83 | 0.48 | 7.44 | 5.07 | 4.37 | 2.00 | 0.91 | 0.48 |
| -9 | 0.62 | 0.32 | 6.15 | 3.38 | 3.50 | 1.00 | 0.78 | 0.32 |
| -8 | 0.47 | 0.32 | 5.54 | 5.07 | 3.34 | 2.00 | 0.70 | 0.32 |
| -7 | 0.51 | 0.48 | 4.56 | 5.07 | 2.92 | 2.00 | 0.50 | 0.48 |
| -6 | 0.44 | 0.64 | 4.69 | 6.76 | 2.96 | 3.00 | 0.39 | 0.48 |
| -5 | 0.44 | 0.80 | 4.09 | 10.14 | 2.75 | 3.00 | 0.40 | 0.64 |
| -4 | 0.42 | 0.96 | 3.98 | 13.52 | 2.56 | 4.00 | 0.42 | 0.80 |
| -3 | 0.42 | 1.12 | 3.79 | 18.59 | 2.70 | 5.00 | 0.37 | 0.96 |
| -2 | 0.43 | 1.28 | 3.82 | 21.97 | 2.40 | 6.00 | 0.38 | 1.12 |
| -1 | 0.45 | 1.60 | 4.38 | 28.73 | 2.32 | 8.00 | 0.39 | 1.44 |
| 0 | 0.60 | 2.08 | 5.83 | 38.87 | 2.43 | 10.00 | 0.49 | 1.92 |
| 1 | 0.90 | 2.24 | 8.71 | 54.08 | 2.74 | 14.00 | 0.73 | 2.56 |
| 2 | 1.47 | 2.88 | 13.95 | 72.67 | 3.19 | 17.00 | 1.09 | 3.68 |
| 3 | 2.61 | 3.04 | 24.16 | 98.02 | 3.62 | 23.00 | 1.82 | 4.96 |
| 4 | 4.84 | 4.00 | 43.70 | 138.58 | 3.91 | 32.00 | 2.99 | 6.40 |
| 5 | 9.31 | 5.12 | 81.07 | 197.73 | 4.09 | 37.00 | 4.70 | 8.96 |
| 6 | 18.42 | 5.44 | 149.06 | 265.33 | 4.04 | 52.00 | 7.09 | 11.84 |
| 7 | 36.56 | 7.04 | 278.62 | 378.56 | 4.17 | 65.00 | 9.92 | 14.72 |
| 8 | 72.40 | 8.64 | 518.92 | 549.25 | 4.18 | 73.00 | 12.78 | 17.44 |
| 9 | 146.95 | 9.76 | 955.57 | 763.88 | 4.04 | 82.00 | 14.91 | 20.64 |
| 10 | 297.37 | 12.16 | 1717.04 | 1098.50 | 3.77 | 88.00 | 15.83 | 23.36 |

Table 3: Ratio of the cross validation cost of CG and the cross validation cost of SMO algorithm.

| Banana ($\sigma^2 = 1.8221$) | Image ($\sigma^2 = 2.7183$) | Splice ($\sigma^2 = 29.9641$) | Waveform ($\sigma^2 = 24.5325$) |
|---|---|---|---|
| 1.89 | 2.62 | 5.77 | 1.34 |

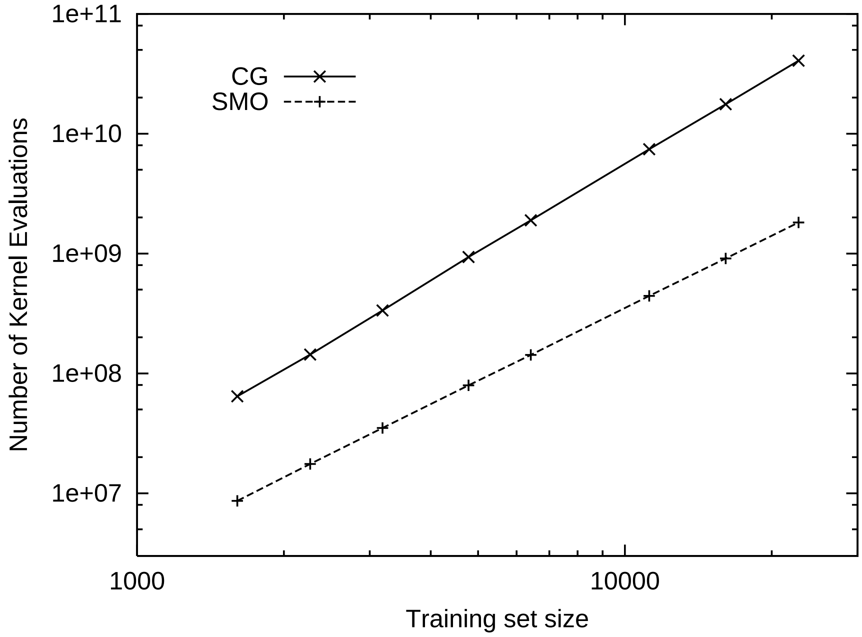


Figure 1: Variation of computational cost with training set size for Adult dataset.

