Volume 28, N. 3, pp. 327–364, 2009
Copyright © 2009 SBMAC
ISSN 0101-8205
www.scielo.br/cam

On the use of the Spectral Projected Gradient method for Support Vector Machines*

DEBORA CORES†, RENÉ ESCALANTE,
MARÍA GONZÁLEZ-LIMA and OSWALDO JIMÉNEZ
Universidad Simón Bolívar
Departamento de Cómputo Científico y Estadística
Apdo. 89000, Caracas 1080-A, Venezuela
E-mails: cores@cesma.usb.ve / rene@cesma.usb.ve /
mgl@cesma.usb.ve / oswjimenez@usb.ve

Abstract. In this work we study how to solve the SVM optimization problem by using the Spectral Projected Gradient (SPG) method with three different strategies for computing the projection onto the constrained set. One of the strategies is based on Dykstra's alternating projection algorithm since there is not a mathematical equation for the projection onto the whole constrained set but the projection on each restriction is easy to compute with exact formulations. We present another strategy based on the Karush-Kunh-Tucker optimality conditions, we call it the Projected-KKT algorithm. We compare these strategies with a third one proposed by Dai and Fletcher. The three schemes are low computational cost and their use within the SPG algorithm leads to a solution of the SVM problem. We study the computational performance of the three strategies when solving randomly generated as well as real life SVM problems. The numerical results show that Projected-KKT is competitive in general with the Dai and Fletcher algorithm, and it is more efficient for some specific problems. They both outperform Dykstra's algorithm in all the tests.

Mathematical subject classification: Primary: 46N10; Secondary: 47N40.

Key words: support vector machines, quadratic programming, Spectral Projected Gradient method.

#CAM-102/08. Received: 02/XII/08. Accepted: 10/V/09.

*This work was partially supported by the Centro de Estadística y Software Matemático (CESMa) and the Decanato de Investigación y Desarrollo (DID) at USB.

†Corresponding author.

1 Introduction

Support Vector Machines (SVM) [7, 12, 38] has been used in the past years for solving many real life applications. Some of them are pattern recognition and classification problems as for example isolated handwritten digit recognition [9, 10, 12, 33, 34] object recognition [6], speaker identification [32], face detection images [27, 28], text categorization [24] and some nonlinear least squares problems as the inverse density estimation problem [40]. The SVM technique is based on finding the minimal of a single linear constrained quadratic problem subject to lower and upper bounds. Because of the nature of the applications, the quadratic problem to solve is large with dense (and mostly bad conditioned) Hessian. Therefore, efficient low cost optimization algorithms are required.

Different techniques and methods have been used (e.g. [14, 19, 27, 35]), some of them are projected gradient types. However, dealing with dense large scale problems is not easy. Besides, the difficulty when using projected gradient type methods is that the exact projection over the whole constrained set is not explicitly available.

In this paper we consider the Spectral Projected Gradient (SPG) method [5] to solve the SVM optimization problem. The SPG method is a low computational cost and storage since only requires first order information, few floating point operations per iteration, and few function evaluations since it does not required an exhaustive line search strategy as other projected type methods. It utilizes a nonmonotone line search globalization scheme that guarantees convergence without requiring the objective function to decrease at every iteration. These properties of the SPG method make it very attractive for solving large scale problems as it is shown in [5].

In order to solve the projection problem arising at each iteration of the SPG method, we present a new algorithm based on the solution of the Karush-Kuhn-Tucker (KKT) conditions, the so-called Projected-KKT algorithm. The idea is to solve a reduction of the linear system associated to the KKT conditions by using a partition into three sets of indices. If the partition is optimal, the solution of the projection problem is found. When the partition is not optimal, the sets are updated trying to satisfy the strict complementarity conditions and a new linear system is solved. Exact formulas can be derived for the solution of the linear system so each inner iteration is very cheap. The algorithm iterates until a solution is found. This idea is of the same spirit as that of the algorithm presented by Júdice and Pires in [25] but the Projected-KKT algorithm allows to handle lower and upper bounds in contrast with the Júdice and Pires scheme. Besides, comparing both algorithms in the case of two index sets we observe that they follow completely different strategies for moving the indices between sets. Therefore, both algorithms generate different sequence of iterates.

Because there is not a mathematical equation for the projection over the whole constrained set but the projection over each restriction is easy to compute with exact formulations we also consider, following Birgin et al. [3], a version of the Dykstra's alternating method for solving the projection problem.

Dai and Fletcher in [14] propose an algorithm based on secant approximation to solve quadratic programming problems with single linear constraints and bounds in the case that the Hessian matrix of the quadratic function is diagonal and positive definite. Therefore, this algorithm can be also used to solve the projection problem as it is illustrated in [14].

In this work we propose to solve the SVM problem with the Spectral Projected Gradient (SPG) method but computing the projection over the whole constrained set by three different projection techniques: the proposed Projected-KKT algorithm, a version of the Dykstra's algorithm, and the recently developed algorithm based on secant approximations due to Dai and Fletcher. The nature of these methods is different but we consider that their comparison is fair since they are all viable, and potentially low cost, approaches for solving the quadratic problem.

The SPG method assures convergence to a stationary point of the constrained quadratic problem from any initial guess (see [3, 5]). Also, the SPG algorithm utilizes a nonmonotone line search strategy that does not force the objective function to decrease at each iteration which reduces the number of function evaluations when it is compared with a monotone line search.

The SPG method has also been used in the SVM context in [35, 41]. In [35] the inner quadratic subproblems (arising when using GVP or SPG methods) are solved via the bisection-like algorithm proposed by Pardalos and Kovoor [29]. In [41] the emphasis relies on the decomposition techniques and the development of parallel software. There, the secant-based algorithm by Dai and Fletcher is used for solving the large size projection subproblems. Our work studies different alternatives for solving the projection subproblem and the impact on the SPG method by performing extensive numerical experiments with SVM-type problems. We show that the Projected-KKT algorithm we propose is competitive with the secant-based algorithm introduced in [14].

The outline of the paper is the following. In the next two sections we give a brief introduction to SVM, we describe the optimization problem to be solved and we introduce the SPG method when applied to the SVM optimization problem. The following section is devoted to present the projection strategies. The last section presents numerical results with the three strategies for randomly generated problems and for several interesting real life problems as detecting face images, object recognition, text categorization problems, and isolated handwritten digit recognition among others.

2 The Support Vector Machine Problem

The Support Vector Machines (SVM) is an interesting classification technique developed by Vapnik and his group at AT&T Bell laboratories [12, 38]. In this section, we give a brief introduction to this technique and describe the quadratic optimization problem to be solved when using SVM. The notation adopted in this section is, with slight modifications, that of [27].

Let us consider the given data set $\{(x_i, y_i), i = 1, \dots, m$ with $x_i \in \mathbb{R}^n$ and $y_i \in Z\}$. In the SVM context, the optimization problem corresponding to the case in which the given data is linearly separable is the following

$$\begin{array}{ll} \min_{(w,b)} & \frac{1}{2}\|w\|^2 \\ \text{subject to} & y_i(w^T x_i + b) \geq 1 \ \ \forall\, i = 1, \dots, m, \end{array} \tag{1}$$

where $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$.

In order to motivate this optimization problem let us consider, for simplicity, only the cases where $y_i = 1$ or $y_i = -1$, for all i. In fact, we will consider only these cases in the whole paper. Therefore, we have two index sets, these are: $I_1 = \{i : y_i = 1\}$ and $I_{-1} = \{i : y_i = -1\}$ corresponding to two classes, say

Class 1 and Class 2, respectively, according to which class each x_i belongs to. Notice that $I_1 \bigcup I_{-1} = \{1, \dots, m\}$.

To solve problem (1) means to find the hyperplane with normal vector w that maximizes the separation margin between Class 1 and Class 2. Once a solution (w^*, b^*) of (1) is found, given any point $x \in \mathbb{R}^n$, the value of the function $f(x) = \operatorname{sign}({w^*}^T x + b^*)$ indicates to which class this point belongs. Observe that the number of constraints of problem (1) is equal to the number of data samples which, for real life problems, is large.

The support vectors are called the vectors from the given data set where at least one of the constraints in the feasible set of (1) is satisfied as equality. It is easy to see that these support vectors necessarily exist. Indeed, if

$$
{w^*}^T x_i + b^* > 1 \ \forall\, i \in I_1 \quad \text{or} \quad {w^*}^T x_i + b^* < -1 \ \forall\, i \in I_{-1},
$$

then there exists a pair $(\tilde{w}, \tilde{b})$ with $\|\tilde{w}\| = \|w^*\|$ such that, for any $i \in I_1$, $\tilde{w}^T x_i + \tilde{b} = 1$ and, for any $i \in I_{-1}$, $\tilde{w}^T x_i + \tilde{b} = -1$.

The dual problem corresponding to problem (1) is a quadratic program of the form

$$
\begin{array}{ll}
\max_\lambda & e^T\lambda - \frac{1}{2}\lambda^T D\lambda \\
\text{subject to} & y^T\lambda = 0 \\
& \lambda \geq 0,
\end{array}
\tag{2}
$$

where $\lambda = (\lambda_1, \dots, \lambda_m)^T$ are the Lagrange multipliers, $y = (y_1, \dots, y_m)^T$, e is the m-vector of all ones, and $D \in \mathbb{R}^{m\times m}$ is the symmetric positive semi-definite matrix defined as $D_{ij} = (y_i x_i)^T (y_j x_j)$. Let us call λ^* a solution of (2). Then we can recover the solution (w^*, b^*) of (1) by using duality theory. That is,

$$
w^* = \sum_{i=1}^{m} \lambda_i^* y_i \hat{x}_i \tag{3}
$$

where the support vectors $\hat{x}_i$ are the ones corresponding to $\lambda_i^* > 0$, so b^* solves the equation

$$
{w^*}^T \hat{x}_i + b^* = y_i \tag{4}
$$

for any support vector.

Then, the decision function, that is the function that says if a given point x is in Class 1 or 2, can be written as

$$f(x) = \operatorname{sign}\left(\sum_{i=1}^{m} \lambda_i^* y_i \hat{x}_i^T x + b^*\right),$$

with λ^* the optimal multiplier vector. This is saying that for any $x \in \mathbb{R}^n$ if $f(x) > 0$, then x is on Class 1, and if $f(x) < 0$, then x is on Class 2.

Hence, in order to solve (1) we can just solve the dual problem (2). One advantage of doing this is that constraints in (2) are simpler than in the original problem. Another important advantage is that working with the dual problem the dimension of the feature space for the classification can be increased without increasing the dimension of the optimization problem to solve. This last comment will become clearer in the following.