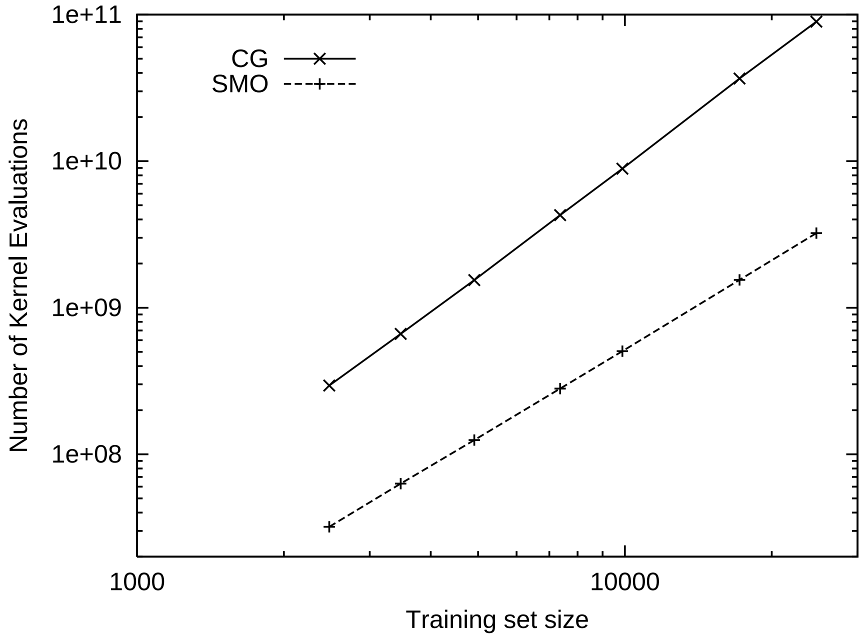


Figure 2: Variation of computational cost with training set size for Web dataset.

Table 4: Generalization performances of LS-SVM and SVM on four benchmark datasets.

| Dataset | LS-SVM | SVM |
|---|---|---|
| Banana | .1161 | .1180 |
| Image | .0208 | .0228 |
| Splice | .1012 | .0989 |
| Waveform | .1089 | .1076 |

uniform grid in the $(\log C, \log \sigma)$ space. The fraction of test set errors (TErr) are given in Table 4. It is clear that the generalization capabilities of both methods are comparable. This observation is consistent with that made by Van Gestel et al. (2000).

## 7 Solution of KFD

Mika et al. (1999) first proposed the idea of (regularized) KFD. Mika, Rätsch and Müller (2001) gave a quadratic programming formulation of KFD employing general regularizers. For the case of K-regularizer, i.e., $\|w\|^2$, Van Gestel et al. (2002) established the equivalence[4] between LS-SVM and KFD. In this section we give an alternative to Van Gestel et al's way of establishing the equivalence between the KFD and LS-SVM formulations. Our approach clearly brings out the relationship between the dual variables and the threshold parameter associated with the two formulations.

The quadratic programming formulation of Mika, Rätsch and Müller (2001) is:

$$\begin{array}{ll} \min & \frac{1}{2}\|\hat{w}\|^2 + \frac{C}{2}\sum_i \hat{\xi}_i^2 \\ \text{s.t.} & y_i - (\hat{w}\cdot z_i - \hat{b}) = \hat{\xi}_i \;\; \forall i \\ & \sum_{i\in I_+} \hat{\xi}_i = 0 \quad \text{and} \quad \sum_{i\in I_-} \hat{\xi}_i = 0 \end{array} \tag{7.1}$$

where $I_+$ and $I_-$ are the index sets corresponding to the two classes and, as usual, $y_i$ takes the value 1 if example $i$ is in class 1 and $-1$ otherwise.

The Lagrangian for this problem is:

$$\hat{L} = \frac{1}{2}\|\hat{w}\|^2 + \frac{C}{2}\sum_i \hat{\xi}_i^2 + \sum_i \hat{\alpha}_i[y_i - (\hat{w}\cdot z_i - \hat{b}) - \hat{\xi}_i] + \mu_+ \sum_{i\in I_+} \hat{\xi}_i + \mu_- \sum_{i\in I_-} \hat{\xi}_i$$

[4] By *equivalence* we mean that the normal directions of the optimal hyperplanes associated with the two formulations are the same.

where the $\hat{\alpha}_i$'s, $\mu_+$ and $\mu_-$ are Lagrange multipliers. Setting the gradients of $\hat{L}$ with respect to $\hat{w}$, $\hat{b}$ and $\hat{\xi}_i$'s to zero and using them together with the equalities in (7.1) yields the following square system of equations for solving the $\hat{\alpha}_i$'s, $\hat{b}$, $\mu_+$ and $\mu_-$:

$$\begin{array}{c} \sum_j K_{ij}\hat{\alpha}_j + \frac{1}{C}\hat{\alpha}_i = \hat{b} + \frac{\mu_i}{C} + y_i \ \ \forall\, i \\ \sum_i \hat{\alpha}_i = 0, \quad \sum_{i\in I_+}\hat{\alpha}_i = m_+\mu_+ \ \text{ and } \ \sum_{i\in I_-}\hat{\alpha}_i = m_-\mu_- \end{array} \tag{7.2}$$

where: $\mu_i$ is $\mu_+$ if $y_i = 1$ and $\mu_-$ if $y_i = -1$; $m_+$ is the number of examples in class 1 (positive class); $m_-$ is the number of examples in class 2 (negative class); and $m = m_+ + m_-$. These equations imply that $m_+\mu_+ + m_-\mu_- = 0$. Let us define $\mu$ using the equation

$$\mu = m_+\mu_+ = -m_-\mu_- \tag{7.3}$$

If we define

$$a = 1 + \frac{\mu}{2C}\left(\frac{1}{m_+} + \frac{1}{m_-}\right) \quad \text{and} \quad b = \frac{1}{a}\left[\hat{b} + \frac{\mu}{2C}\left(\frac{1}{m_+} - \frac{1}{m_-}\right)\right] \tag{7.4}$$

then the right hand side of each of the first level equations in (7.2), i.e., $\hat{b} + \frac{\mu_i}{C} + y_i$ can be rewritten as $a(b + y_i)$. If we also define:

$$\alpha_i = \frac{\hat{\alpha}_i}{a} \quad \forall i \tag{7.5}$$

it is easy to see that the $\alpha_i$ and $b$ are obtained by solving

$$\begin{array}{c} \sum_j K_{ij}\alpha_j + \frac{1}{C}\alpha_i - b = y_i \ \ \forall\, i \\ \sum_i \alpha_i = 0 \end{array} \tag{7.6}$$

Using the $\alpha_i$'s we can get $\mu$ by solving

$$\sum_{i\in I_+} \alpha_i = \mu/a = \mu/\left[1 + \frac{\mu}{2C}\left(\frac{1}{m_+} + \frac{1}{m_-}\right)\right] \tag{7.7}$$

Then (7.3), (7.4) and (7.5) yield $\mu_+$, $\mu_-$, $a$, $\hat{b}$ and the $\hat{\alpha}_i$'s.

Since (7.6) is nothing but the solution of the instance of (2.2) corresponding to LS-SVM, it follows that the solution of (7.1) can be obtained by solving LS-SVM; this can be numerically done using the SMO algorithm of section 4. The two equality constraints in the last row of (7.1) only contribute to a scaling of $w$ and a shift in $b$ given by the LS-SVM solution. Thus, if $b$ and

$\hat{b}$ associated with (2.2) and (7.1) are determined in a uniform way (say, using the class posterior probabilities obtained in a post-processing step (Platt, 1999)) then the classifiers obtained by using these two formulations are identical.

## 8 Primal problem without $b$

In kernel ridge regression (Saunders et al. 1998) and Bayesian designs (Van Gestel et al. 2002) it is usual to consider the primal problem (2.2) without the $b$ parameter.[5] When $b$ is absent, the equality constraint $\sum_i \alpha_i = 0$ disappears from the dual, (2.10) and the optimality conditions for the dual problem become

$$F_i = 0 \ \ \forall i \tag{8.1}$$

Because of the absence of the equality constraint, the SMO algorithm can simply modify one $\alpha_i$ at a time. If the indices are chosen sequentially for updating, then the SMO algorithm is precisely the SOR algorithm (Bertsekas, 1999; Hamers et al. 2000). Efficiency of this SOR algorithm can be significantly improved if we modify it so that, at any given instance, the index that gives the best increase in $f$ value is chosen. This can be neatly done by maintaining a cache for the $F_i$'s. Like the sequential SOR algorithm the modified algorithm also requires only $O(m)$ computations and kernel evaluations in each step. However, the modified algorithm requires much fewer steps to achieve the termination criterion. Let us now work out the details of the modified algorithm.