If the data set is linearly nonseparable ([13], this is, that there does not exist a solution of problem (1)) then two variants are considered. On one hand, (1) is changed to

$$\begin{array}{ll} \min_{(w,b,\xi)} & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^m \xi_i^h \\ \text{subject to} & y_i(w^T x_i + b) \geq 1 - \xi_i \;\; i = 1, \ldots, m, \\ & \xi_i \geq 0, \;\; i = 1, \ldots, m, \end{array} \tag{5}$$

where h and C are positive constants to be chosen and $\xi = (\xi_1, \ldots, \xi_m)$ is a vector with variables that measure the amount of violation of the constraints.

Additionally, in the nonseparable case, the nonlinear decision surface is computed by mapping the input variable x into a higher dimensional "feature space" and by working with linear classification in that space. In other words, we define an application for $x \in \mathbb{R}^n$ into a "feature vector":

$$x \longrightarrow \Phi(x) = \big(a_1\phi_1(x), a_2\phi_2(x), \ldots, a_n\phi_n(x), \ldots\big)^T,$$

where $a_i \in \mathbb{R}$ and $\{\phi_n\}_{n=1}^{\infty}$ are some real functions. So, in the nonseparable case, the primal problem is obtained by substituting the variable x with the new "feature vector" $\Phi(x)$ in (5).

The dual problem corresponding to (5) (for $h = 1$) is given by

$$\begin{array}{ll} \max_{\lambda} & e^T\lambda - \frac{1}{2}\lambda^T D\lambda \\ \text{subject to} & y^T\lambda = 0 \\ & 0 \leq \lambda_i \leq C \quad \text{for } i = 1, \dots, m \end{array} \tag{6}$$

where $D \in \mathbb{R}^{m\times m}$ is a symmetric positive semi-definite matrix with positive diagonal, defined as $D_{ij} = y_i y_j \Phi(x_i)^T\Phi(x_j)$.

Observe that the dimension of problem (6) is the same of problem (2) and therefore, the computational effort for solving it is the same.

In practice, an explicit description of Φ is not needed. This is avoided by using the concept of kernel function K. Let K be defined as $K(x, y) = \Phi(x)^T\Phi(y)$. Therefore, in problem (6) the matrix D is given by $D_{ij} = y_i y_j K(x_i, x_j)$. Using the kernel definition, the decision function can be re-written as

$$f(x) = \text{sign}\left(\sum_{i=1}^{m} y_i\lambda_i^* K(x, x_i) + b^*\right),$$

which is now a nonlinear function given by linear superposition of kernel functions (one for each support vector). Therefore, in theory, what is really needed is to find a function that preserves, in higher dimensional spaces, the properties of the inner product, these are the kernel functions. We refer to [9] for the interested reader. In practice, different kernel functions are known and they are used for SVM problems. Some work has even been performed with non positive semi-definite Kernels as the sigmoid (see [23]). Experimental knowledge and data distribution may suggest the kernel function to use. But to find the right kernel for each application is a difficult task that goes beyond our work. In our numerical experimentation we use the suggested kernels in the literature.

In this work we are interested in solving the convex quadratic Problem (6). Observe that the Hessian of the objective function has dimension $m \times m$ being m the number of data samples, therefore it is usually large for real applications. Besides, the matrix is dense. Therefore, low cost and storage methods are preferable for solving it. We propose to solve it with the SPG method which is described in the next section.

Finally, observe that if the data is linearly separable, $\text{rank}(D) = n$ if $n < m$ (which is usually the case in practice). Therefore, in these instances the matrix is singular. In the case of using the kernel, the rank of the matrix D depends on the dimension of the feature space given by the kernel. Larger this dimension is, larger is the rank of D. We will show that our proposal Projected-KKT is very competitive when used with SPG in the case the matrix D is (or nearly) singular.

3 Application of the Spectral Projected Gradient Method to the SVM Problem

Let Ω be a closed and convex set. The Spectral Projected Gradient (SPG) method [5] applied to any constrained optimization problem of the form

$$\begin{array}{ll} \min & f(\lambda) \\ \text{subject to} & \lambda \in \Omega \end{array} \tag{7}$$

are recently developed iterative techniques, based on the Spectral Gradient (SG) method, that allow to minimize a nonlinear function over a convex set. The SG method solves unconstrained optimization problems by using as a search direction the negative gradient with a particular choice of the steplength named spectral length (for more details see [30]). This direction is called the spectral gradient step. The SG method has shown to be an efficient technique for solving large scale problems as is shown in [30] where the authors compare the SG method with the popular extensions of the conjugate gradient methods PR$^+$ and CONMIN developed by Gilbert and Nocedal [20] and Shanno and Phua [36] respectively. They conclude that the SG method outperforms CONMIN and PR$^+$ in number of gradient evaluations and CPU time for most of the problems tested.

The SPG method finds a solution of problem (7) by using the projection of the spectral gradient step over the convex set Ω, as any projection gradient method. That is:

$$\lambda^{k+1} = \lambda^k + \rho^k d^k \tag{8}$$

where the superscript k corresponds to the iteration, and

$$d^k = P_\Omega\big(\lambda^k - \alpha^k \nabla f(\lambda^k)\big) - \lambda^k,$$

with P_Ω denoting the projection operator onto the convex set Ω. The parameter α^k is the spectral steplength at iteration k, and it is given by

$$\alpha^k = \frac{(s^k)^T s^k}{(s^k)^T z^k}, \tag{9}$$

where

$$s^k = \lambda^k - \lambda^{k-1} \quad \text{and} \quad z^k = \nabla f(\lambda^k) - \nabla f(\lambda^{k-1}).$$

The steplength ρ^k is chosen at each iteration such that convergence is guaranteed from any initial guess by satisfying, for some $M > 0$ the following nonmonotone condition

$$f(\lambda^{k+1}) \leq \max_{0 \leq j \leq \min\{k, M-1\}} f(\lambda^{k-j}) + \gamma \rho^k \nabla f(\lambda^k)^T d^k, \tag{10}$$

with $\gamma \in (0, 1)$ a chosen parameter.

The SPG method does not guarantee descent of the objective function at every iteration in contrast with the classic Projected Gradient method (see [26]), because of the spectral steplength (9) and the nonmonotone condition (10). In [5], the authors show that for many examples or applications, this nonmonotone behavior of the objective function makes the method very competitive and sometimes preferable than other optimization techniques. Also, in the SPG method, the spectral choice of the steplength is the essential feature that adds efficiency in the projected gradient methodology. The SPG method is a low cost and storage technique since few floating point operations and only first order information are required. Moreover, the nonmonotone condition (10) allows the objective function to increase at some iterations, which implies less function evaluations than the Armijo type condition (see [15]), used frequently in the classical projected gradient method or gradient type methods.

For solving the SVM problem (6), we are interested in projecting onto the set of restrictions $\Omega = \mathcal{B} \cap \mathcal{H}$, where

$$\begin{aligned} \mathcal{B} &= \left\{ \lambda \in \mathbb{R}^m : 0 \leq \lambda_i \leq C \quad \text{for} \quad i = 1, \dots, m \right\} \quad \text{and} \\ \mathcal{H} &= \left\{ \lambda \in \mathbb{R}^m : y^T \lambda = 0 \right\}, \end{aligned} \tag{11}$$

where $y \in \mathbb{R}^m$ satisfies $|y_i| = 1$, for all i, and it corresponds to the problem (6). It is clear that Ω is a convex set.

The projection of any $u \in \mathbb{R}^n$ onto the set $\mathcal{B}$ (denoted by $P_{\mathcal{B}}$) does not require additional floating point operations, and it is given, for $i = 1, \dots, m$, by

$$(P_{\mathcal{B}}(u))_i = \begin{cases} 0 & \text{if} \quad u_i < 0 \\ C & \text{if} \quad u_i > C \\ u_i & \text{if} \quad 0 \leq u_i \leq C. \end{cases} \tag{12}$$

and the projection onto the set $\mathcal{H}$ (denoted by $P_{\mathcal{H}}$), only requires 2 inner products and it is given by:

$$P_{\mathcal{H}}(u) = u + \frac{y^T u}{y^T y} y. \tag{13}$$

It is important to stress that there is not an explicit mathematical equation for the projection of any vector over the whole set Ω. In this work, we propose to use three different strategies for projection over the set Ω. One strategy uses the Dykstra's algorithm given in [8] and [16], the second one is based on the Karush-Kuhn-Tucker optimality conditions together with a combinatorial approach, in the same spirit of the Júdice and Pires algorithm [25], and the third one uses the Dai and Fletcher secant scheme [14]. We call the second approach the Projected-KKT algorithm, and it is the one developed in this work.

In the next section, we describe the Dykstra's and Projected-KKT methods, and comment on the Dai and Fletcher's strategy.

4 Projection Strategies

Given $u \in \mathbb{R}^m$, we are interested in solving the projection problem

$$\begin{array}{ll} \min_{\lambda} & \frac{1}{2}\|\lambda - u\|_2^2 \\ \text{subject to} & \lambda \in \Omega = \mathcal{B} \cap \mathcal{H}. \end{array} \tag{14}$$

Problem (14) is a particular case of a quadratic programming problem, where the Hessian of the objective function is the identity matrix.

In the following we describe the three methods used for solving this problem.

4.1 Dykstra's Algorithm

Problem (14) is the minimization of a strictly convex quadratic function over the intersection of a finite collection of closed and convex sets in the vector space

$\mathbb{R}^m$. From this point of view, problem (14) can be solved by means of the alternating projection method, first proposed by Dykstra [16] for closed and convex cones, and later extended by Boyle and Dykstra [8] for closed and convex sets. Here we consider a particular case, due to Escalante and Raydan ([17, 18]), of a more general application of Dykstra's algorithm for solving the least-squares matrix problem.

The alternating projection method dates back to von Neumann [39] who treated the problem of finding the projection of a given point in a Hilbert space onto the intersection of two closed subspaces. Later, Cheney and Goldstein [11] extended the analysis of von Neumann's alternating projection scheme to the case of two closed and convex sets. In particular, they established convergence under mild assumptions. However, the limit point is not necessarily the closest one in the intersection set. Therefore, the alternating projection method, proposed by von Neumann, is not useful for problem (14).

Dykstra [16] found a clever modification of von Neumann's scheme. It guarantees convergence to the closest point in the intersection of closed and convex sets that are not necessarily closed subspaces. Dykstra's algorithm can also be obtained via duality (see [22]).

Next we present a version of Dykstra's algorithm ([17] and [18]) applied to problem (14).

Algorithm 1 (Dykstra's algorithm):
Given $u \in \mathbb{R}^m$, set $\lambda^0 = u$, and $I_{\mathcal{B}}^0 = 0 \in \mathbb{R}^m$.
For $k = 0, 1, 2, \ldots$,

$$\begin{aligned} \lambda^k &= P_{\mathcal{H}}(\lambda^k) - I_{\mathcal{B}}^k \\ I_{\mathcal{B}}^{k+1} &= P_{\mathcal{B}}(\lambda^k) - \lambda^k \\ \lambda^{k+1} &= P_{\mathcal{B}}(\lambda^k) \end{aligned} \tag{15}$$

Here $I_{\mathcal{B}}^k$ plays the role of the increments introduced by Dykstra [16]. There the vector of increments $I_{\mathcal{H}}^k = 0$ for all k, since in this case $\mathcal{H}$ is a hyperplane (i.e. a translated subspace).