Suppose at a given $\alpha$, an index $i$ is considered for updating. Define:

$$\tilde{\alpha}_i(t) = \alpha_i + t \ ; \quad \tilde{\alpha}_j(t) = \alpha_j \ \forall j \neq i \ ; \quad \text{and} \quad \phi(t) = f(\tilde{\alpha}(t)) \tag{8.2}$$

With $t^\star$ and $\alpha_{\text{new}}$ as defined in (4.1), it is easy to check that

$$t^\star = \frac{F_i}{\tilde{k}(x_i, x_i)} \quad \text{and} \quad f(\alpha_{\text{new}}) - f(\alpha) = \frac{F_i^2}{2\tilde{k}(x_i, x_i)} \tag{8.3}$$

Thus, at each step of the SMO algorithm we can choose $i$ that has the largest value of $\frac{F_i^2}{2\tilde{k}(x_i,x_i)}$. Since a cache is maintained for the $F_i$'s and $\tilde{k}(x_i, x_i)$, $i = 1, \ldots, m$ can be precomputed and

[5] If $b$ is present and the term $\gamma b^2/2$ is added to the cost function in (2.2), that is equivalent to a primal problem without $b$ if the constant $1/\gamma$ is added to the kernel function $k$.

stored, the above step can be efficiently done. Convergence of the resulting SMO algorithm can be established along the lines given in section 4. The proofs actually become much simpler because of the simpler problem and algorithm associated with the absence of $b$.

The solution of (2.10) without the equality constraint corresponds to the solution of $\tilde{K}\alpha = y$ and so the solution can also be obtained directly using the conjugate gradient method. Duality gap ideas discussed in section 3 can be easily adapted (simply omit $b$ in all calculations) to give efficient stopping criteria for both SMO as well as CG methods.

# 9 Conclusion

In this paper we have given a new algorithm for least squares formulations of SVMs (LS-SVMs and kernel ridge regression) proved its convergence and discussed implementation aspects. Our algorithm also applies to a particular form of regularized Kernel Fisher Discriminant method. The algorithm is robust in the sense that on the many complex datasets we have tried there was not even a single case of failure. It is also fast and scales well to large size problems. It provides a better alternative to the conjugate gradient algorithm developed by Hamers et al. (2001), especially for large problems.

# References


Bertsekas, D.P., and Tsitsiklis, J.N. (1997). *Parallel and Distributed Computation: Numerical Methods.* Athena Scientific.

Bertsekas, D.P. (1999). *Nonlinear Programming.* Athena Scientific, 2nd Edition.

Chang, C.C., and Lin, C.J. (2001). *LIBSVM: A library for support vector machines,* Dept. of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

DeCoste, D., and Wagstaff, K. (2000). *Alpha seeding for support vector machines,* International Conference on Knowledge Discovery & Data Mining (KDD-2000).

Fletcher, R. (1987). *Practical Methods of Optimizations.* John Wiley and Sons, 2nd Edition.

Golub, G.H., and Van Loan, C.F. (1996). *Matrix Computations,* third edition, The Johns Hopkins University Press Ltd., London.

Hamers, B., Suykens, J., and De Moor, B. (2001). *A comparison of iterative methods for least squares support vector machine classifiers,* Internal Report 01-110, ESAT-SISTA, K.U.Leuven (Leuven, Belgium).

Keerthi, S.S., Shevade, S.K., Bhattacharyya, C., and Murthy, K.R.K. (2001). *Improvements to Platt's SMO algorithm for SVM design,* Neural Computation, vol. 13, pp.637-649.

Mika, S., Rätsch, G., Weston, J., Schölkopf, B., and Müller, K.R. (1999). *Fisher discriminant analysis with kernels,* In Y.H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX,* pp.41-48, IEEE.

Mika, S., Rätsch, G., and Müller, K.R. (2001). *A mathematical programming approach to the kernel Fisher algorithm,* In T.K. Leen, T.G. Dietterich, and V. Tresp, editors, Advances in Neural Information Processing Systems 13, pp.591-597, MIT Press.

Platt, J. (1998). *Sequential minimal optimization: A fast algorithm for training support vector machines,* Technical Report MSR-TR-98-14, Microsoft Research, Redmond.

Platt, J. (1999). *Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods,* In A. Smola, B. Schölkopf and D. Schuurmans (eds.), *Advances in Large Margin Classifiers*, MIT Press, Cambridge, MA.