It is important to note that the numerical performance of Algorithm 1 may be influenced by the angle between the sets $\mathcal{H}$ and $\mathcal{B}$. Next proposition addresses this topic.

Proposition 4.1. *Let us denote θ the angle formed by the hyperplane defined by the equation $y^T\lambda = 0$ and the box constraints. Then, $|\cos\theta| = \frac{1}{\sqrt{m}}$.*

Proof. The normal vectors to the box constraints are the coordinate vectors e^j with $j = 1, \ldots, m$. In the SVM application, $|y_j| = 1$ for all j. Therefore, $1 = |(e^j)^T y| = \|e^j\| \|y\| |\cos\theta| = \sqrt{m}|\cos\theta|$ for all j. □

This result shows that for large values of m, the angle between the hyperplane and the box constraints is close to $\frac{\pi}{2}$, which may imply that Algorithm 1 requires many iterations for convergence. However, this angle is not the only element that influence on the behavior of Algorithm 1. The choice of stopping criteria in Algorithm 1 is also very important. Birgin and Raydan in [4] present a simple example, where many of the stopping criteria used for alternating projection methods fail to converge and they propose a robust one. The example presented in [4] consists of two convex sets: one hyperplane and a set containing lower and upper bounds of the variables as problem (14). Therefore, projecting over simple convex and closed sets as the optimization problem (14) could take many iterations to converge.

4.2 Projected-KKT Algorithm

The Lagrangian function associated to the problem (14) is given by

$$\mathcal{L}(\lambda, \mu, w, v) = \frac{1}{2}\sum_{i=1}^{m}(u_i - \lambda_i)^2 + \mu\sum_{i=1}^{m} y_i\lambda_i - \sum_{i=1}^{m} w_i\lambda_i - \sum_{i=1}^{m} v_i(C - \lambda_i).$$

where μ, w and v are the Lagrange multiplier vectors associated with the equality constraints, the lower bound constraints and the upper bound constraints respectively. Therefore, the Karush-Kuhn-Tucker (KKT) conditions of problem (14) are

$$-(u_i - \lambda_i) + \mu y_i - w_i + v_i = 0 \tag{16}$$

$$\sum_{i=1}^{m} y_i\lambda_i = 0 \tag{17}$$

$$w_i\lambda_i = 0 \tag{18}$$

$$v_i(C - \lambda_i) = 0 \tag{19}$$

$$w_i \geq 0 \tag{20}$$

$$v_i \geq 0 \tag{21}$$

$$C - \lambda_i \geq 0 \tag{22}$$

$$\lambda_i \geq 0 \tag{23}$$

for all $i = 1, \ldots, m$. Given $\lambda \in \mathcal{B}$ let (L, U, I) be a partition of the index set $N = \{1, \ldots, m\}$ such that

$$\begin{aligned} L &= \big\{ i \in N : \ \lambda_i = 0 \big\}, \\ U &= \big\{ i \in N : \ \lambda_i = C \big\}, \ \text{ and} \\ I &= \big\{ i \in N : \ 0 < \lambda_i < C \big\}. \end{aligned}$$

Solving (14) means to find an optimal partition L, U, I such that equations (16) to (21) are satisfied.

Let us denote the dimensions of the sets L and U as l and s respectively, so the dimension of the set I is $m - l - s$.

From now on we use z_B to denote the vector which components are the z_i entries with $i \in B$. So, from the definitions of the sets L,U and I, and the equations (18) and (19), we have that

$$\lambda_L = v_L = 0 \in \mathbb{R}^l, \quad w_U = 0 \in \mathbb{R}^s, \quad w_I = v_I = 0 \in \mathbb{R}^{m-l-s}$$
$$\text{and} \quad \lambda_U = C_U = (C, C, \ldots, C)^T \in \mathbb{R}^s.$$

Using this notation, we can reduce the KKT system to a system of equations of order $(m + 1) \times (m + 1)$ instead of order $(3m + 1) \times (3m + 1)$ and the unknowns of the reduced KKT system are μ, w_L, v_U and λ_I. The reduced KKT equations can be written as

$$-u_L + \mu y_L - w_L = 0 \tag{24}$$

$$-u_U + C_U + \mu y_U + v_U = 0 \tag{25}$$

$$-u_I + \lambda_I + \mu y_I = 0 \tag{26}$$

$$y_U^T C_U + y_I^T \lambda_I = 0. \tag{27}$$

The Projected-KKT method is an iterative method that tries to find an optimal partition by starting with $\lambda^0 \in \mathcal{B}$ and solving the equations (24) to (27) by computing

$$\mu = \frac{{y_U}^T C_U + {y_I}^T u_I}{\|y_I\|_2^2} \tag{28}$$

$$w_L = -u_L + \mu y_L \tag{29}$$

$$v_U = -C_U + u_U - \mu y_U \tag{30}$$

$$\lambda_I = u_I - \mu y_I \tag{31}$$

The partition (L, U, I) is optimal if the following conditions are satisfied

$$0 \le \lambda_i = u_i - \mu y_i, \quad \text{for all } \; i \in I \tag{32}$$

$$C \ge \lambda_i = u_i - \mu y_i, \quad \text{for all } \; i \in I \tag{33}$$

$$0 \le v_i = -C + u_i - \mu y_i, \quad \text{for all } \; i \in U \tag{34}$$

$$0 \le w_i = -u_i + \mu y_i, \quad \text{for all } \; i \in L \tag{35}$$

Thus, the idea behind the proposed Projected-KKT algorithm is to rearrange the indices in the sets L, U and I that do not satisfy equations (32), (33), (34) and (35). The way that these indices are eliminated from one set and added to another one is very simple, and it is explained in the following algorithm, where the expression λ_i^k denotes the ith component of the vector λ at iteration k.

Algorithm 2 (Projected-KKT algorithm):

- **Step 0:** Given $u \in \mathbb{R}^m$, let $N = \{1, 2, \ldots, m\}$ and $k = 0$
- **Step 1:** $\lambda^k = P_{\mathcal{H}}(u)$,
- **Step 2:** If $0 \le \lambda_i^k \le C$, for all $i \in N$, stop λ^k is the solution of problem (14).
- **Step 3**: $\lambda^k = P_{\mathcal{B}}(\lambda^k)$
- **Step 4**: Let $L^k = \{i \in N : \; \lambda_i^k = 0\}$, $U^k = \{i \in N : \; \lambda_i^k = C\}$ and $I^k = N - (L^k \bigcup U^k)$,

- **Step 5:** If $I^k = \emptyset$ then
  - **Step 5.1:** If $y^T\lambda^k = 0$, stop λ^k is a vertex solution.
  - **Step 5.2:** If $y^T\lambda^k \neq 0$, do:
    - **Step 5.2.1:** $\lambda^{k+1} = P_{\mathcal{H}}(\lambda^k)$,
    - **Step 5.2.2:** $k = k + 1$ and repeat Step 3 and Step 4.
- **Step 6:** Compute the solution of the KKT system μ^k, w_{L^k}, v_{U^k} and λ_{I^k}, using equations (28), (29), (30) and (31)
- **Step 7:** If conditions (32), (33), (34) and (35) are satisfied, then stop: λ^k is the solution of problem (14).
- **Step 8:** Let

$$\begin{aligned} V_1 &= \{i \in I^k : \text{ (32) is not satisfied}\} \\ V_2 &= \{i \in I^k : \text{ (33) is not satisfied}\} \\ V_3 &= \{i \in U^k : \text{ (34) is not satisfied}\} \\ V_4 &= \{i \in L^k : \text{ (35) is not satisfied}\} \end{aligned}$$

- **Step 9:** $L^{k+1} = (L^k - V_4)\bigcup V_1$, $U^{k+1} = (U^k - V_3)\bigcup V_2$, $I^{k+1} = N - (L^{k+1}\bigcup U^{k+1})$

$$\lambda_i^{k+1} = \begin{cases} 0 & \text{if } i \in L^{k+1} \\ C & \text{if } i \in U^{k+1} \\ P_{\mathcal{B}}(\lambda_I^k) & \text{if } i \in I^{k+1} \end{cases} \tag{36}$$

- **Step 10:** If $I^{k+1} = \emptyset$ then
  - **Step 10.1:** If $y^T\lambda^{k+1} = 0$, then stop: λ^{k+1} is the solution of problem (14).
  - **Step 10.2:** If $y^T\lambda^{k+1} \neq 0$, eliminate an index from L^{k+1} or U^{k+1} and add it to the set I^{k+1}.
- **Step 11:** $k = k + 1$ and go to Step 6.

As we mentioned in the introduction the proposed Projected-KKT algorithm can be viewed as a finite termination algorithm in the sense that three finite index sets are defined at each iteration. The indices move from one set to another depending on the solution of the KKT first order optimality conditions. This idea is of the same spirit as that of the active set method or the algorithm presented by Júdice and Pires in [25]. The Projected-KKT algorithm allows to handle lower and upper bounds in contrast with the Júdice an Pires scheme. Comparing both algorithms in the case of two index sets we observe that they follow completely different strategies for moving the indices between sets. Therefore, both algorithms generate different sequence of iterates. Also, the initial iterate in the Projected-KKT algorithm is found by making use of the type of problem being solved. This choice, which is relevant to the performance of the algorithm, is different from the choice of the initial index set in [25]. Moreover, the Projected-KKT algorithm exploits the fact that the Hessian of the objective function is the identity.

The algorithm is well defined since every iteration has a solution and because there is a finite choice for the index sets the algorithm terminates in a finite number of iterations unless cycling occurs. In practice for diagonal positive definite Hessians, as in the projection problem, cycling never occurs. In fact, we have performed extensive experimentation using the algorithm for solving quadratic problems with any positive semi-definitive Hessian and we have observed that cycling is associated with the deficient rank of this matrix.

4.3 Dai and Fletcher method

Dai and Fletcher in [14] presented an optimization method for solving linearly constrained quadratic programming problems, where the Hessian matrix is diagonal and positive definite as the projected problem (14). This optimization strategy is based on secant approximations. The authors show numerical results that illustrate the performance of a modification of the SPG method and their projection strategy for some randomly generated problems and real life SVM problems. Details on the Dai and Fletcher algorithm can be found in [14]. The Dai and Fletcher scheme will be used in this work as another projection strategy for solving problem (14) together with the SPG method presented by Raydan in [5].

5 Numerical Experimentation

In this section we present the numerical results obtained by applying the SPG method for solving the SVM quadratic programming problem (6). For this quadratic problem the convex constrained set is denoted by $\Omega = \mathcal{B} \cap \mathcal{H}$, where the sets $\mathcal{B}$ and $\mathcal{H}$ are given in equation (11). The projection over the convex set Ω is computed by three different methods: The Dykstra's algorithm presented in Section 4.1, the Projected-KKT algorithm presented in Section 4.2, and the recently developed method by Dai and Fletcher [14]. Our aim in this section is to compare the performance of the three different projection schemes when using SPG method for solving random and real life SVM problems.

5.1 Test Problems

We consider two different kind of test problems: the first class consists of synthetic data randomly generated and the second class is taken from real applications databases. The kernels considered in this work are polynomials of degree d given by $K(x, y) = (1 + x^T y)^d$ and a Gaussian function with parameter σ given by

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{\sigma}\right).$$

Randomly generated test problems: the first four databases of this group consist of 250 nonlinearly separable samples of dimension two, where each component of the sample was randomly generated in the interval $[-0.5, 0.5]$ and has an uniform distribution. The classification of each sample x_i is given by

$$y_i = \begin{cases} -1 & \text{if} \quad f_j(x_i) \leq 0, \quad i \leq 245 \\ -1 & \text{if} \quad f_j(x_i) > 0, \quad i \geq 245 \\ 1 & \text{if} \quad f_j(x_i) > 0, \quad i < 245 \\ 1 & \text{if} \quad f_j(x_i) \leq 0, \quad i > 245, \end{cases} \tag{37}$$

where $j = 1, 2, 3, 4$. For these randomly generated problems, the mathematical expressions of the real separators and their level curves are known, so they can be drawn in $\mathbb{R}^3$ or in $\mathbb{R}^2$. From now on $x_i(k)$ denotes the k-th component of the vector x_i. Each value of j generates a database as follows:

lineal_wk: corresponds to $j = 1$ and $f_1(x_i) = x_i(1) + x_i(2)$.
No kernel is considered for this database.

sinoid_kp5: corresponds to $j = 2$ and $f_2(x_i) = x_i(2) - 0.25 \sin(8x_i(1))$.
A polynomial of degree five is considered as the kernel for this database.

polin3_kp3: corresponds to $j = 3$ and $f_3(x_i) = x_i(2) - 10(x_i(1))^3$.
A polynomial of degree three is used as the kernel for this database.

norm2_kg: corresponds to $j = 4$ and $f_4(x_i) = \|x_i\|_2^2 - (0.35)^2$.
A Gaussian kernel with $\sigma = 0.3$ is considered for this database.

The remaining databases of this group are:

Riply database: this database is taken from the Statistical Pattern Recognition Toolbox for Matlab and was proposed by Riply in [31]. It consists of 250 samples non linearly separable of dimension two, with 125 samples in class 1 and 125 samples in class 2. We denote by **riply_wk** this database without any kernel, and by **riply_kp5** when a polynomial of degree five is used as a kernel.

Madelom database: this database belongs to the *NIPS 2003 Feature Selection Challenge* [1]. It consists of 2000 samples of dimension 500, in which 1000 samples belong to class 1 and the other 1000 samples belong to class 2. We denote by **madelom_wk** this database without any kernel and by **madelom_kg** when a Gaussian kernel with $\sigma = 1$ is used.

Real applications databases:

Sonar database: this data was used by Gorman and Sejnowski [21] in their study of the classification of sonar signals using a neural network. It consists of 208 non linearly separable samples of dimension 60, but polynomially separable, representing data from sonar signals bouncing off a metal cylinder and sonar signals bouncing off a roughly cylindrical rock. Here 111 samples belong to class 1 and 97 samples belong to class 2. We denote this database by **sonar_kp3**, since we use a polynomial of degree three as a kernel.

Images database: it contains 6977 non linearly separable samples representing human (2429 samples) and non human (4548 samples) faces taken from *CBCL MIT* [37]. Since the number of samples is large we generated four sets of samples: **image500_kp2** containing 500 samples randomly selected from the original database; **image1500_kp2** with 1500 samples randomly selected from the original database; **image4500_kp2** consisting of 4500 samples randomly taken from the original database; and **image6977_kp2** the whole database. Each sample set maintains the human to non human faces proportion of the complete database. For this database a polynomial of degree two is used as a kernel.

Arcene database: this database consists of 100 samples of dimension 10000, where 56 samples are in class 1 and 44 samples are in class 2. These samples come from mass spectrometry and contain information about cancerigenic and non cancerigenic cells [1]. Since no kernel is used for this database, we denote it by **arcene_wk**.

Dexter database: this set of data belongs to text type classification, since it tries to determine which of the Reuters articles talk about "corporate acquisitions". It contains 300 samples of dimension 20000 with 150 samples in class 1 and 150 samples in class 2. We denote it by **dexter_wk** because no kernel is used.

Gisette database: the objective of this database is to distinguish the handwriting images of the digits "9" and "4". In this database there are 6000 samples of dimension 5000 with 3000 samples in class 1 and the other 3000 samples in class 2. We denote it by **gisette_kp2** since a polynomial of degree two is used as a kernel.

The Arcene, Dexter and Gisette databases can be found in *NIPS 2003 Feature Selection Challenge* [1].

Ada database: this database intends to determine persons with high income from census data. It is composed by 4147 samples of dimension 48, where 318 of those are in class 1 and the remaining ones are in class 2. We denote by **ada_wk** the database without kernel and by **ada_kg** the same database when a Gaussian kernel with $\sigma = 0.5$ is used.

Gina database: it is considered as a handwriting digit recognition problem, since it tries to find out if the handwriting image of a digit is odd or even. This database has 3153 samples of dimension 970, where 1603 samples are in class 1 and 1550 are in class 2. We denote it by **gina_wk** since no kernel is used.

Hiva database: it is related to data representing active and non active components again HIV AIDS. It consists of 3845 samples of dimension 1617, where 3710 are in class 1 and 135 are in class 2. We denote it by **hiva_wk**.

Sylva database: samples in this database contains information about forest types. It has 13086 samples of dimension 216. We select a subset of 7500 samples from this database, such that 7047 samples are in class 1 and 453 are in class 2. We denote this subset of data by **sylva_kp2** since a polynomial of degree two is used as a kernel.

The Ada, Gina, Hiva and Sylva databases are part of the *WCCI Performance Prediction Challenge* [2].

5.2 Numerical Results

The stopping criterium used in the SPG method is given by

$$\|P_\Omega\big(\lambda^k - \nabla f(\lambda^k)\big) - \lambda^k\|_p \le tol_p \quad \text{for} \quad p = 2, \infty, \tag{38}$$

where $tol_2 = 10^{-4}$ and $tol_\infty = 10^{-8}$. The other input parameters for the SPG method are: $\alpha_{\min} = 10^{-10}$, $\alpha_{\max} = 10^{10}$, $M = 10$, $\gamma = 10^{-10}$, $\sigma_1 = 0.1$ and $\sigma_2 = 0.9$. For more details about these input parameters see [5]. Additionally, the maximum number of function evaluations and iterations are 750000 and 500000 respectively.

In order to compare the numerical results attained by the SPG method with the three different projection strategies, we set: in the general Dykstra algorithm, the following stopping criteria,

$$\|I_{\mathcal{B}}^{i-1} - I_{\mathcal{B}}^{i}\|_F^2 \leqslant 10^{-12},$$

and the maximum number of projections for iteration to 10000. For the Dai and Fletcher projection scheme, we used $|r(\lambda)| \le 10^{-12}$ or $\Delta\lambda \le 10^{-14}$ as tolerances for the two criteria proposed by these authors in [14]. The proposed Projected-KKT is stopped when conditions (32), (33), (34) and (35) are satisfied or the maximum number of iterations per iteration is 5000.

The initial iterate for the SPG method is randomly generated and is the same for the three projection strategies used.

The algorithms were implemented using a fortran 90 compiler and double precision. The experiments were run in a PC with Intel Core 2 Duo processor, 2.13 MHz and 2 GB RAM memory.

The Table 1 summarizes the main features of the databases considered in this work. The variables **m** and **n** represent the number of samples in each database and their dimension respectively; **class 1** and **class 2** in columns 4 and 5 correspond to the amount of samples in each class. In the column 6 denoted by **kernel**, the words: **wk** means no kernel is used, **kpd** means a polynomial kernel of degree **d** is considered, and **kg(**σ**)** means that a Gaussian kernel with parameter σ is utilized. This table also includes the rank of the matrix D in problem (6) for each database, and it is denoted by **rankD**. The last column, denoted by **condD**, shows the condition number of matrix D for each database.

Database	**m**	**n**	**class 1**	**class 2**	**kernel**	**rankD**	**condD**
lineal_wk	250	2	126	124	wk	2	3.77E+14
sinoid_kp5	250	2	123	127	kp5	21	4.38E+14
polin3_kp3	250	2	118	132	kp3	10	3.81E+14
norm2_kg	250	2	144	106	kg(0.30)	126	3.01E+15
riply_wk	250	2	125	125	wk	2	4.70E+14
riply_kp5	250	2	125	125	kp5	21	4.42E+14
madelon_wk	2000	500	1000	1000	wk	500	1.86E+14
madelon_kg	2000	500	1000	1000	kg(1.00)	2000	3.86E+04
sonar_kp3	208	60	111	97	kp3	208	6.84E+05
image500_kp2	500	283	326	174	kp2	499	1.41E+17
image1500_kp2	1500	283	977	523	kp2	1412	1.07E+16
image4500_kp2	4500	283	2933	1567	kp2	4497	1.30E+18
image6977_kp2	6977	283	4548	2429	kp2	6886	1.76E+17
arcene_wk	100	10000	44	56	wk	100	4.30E+03
dexter_wk	300	20000	150	150	wk	300	2.20E+02
gisette_kp2	6000	5000	3000	3000	kp2	6000	3.99E+03
ada_wk	4147	48	1029	3118	wk	42	2.19E+14
ada_kg	4147	48	1029	3118	kg(0.50)	1929	8.84E+15
gina_wk	3153	970	1550	1603	wk	970	3.33E+14
hiva_wk	3845	1617	135	3710	wk	1410	5.99E+14
sylva7500_kp2	7500	216	453	7047	kp2	3861	4.85E+14

Table 1 – Database features.

The Table 2 shows the CPU time required for the SPG method with the three projection strategies. The columns denoted by **Dykstra**, **DaiFlet** and **ProjKKT** contain the CPU time for SPG method with Dykstra's method, Dai and Fletcher projection scheme and the proposed Projected-KKT algorithm, respectively. The numbers inside parenthesis represent the percentage of CPU time consumed by each projection strategy in the SPG method. The best CPU time for each database and for each value of C is highlighted on Table 2. The asterisk mark (*) on the right side of CPU times means that the maximum number of iterations established in the SPG method was reached before convergence.