Rätsch, G. (1999). *http://ida.first.gmd.de/˜raetsch/data/benchmarks.htm*

Saunders, C., Gammerman, A., and Vovk, V. (1998). *Ridge regression learning algorithm in dual variables,* Proceedings of the $15^{th}$ International Conference on Machine Learning ICML-98, Madison, WI, USA.

Smola, A., and Schölkopf, B. (1998) *A tutorial on support vector regression,* NeuroColt2 Technical Report NC2-TR-1998-030, ESPRIT Working Group in Neural and Computational Learning II.

Suykens, J., and Vandewalle, J. (1999). *Least squares support vector machine classifiers,* Neural Processing Letters, vol. 9, no. 3, June 1999, pp. 293-300.

Suykens, J., Lukas, L., Van Dooren, P., De Moor, B., and Vandewalle, J., (1999). *Least squares support vector machine classifiers: a large scale algorithm,* in Proc. of the European Conference on Circuit Theory and Design (ECCTD'99), Stresa, Italy, Sep. 1999, pp. 839-842.

Vanderbei, R.J. (1994). *LOQO: An Interior Point Code for Quadratic Programming,* Technical Report SOR-94-15, Statistics and Operations Research, Princeton University, NJ, USA.

Van Gestel, T., Suykens, J., Baesens, B., Viaene, S., Vanthienen, J., Dedene, G., De Moor, B., and Vandewalle, J. (2000). *Benchmarking least squares support vector machine classifiers,* Internal Report 00-37, ESAT-SISTA, K.U.Leuven (Leuven, Belgium), 2000.

Van Gestel, T., Suykens, J., Lanckriet, G., Lambrechts, A., De Moor, B., and Vandewalle, J. (2002). *A Bayesian framework for least squares support vector machine classifiers,* Neural Computation.

## Appendix. Pseudocode for the SMO Algorithm.

Notes for the pseudocode given below:

1. `kernel(i,j)` refers to $\tilde{k}(x_i, x_j)$ given by (2.4).
2. The $\alpha_i$ are stored in `alpha` array while the $F_i$ are stored in `Fcache` array.
3. $\epsilon$ in (3.13) is referred to as `epsilon`; $10^{-6}$ is a suitable value for it.
4. `eps` is used to check if sufficient change has occurred in each step. It can be set to $10^{-12}$.
5. The function `fabs` returns the absolute value of a real number.
6. Any line in the pseudocode that begins with the character "%" is treated as a comment.

```
procedure takeStep(i1, i2)
    alph1 = alpha[i1]; alph2 = alpha[i2]; y1 = y[i1]; y2 = y[i2];
    F1 = Fcache[i1]; F2 = Fcache[i2]; gamma = alph1 + alph2;
    k11 = kernel(i1,i1); k22 = kernel(i2,i2); k12 = kernel(i1,i2);
    eta = 2*k12-k11-k22; a2  = alph2 - (F1-F2)/eta;
    if (fabs(a2-alph2) < eps*(a2+alpha2+eps)) return 0;
    a1 = gamma-a2; alpha[i1] = a1; alpha[i2] = a2;
    Update the DualObjectiveFunction using (4.11).
    Update Fcache[i] for all i in I using (4.10).
```

```
     Compute (i_low, b_low) and (i_up, b_up) using (3.4).
     return 1;
endprocedure

procedure ComputeDualityGap()
     DualityGap = 0; set b using (3.16) or (3.17).
     loop over all i
         xi = b + (alpha[i]/C) - Fcache[i];
         DualityGap += alpha[i](Fcache[i]-(0.5*alpha[i]/C)) + C*xi*xi/2;
     return DualityGap;
endprocedure

procedure InitializeVariables(delta)
     if delta is non-zero
         % delta denotes the factor by which C is changed.
         for every i, reset Fcache[i] using (4.12).
         compute DualObjectiveFunction using (4.13).
     else
         initialize alpha[i]=0 for all i.
         set Fcache[i]=-y[i] for all i  and DualObjectiveFunction to 0.
     Compute (i_low, b_low) and (i_up, b_up) using (3.4).
endprocedure

procedure SMO()
     % If solution is available at some value of C=C1 and solution is to be
     % found at C=C2 then set delta := C2/C1. Otherwise, set delta := 0.
     InitializeVariables(delta);
     numChanged = 1; DualityGap = ComputeDualityGap();
     while (DualityGap > epsilon*DualObjectiveFunction && numChanged != 0)
         numChanged = takeStep(i_up, i_low); DualityGap = ComputeDualityGap();
% If the procedure stops with numChanged=0, it means that epsilon is too small.
endprocedure
```