Tables 3, 4 and 5 show the performance of the SPG method for different values of the parameter C when the projection over the set Ω is computed by Dykstra method, Dai and Fletcher strategy and the proposed Projected-KKT, respectively. The third column in these tables is denoted by **Flag** and the values of this parameter could be 0 (meaning problem (6) was solved); 4 (meaning that the projection algorithm reached the maximum number of iterations allowed; 2 (the SPG method reached the maximum number of iterations allowed); and 3 (meaning the SPG method requires more than the maximum number of function evaluations allowed). In these tables the variables **#Iter** in column 4, **#EvalF** in column 5, **MaxPIter** in column 6, indicate the number of iterations, number of function evaluations utilized for the SPG method, and the maximum number of iterations used by the projection strategy, respectively. The number appearing in column 7, denoted by **MeanPIter** gives the average number of iterations for the projection algorithm. Finally, columns 8 and 9, denoted by **pgtwon** and **pginfn**, represent the values of the stopping criteria given by expression (38) when $p = 2$ and $p = \infty$, respectively.

The last table, Table 6, illustrates the type of solution obtained by the SPG method with different values of the parameter C for the three algorithms for projecting over the set Ω. Columns 4, 5 and 6 show the number of support vectors that the optimal solution contains for each projection strategy. The last three columns of this table present the percentage of samples badly classified i.e., samples that appear located on the wrong hyperplane.

In all the tables presented in this work, observe that for some databases, we do not present the numerical results for all C, since the numerical behavior of the the optimization technique is the same as the one obtained with the biggest value of C shown in the tables for each database.

Database	**C**	**Dykstra**	**DaiFlet**	**ProjKKT**
lineal_wk	10	1.812 (96.91) (*)	**0.116** (24.14)	0.236 (74.58)
lineal_wk	100	17.269 (97.80)	**0.604** (30.46)	0.688 (31.98)
lineal_wk	1000	97.662 (97.58) (*)	4.104 (31.38)	**2.292** (25.48)
lineal_wk	10000	0.832 (80.77) (*)	36.774 (44.16)	**31.306** (24.97)
sinoid_kp5	10	15.989 (98.37)	**0.328** (32.93)	0.492 (39.84)
sinoid_kp5	100	54.823 (97.21)	**2.008** (27.29)	2.048 (29.49)
sinoid_kp5	1000	325.988 (97.31)	11.189 (28.85)	**10.077** (27.43)
sinoid_kp5	10000	1125.946 (96.88) (*)	61.856 (35.25)	**57.644** (28.80)
polin3_kp3	10	6.968 (98.51)	**0.148** (29.73)	0.160 (35.00)
polin3_kp3	100	21.085 (97.42)	**0.800** (31.50)	0.884 (35.29)
polin3_kp3	1000	246.543 (97.72) (*)	**10.365** (28.44)	10.813 (23.60)
polin3_kp3	10000	182.647 (90.37) (*)	**114.367** (35.76)	134.192 (27.74)
norm2_kg	10	4.916 (97.80)	0.152 (50.00)	**0.132** (18.18)
norm2_kg	100	11.237 (96.94)	**0.388** (30.93)	0.436 (26.61)
norm2_kg	1000	89.650 (97.07)	**4.256** (32.61)	22.997 (29.78)
norm2_kg	10000	203.869 (96.39)	14.065 (39.93)	**13.049** (34.49)
riply_wk	10	6.196 (98.32) (*)	**0.220** (40.00)	0.356 (55.06)
riply_wk	100	18.165 (97.20)	0.732 (37.16)	**0.560** (23.57)
riply_wk	1000	148.537 (97.21)	7.536 (45.22)	**5.028** (26.25)
riply_wk	10000	13.561 (73.69) (*)	68.348 (51.07)	**47.239** (23.93)
riply_kp5	10	26.734 (97.58)	1.248 (34.30)	**1.076** (39.78)
riply_kp5	100	286.534 (97.21)	13.741 (36.01)	**12.013** (40.39)
riply_kp5	1000	4030.092 (97.16) (*)	255.900 (44.60) (*)	**152.574** (39.96)
riply_kp5	10000	136.653 (82.71) (*)	260.588 (48.43) (*)	199.184 (31.51) (*)
madelon_wk	10	64.812 (58.46)	**31.098** (6.55)	33.562 (3.67)
madelon_wk	100	604.186 (53.08)	**300.703** (7.03)	304.187 (3.66)
madelon_wk	1000	4665.764 (48.00) (*)	7881.801 (4.26) (*)	3116.455 (3.36) (*)
madelon_wk	10000	5194.993 (38.58) (*)	11391.840 (11.02) (*)	10506.413 (3.58) (*)
madelon_kg	10	14.993 (31.16)	12.789 (4.44)	**11.285** (4.08)
madelon_kg	100	17.017 (14.41)	16.245 (3.30)	**15.153** (4.12)
sonar_kp3	10	4.588 (88.75)	**0.868** (33.18)	0.932 (38.63)
image500_kp2	10	4.036 (90.19)	**0.436** (22.94)	0.440 (30.00)
image1500_kp2	10	61.596 (84.26)	**11.257** (6.75)	11.905 (7.90)
image4500_kp2	10	1195.367 (64.18)	**385.496** (2.48)	412.634 (2.85)
image6977_kp2	10	5580.069 (60.60)	**2161.987** (1.65)	2400.370 (2.12)
arcene_wk	10	0.116 (100.00)	**0.036** (55.56)	**0.036** (77.78)
dexter_wk	10	0.044 (63.64)	**0.020** (40.00)	**0.020** (20.00)
gisette_kp2	10	56.552 (26.00)	41.791 (1.68)	**40.547** (1.81)
ada_wk	10	915.349 (87.49) (*)	**166.138** (4.32)	180.827 (5.31)
ada_kg	10	11564.371 (87.71)	**1432.045** (3.87)	1437.710 (5.68)
gina_wk	10	2141.522 (38.53)	1341.688 (2.62)	**1331.595** (2.29)
gina_wk	100	3101.570 (39.80)	2458.054 (2.90)	**1999.141** (2.22)
hiva_wk	10	1856.160 (52.88)	**962.012** (2.76)	990.402 (3.87)
hiva_wk	100	12773.686 (54.11)	**6055.622** (2.69)	6243.914 (3.89)
sylva7500_kp2	10	10731.107 (68.64)	**3026.937** (1.24)	3178.783 (2.22)
sylva7500_kp2	100	26107.464 (66.87)	8151.021 (1.20)	**7837.650** (2.18)
sylva7500_kp2	1000	25235.285 (66.52)	**7628.405** (1.21)	7839.946 (2.17)

Table 2 – CPU time in seconds used for the SPG method with the three projecting techniques.

Database	**C**	**flag**	**# iter**	**# Evalf**	**MaxPIter**	**MeanPIter**	**pgtwon**	**pginfn**
lineal_wk	10	4	149	195	10001	338.20	6.93E-08	2.13E-04
lineal_wk	100	0	1104	1720	4042	443.98	1.76E-09	3.38E-05
lineal_wk	1000	4	8090	13004	10001	340.09	3.07E-04	1.20E-02
lineal_wk	10000	4	565	957	10001	32.23	1.41E+06	1.79E+02
sinoid_kp5	10	0	1102	1612	4228	411.02	2.76E-08	9.47E-05
sinoid_kp5	100	0	5287	7886	1303	292.64	7.07E-09	5.17E-05
sinoid_kp5	1000	0	30981	47114	3187	304.96	3.06E-08	9.81E-05
sinoid_kp5	10000	4	124690	180433	10001	263.55	1.40E-01	2.44E-01
polin3_kp3	10	0	416	594	4715	482.31	6.09E-09	5.43E-05
polin3_kp3	100	0	1985	2850	1087	304.42	5.18E-09	4.04E-05
polin3_kp3	1000	4	19275	30896	10001	372.00	3.29E-06	1.40E-03
polin3_kp3	10000	4	63626	92143	10001	73.64	8.80E-01	4.57E-01
norm2_kg	10	0	405	576	743	344.61	1.25E-08	6.19E-05
norm2_kg	100	0	986	1507	701	329.10	3.36E-08	8.97E-05
norm2_kg	1000	0	9046	14553	2856	287.65	1.63E-08	8.32E-05
norm2_kg	10000	0	25671	39051	790	230.08	2.75E-08	8.96E-05
riply_wk	10	4	310	421	10001	559.02	8.95E-06	2.19E-03
riply_wk	100	0	1727	2498	3625	289.84	2.73E-12	1.18E-06
riply_wk	1000	0	15092	21686	4090	270.53	2.90E-14	1.39E-07
riply_wk	10000	4	13099	18988	10001	21.00	1.42E+02	2.15E+00
riply_kp5	10	0	2305	3383	1210	325.66	1.58E-08	6.68E-05
riply_kp5	100	0	28173	42603	4552	283.82	1.80E-08	9.26E-05
riply_kp5	1000	4	400159	606886	10001	280.17	1.08E-06	7.84E-04
riply_kp5	10000	4	89954	115015	10001	34.21	8.89E+01	1.98E+00
madelon_wk	10	0	1168	1832	356	109.95	2.91E-07	9.87E-05
madelon_wk	100	0	11977	19696	688	89.04	5.84E-07	1.00E-04
madelon_wk	1000	4	101695	169778	10001	72.63	2.07E-05	6.40E-04
madelon_wk	10000	4	150051	231844	10001	45.05	2.95E+00	2.05E-01
madelon_kg	10	0	462	691	56	34.15	7.59E-07	9.87E-05
madelon_kg	100	0	634	992	44	14.23	1.68E-07	9.96E-05
sonar_kp3	10	0	2734	4428	112	50.02	1.41E-07	9.83E-05
image500_kp2	10	0	310	461	592	179.85	3.96E-08	6.03E-05
image1500_kp2	10	0	786	1236	910	345.44	9.78E-08	8.27E-05
image4500_kp2	10	0	3738	5929	752	358.00	1.73E-07	9.77E-05
image6977_kp2	10	0	8225	12606	975	465.32	2.10E-08	9.90E-05
arcene_wk	10	0	329	504	80	19.26	6.34E-08	9.97E-05
dexter_wk	10	0	44	46	19	14.00	3.62E-08	5.58E-05
gisette_kp2	10	0	209	296	220	87.27	5.37E-08	9.28E-05
ada_wk	10	4	1229	1817	10001	1104.82	1.88E-05	2.82E-03
ada_kg	10	0	14796	22983	2794	1176.12	8.18E-08	9.59E-05
gina_wk	10	0	22965	36386	122	82.90	5.76E-07	9.92E-05
gina_wk	100	0	33196	53059	122	86.28	8.10E-07	9.95E-05
hiva_wk	10	0	10056	16020	529	193.27	7.48E-08	9.99E-05
hiva_wk	100	0	67240	107262	526	203.80	3.12E-08	9.24E-05
sylva7500_kp2	10	0	10387	16233	1169	748.94	2.07E-07	9.71E-05
sylva7500_kp2	100	0	26693	41740	968	692.03	4.87E-08	1.00E-04
sylva7500_kp2	1000	0	25749	40643	964	689.03	8.00E-08	6.78E-05

Table 3 – Performance of the SPG method with Dykstra projecting technique.

Database	C	flag	# iter	# Evalf	MaxPIter	MeanPIter	pgtwon	pginfn
lineal_wk	10	0	261	372	26	4.34	1.20E-08	7.73E-05
lineal_wk	100	0	1397	2267	17	3.68	2.37E-10	1.24E-05
lineal_wk	1000	0	9310	14722	19	4.52	2.61E-09	3.76E-05
lineal_wk	10000	0	68365	110268	30	9.68	8.93E-09	7.71E-05
sinoid_kp5	10	0	850	1281	9	1.87	2.96E-08	9.95E-05
sinoid_kp5	100	0	5129	7647	11	2.39	3.18E-08	9.53E-05
sinoid_kp5	1000	0	27839	41901	14	3.27	2.32E-08	9.25E-05
sinoid_kp5	10000	0	142689	210385	24	5.47	3.62E-08	8.81E-05
polin3_kp3	10	0	374	536	10	2.63	1.82E-08	8.78E-05
polin3_kp3	100	0	2025	3001	10	2.79	1.06E-08	8.17E-05
polin3_kp3	1000	0	24839	40828	19	3.35	1.70E-08	7.93E-05
polin3_kp3	10000	0	263314	379482	23	5.58	1.66E-08	8.97E-05
norm2_kg	10	0	375	504	17	4.22	7.72E-09	5.19E-05
norm2_kg	100	0	899	1345	19	4.10	2.21E-08	7.59E-05
norm2_kg	1000	0	9555	15487	26	4.95	2.24E-08	8.85E-05
norm2_kg	10000	0	29213	45204	22	7.46	7.14E-08	9.84E-05
riply_wk	10	0	462	707	24	5.95	7.90E-09	6.61E-05
riply_wk	100	0	1689	2463	18	4.51	1.99E-09	3.16E-05
riply_wk	1000	0	15029	22005	26	8.22	1.08E-16	8.44E-09
riply_wk	10000	0	119819	173767	34	12.00	2.69E-10	1.34E-05
riply_kp5	10	0	2865	4233	17	4.56	3.46E-08	9.46E-05
riply_kp5	100	0	31194	46661	20	4.84	3.56E-08	8.80E-05
riply_kp5	1000	2	500001	747304	27	8.69	2.05E-06	8.80E-04
riply_kp5	10000	2	500001	677417	26	9.67	2.07E+01	1.00E+00
madelon_wk	10	0	1263	1972	21	6.86	6.51E-08	4.51E-05
madelon_wk	100	0	11754	19371	25	7.83	5.41E-07	9.31E-05
madelon_wk	1000	3	157426	750000	61	9.51	5.33E-06	3.02E-04
madelon_wk	10000	3	461746	750000	28	13.27	1.07E-02	1.62E-02
madelon_kg	10	0	553	817	5	2.94	4.85E-07	9.40E-05
madelon_kg	100	0	673	1079	7	2.83	4.18E-07	4.60E-05
sonar_kp3	10	0	2707	4421	8	3.79	3.65E-08	7.15E-05
image500_kp2	10	0	282	411	9	5.54	1.16E-08	7.10E-05
image1500_kp2	10	0	865	1335	11	6.02	4.95E-08	9.64E-05
image4500_kp2	10	0	3275	5215	12	6.91	6.68E-08	8.97E-05
image6977_kp2	10	0	8107	12040	12	6.60	6.60E-08	9.56E-05
arcene_wk	10	0	324	483	6	3.33	5.73E-08	9.85E-05
dexter_wk	10	0	44	46	4	2.87	3.62E-08	5.58E-05
gisette_kp2	10	0	202	291	9	4.54	1.58E-07	8.04E-05
ada_wk	10	0	1725	2509	26	9.52	2.73E-08	9.92E-05
ada_kg	10	0	14392	22297	23	8.72	4.99E-08	9.73E-05
gina_wk	10	0	23177	37135	10	3.26	5.40E-07	9.55E-05
gina_wk	100	0	42692	67484	9	4.03	8.10E-07	1.00E-04
hiva_wk	10	0	10699	17130	16	6.16	4.50E-08	9.93E-05
hiva_wk	100	0	67675	107838	15	5.99	1.50E-07	6.46E-05
sylva7500_kp2	10	0	9234	14486	18	5.04	7.83E-08	9.68E-05
sylva7500_kp2	100	0	24618	38663	15	4.80	4.77E-08	9.69E-05
sylva7500_kp2	1000	0	23118	36102	18	4.80	1.21E-07	9.62E-05

Table 4 – Performance of the SPG method with Dai and Fletcher projecting technique.

Database	C	flag	# iter	# Evalf	MaxPIter	MeanPIter	pgtwon	pginfn
lineal_wk	10	0	237	349	1341	13.80	1.06E-08	7.27E-05
lineal_wk	100	0	1584	2521	1410	1.79	6.37E-10	2.04E-05
lineal_wk	1000	0	5704	9098	55	0.96	2.60E-09	3.73E-05
lineal_wk	10000	0	78039	126123	84	0.94	1.12E-10	7.84E-06
sinoid_kp5	10	0	1092	1635	13	2.25	5.89E-09	4.07E-05
sinoid_kp5	100	0	5099	7572	27	1.34	1.52E-08	6.45E-05
sinoid_kp5	1000	0	25460	38532	98	1.26	3.97E-08	9.41E-05
sinoid_kp5	10000	0	145933	215893	137	1.41	3.06E-08	9.98E-05
polin3_kp3	10	0	379	532	174	1.88	9.55E-09	8.02E-05
polin3_kp3	100	0	2139	3076	5	1.74	2.25E-08	9.94E-05
polin3_kp3	1000	0	27343	45103	270	0.92	1.71E-08	9.52E-05
polin3_kp3	10000	0	345391	501426	22	1.25	1.02E-08	7.29E-05
norm2_kg	10	0	361	502	3	0.95	4.11E-09	3.04E-05
norm2_kg	100	0	1130	1704	3	1.03	1.21E-08	7.58E-05
norm2_kg	1000	0	56112	84443	6	1.66	2.60E-08	9.61E-05
norm2_kg	10000	0	29618	45864	5	2.31	2.24E-08	9.73E-05
riply_wk	10	0	531	841	2980	6.45	8.67E-09	7.17E-05
riply_wk	100	0	1510	2169	2	0.49	2.96E-14	1.40E-07
riply_wk	1000	0	13524	19900	20	0.48	2.07E-10	1.09E-05
riply_wk	10000	0	127330	185203	3	0.49	2.33E-14	1.13E-07
riply_kp5	10	0	2395	3453	7	2.19	2.08E-08	9.46E-05
riply_kp5	100	0	25414	37950	102	2.69	9.16E-09	6.92E-05
riply_kp5	1000	0	313946	493303	157	2.82	2.70E-08	9.82E-05
riply_kp5	10000	2	500001	678177	4	1.24	3.71E+03	2.72E+01
madelon_wk	10	0	1421	2189	2	0.88	6.60E-08	6.01E-05
madelon_wk	100	0	12410	20345	2	0.88	5.36E-07	9.83E-05
madelon_wk	1000	4	133290	227180	5001	0.64	6.54E-03	1.15E-02
madelon_wk	10000	3	462742	750000	2	0.67	1.38E-01	4.35E-02
madelon_kg	10	0	486	727	2	0.86	2.68E-07	7.75E-05
madelon_kg	100	0	635	997	2	0.86	2.27E-07	7.70E-05
sonar_kp3	10	0	2853	4576	3	1.61	1.26E-08	8.46E-05
image500_kp2	10	0	279	405	5	2.96	1.20E-08	7.34E-05
image1500_kp2	10	0	913	1394	6	2.91	2.81E-07	9.97E-05
image4500_kp2	10	0	3464	5590	7	3.77	5.11E-08	9.82E-05
image6977_kp2	10	0	9032	13228	8	4.09	9.51E-09	1.39E-05
arcene_wk	10	0	330	502	2	0.89	6.93E-08	8.83E-05
dexter_wk	10	0	44	46	2	0.87	3.62E-08	5.58E-05
gisette_kp2	10	0	203	279	4	2.07	1.47E-09	1.89E-05
ada_wk	10	0	1851	2709	19	5.19	4.04E-08	9.00E-05
ada_kg	10	0	14152	22010	8	5.83	5.68E-08	9.95E-05
gina_wk	10	0	23165	36956	3	1.11	5.77E-07	9.84E-05
gina_wk	100	0	34648	55603	3	1.10	6.85E-07	9.47E-05
hiva_wk	10	0	11006	17332	6	4.19	4.87E-08	9.95E-05
hiva_wk	100	0	68881	109814	6	4.29	4.12E-08	9.80E-05
sylva7500_kp2	10	0	9683	14963	10	4.95	4.86E-08	6.52E-05
sylva7500_kp2	100	0	23404	36845	6	4.96	1.58E-07	9.95E-05
sylva7500_kp2	1000	0	23404	36845	6	4.96	1.58E-07	9.95E-05

Table 5 – Performance of the SPG method with Projected-KKT technique.

			# support vectors			% of badly classified data		
Database	**m**	**C**	**Dykstra**	**DaiFlet**	**ProjKKT**	**Dykstra**	**DaiFlet**	**ProjKKT**
lineal_wk	250	10	3	2	2	5.60	5.60	5.60
lineal_wk	250	100	3	3	3	4.80	4.80	4.80
lineal_wk	250	1000	6	3	3	4.80	4.80	4.80
lineal_wk	250	10000	143	3	3	34.40	4.80	4.80
sinoid_kp5	250	10	9	9	9	21.60	21.60	21.60
sinoid_kp5	250	100	11	11	11	5.60	5.60	5.60
sinoid_kp5	250	1000	11	11	11	3.60	3.60	3.60
sinoid_kp5	250	10000	13	12	12	5.60	5.20	5.20
polin3_kp3	250	10	5	5	5	2.40	2.40	2.40
polin3_kp3	250	100	7	7	7	2.40	2.40	2.40
polin3_kp3	250	1000	10	9	9	5.20	5.60	5.60
polin3_kp3	250	10000	32	10	10	5.20	29.60	29.60
norm2_kg	250	10	12	12	12	6.40	6.40	6.40
norm2_kg	250	100	13	13	13	2.80	2.80	2.80
norm2_kg	250	1000	18	18	18	7.60	7.60	7.60
norm2_kg	250	10000	18	18	18	40.00	40.00	40.00
riply_wk	250	10	4	3	4	14.00	14.00	14.00
riply_wk	250	100	3	3	3	13.60	13.60	13.60
riply_wk	250	1000	3	3	3	14.00	14.00	14.00
riply_wk	250	10000	192	3	3	27.60	14.00	14.00
riply_kp5	250	10	9	9	9	18.40	18.40	18.40
riply_kp5	250	100	10	10	10	24.00	24.00	24.00
riply_kp5	250	1000	15	15	14	24.00	24.00	24.00
riply_kp5	250	10000	154	80	77	25.20	16.80	24.40
madelon_wk	2000	10	325	325	325	27.05	27.05	27.05
madelon_wk	2000	100	429	429	429	25.10	25.10	25.10
madelon_wk	2000	1000	487	484	1339	23.60	23.60	23.15
madelon_wk	2000	10000	701	533	532	23.15	23.00	22.85
madelon_kg	2000	10	897	897	897	5.55	5.55	5.60
madelon_kg	2000	100	1712	1712	1712	46.05	46.05	45.90
sonar_kp3	208	10	87	87	87	0.00	0.00	0.00
image500_kp2	500	10	75	75	75	60.80	60.80	60.80
image1500_kp2	1500	10	91	91	91	65.13	65.13	65.13
image4500_kp2	4500	10	327	327	327	65.18	65.18	65.18
image6977_kp2	6977	10	390	390	390	63.57	63.57	63.57
arcene_wk	100	10	79	79	79	0.00	0.00	0.00
dexter_wk	300	10	257	257	257	0.00	0.00	0.00
gisette_kp2	6000	10	151	151	151	49.87	49.87	49.87
ada_wk	4147	10	19	8	8	19.89	19.89	19.92
ada_kg	4147	10	52	51	51	23.63	23.63	23.63
gina_wk	3153	10	756	756	756	0.25	0.25	0.25
gina_wk	3153	100	752	752	752	0.00	0.00	0.00
hiva_wk	3845	10	514	514	514	0.03	0.03	0.03
hiva_wk	3845	100	515	514	515	0.03	0.03	0.03
sylva7500_kp2	7500	10	221	221	221	4.48	4.48	4.48
sylva7500_kp2	7500	100	262	262	262	9.96	9.97	9.96
sylva7500_kp2	7500	1000	262	262	262	9.97	9.95	9.96

Table 6 – Training results.

Figures 1, 2, 3, 4, 5 and 6 show the real separator curves and the computed separator curves given by equations (3) and (4) using the optimal solution attained by the SPG method with Projected-KKT for different values of C. The figures for the SPG method with Dai and Fletcher projection strategy are very similar to Figures 1, 2, 3, 4, 5 and 6 and therefore they are not shown here. In the scale legend of these figures appears the symbols utilized to distinguish samples in class 1 from samples in class 2. Also, it is possible to distinguish from all the samples which of them are the support vectors in class 1 and the support vectors in class 2. It is clear that these figures belong to databases with samples of dimension two: (**lineal_wk**, **sinoid_kp5**, **polin3_kp3**, **norm2_kg**, **riply_wk** and **riply_kp5**), since they can be easily drawn.

From Table 2 observe that SPG method together with Dykstra alternating projection scheme requires more CPU time than the SPG method with the others two projection strategies for all databases and all values of C. Inclusive, for some values of C and some databases this method does not converge to an stationary point of problem (6). The SPG-Dykstra method is stopped since the maximum number of SPG iterations (500000) or the maximum number of Dykstra iterations (10000) was reached before convergence occurs. Moreover, the percentage average of CPU time consumed for Dykstra's algorithm in the SPG method for all databases and all values of C, is 54.76 %. In many cases, as for example databases: **lineal_wk**, **sylva7500_kp2**, **ada_kg**, **ada_wk**, **sonar_kp3** and **riply_kp5**, the percentage of CPU time consumed for Dykstra algorithm is higher than the percentage average. Observe that for **ada_kg** and $C = 10$ the CPU time utilized for SPG method with Dykstra strategy is 8 times the CPU time required for SPG method with Dai and Fletcher scheme. Also, for database **norm2_kg** with $C = 10000$, the CPU time required for SPG method with Dykstra algorithm is 15 times the CPU time for SPG method with Projected-KKT. The average percentage of CPU time used for Dai and Fletcher scheme and Projected-KKT strategy in the SPG method are the 20.97% and 21.03% respectively, for all problems solved. Moreover, in terms of efficiency of the three optimization methods, the SPG method using Projected-KKT has a slightly higher percentage of problems solved. So, the difference in average CPU time consumed for these two strategies is irrelevant.

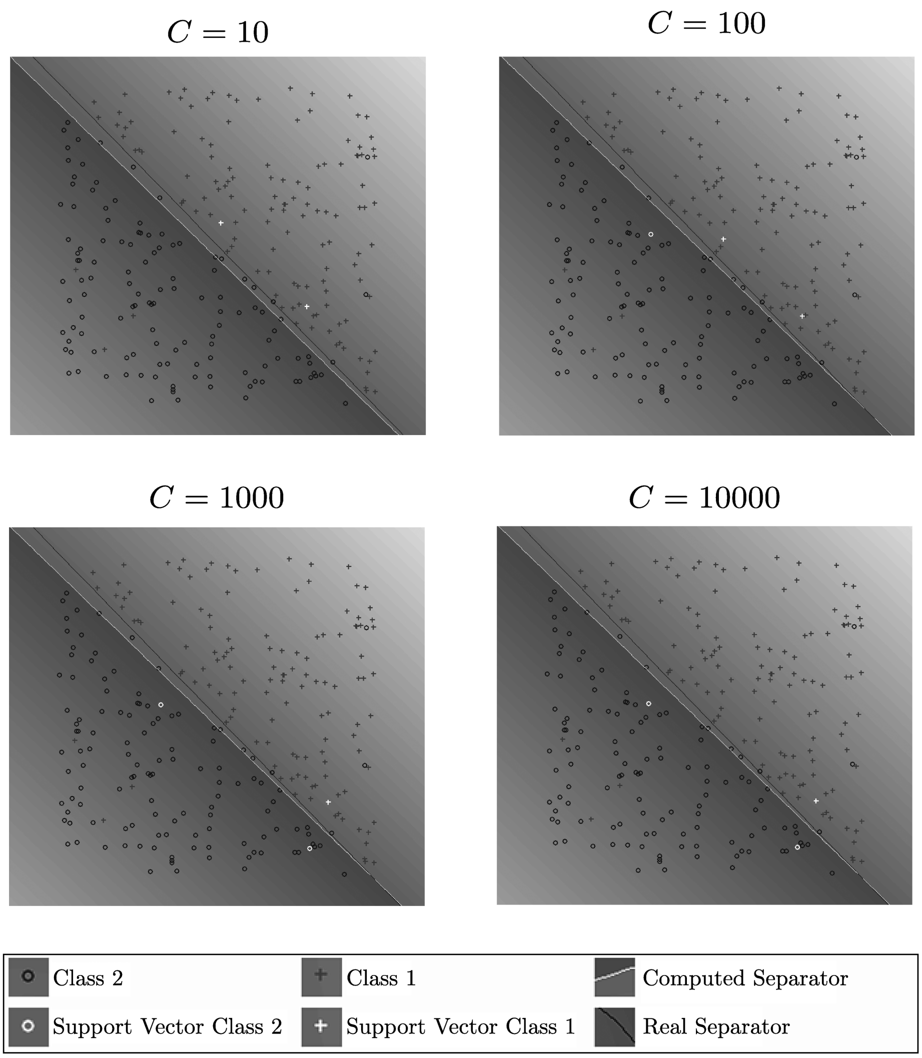

Figure 1 – Classification of samples in database **lineal_wk**.

From Tables 2, 3, 4 and 5, it is clear that the computational performance of the SPG method using Dai and Fletcher projection scheme and using the proposed Projected-KKT is very similar. We can say that both strategies are competitive for solving problem (6) in high contrast with the SPG method with Dykstra's algorithm. For example, for databases **polin3_kp3**, **Sonar**, **Images**, **Ada** and **Hiva** the performance of the SPG method using Dai and Fletcher is

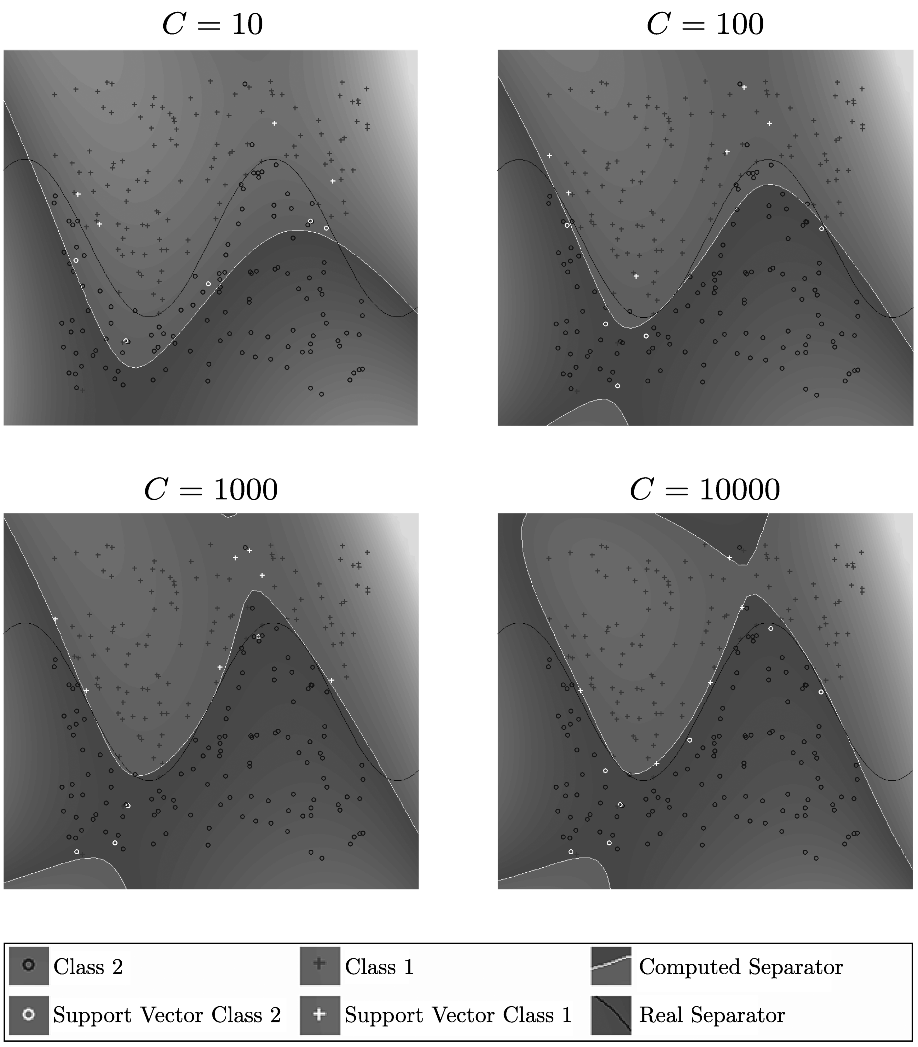

Figure 2 – Classification of samples in database **sinoid_kp5**.

a little better than the performance of the SPG method with Projected-KKT. On the other hand, for databases **Riply**, **Gisette** and **Gina** the SPG method with Projected-KKT shows a better computational behavior when it is compared with SPG method using Dai and Fletcher scheme. For the rest of databases, the best computational performance varies with the method, the kernel used and the value of the parameter C. From now on, we focus our analysis only on the

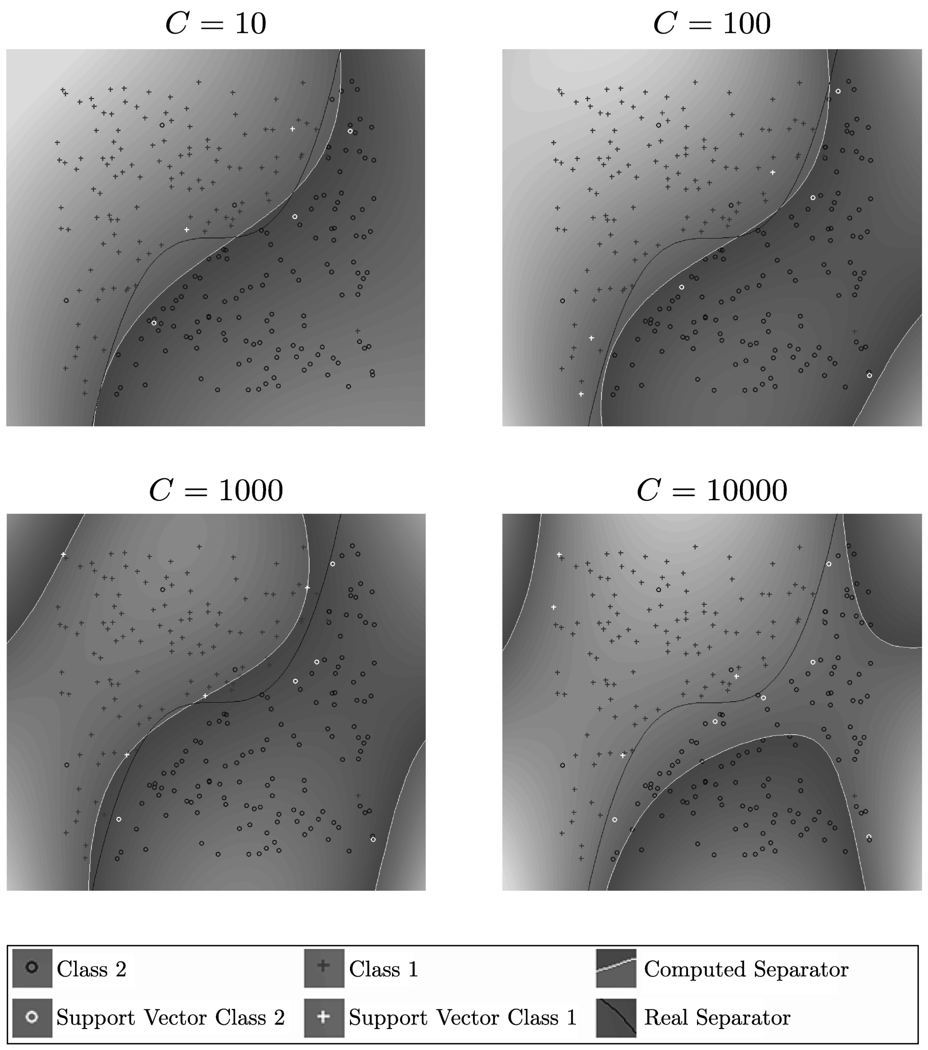

Figure 3 – Classification of samples in database **polin3_kp3**.

SPG method with Dai and Fletcher and Projected-KKT schemes since Dykstra's algorithm is not competitive with the these two techniques.

Even when Dai and Fletcher projection strategy and Projected-KKT method are very precise computing the projection required by the SPG method, small differences in the projection generate different iterations. These differences increase when the matrix D on problem (6) associated to the database is badly

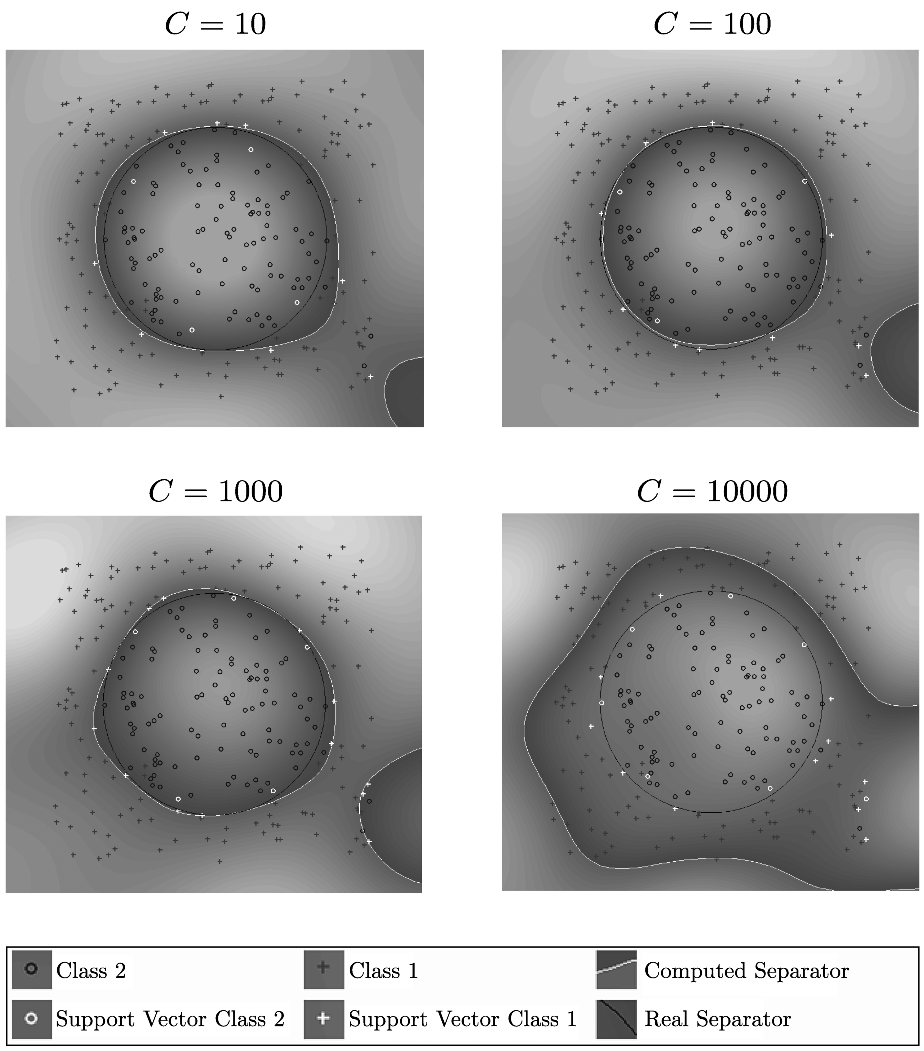

Figure 4 – Classification of samples in database **norm2_kg**.

conditioned. It is clear from tables 3, 4 and 5, that when SPG method uses few iterations, the computational behavior of the two competitive strategies is similar, as for example for databases: **Arcene**, **Dexter** and **Gisette**, where the associated matrix D is not bad conditioned when it is compared with the rest of the databases. In problems where the SPG method uses more iterations, it seems that the difference in the computed projection by Dai and Fletcher

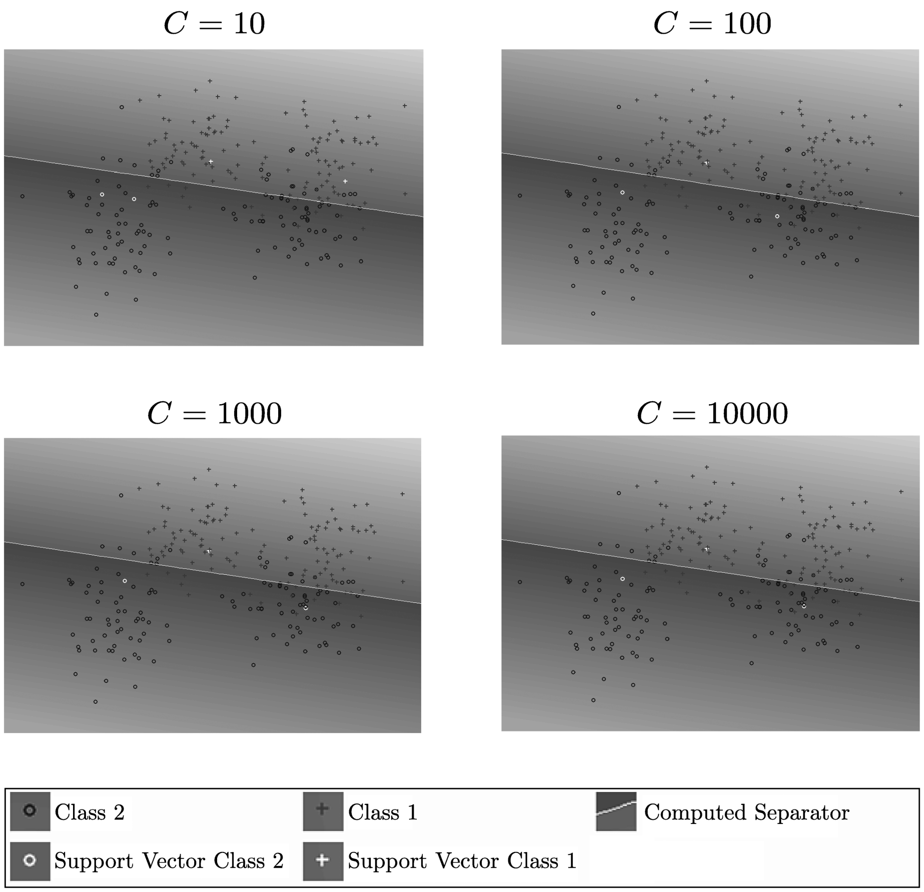

Figure 5 – Classification of samples in database **riply_wk**.

and Projected-KKT increases, making that the SPG with these two projecting strategies have different computational behaviors as for example in databases **Madelon**, **Gina** and **Riply**, where the matrix D is badly conditioned. Another important observation is that the average of projection iterations for SPG method with Dai and Fletcher strategy and Projected-KKT scheme is not necessarily less when the SPG method consumes less iterations. The computational performance of these two methods varies depending of the database (condition number of the matrix D) and the value of C.

It is also important to stress that for databases where the condition number of the matrix D is high, as for example the databases: **lineal_wk**, **sinoid_kp5**

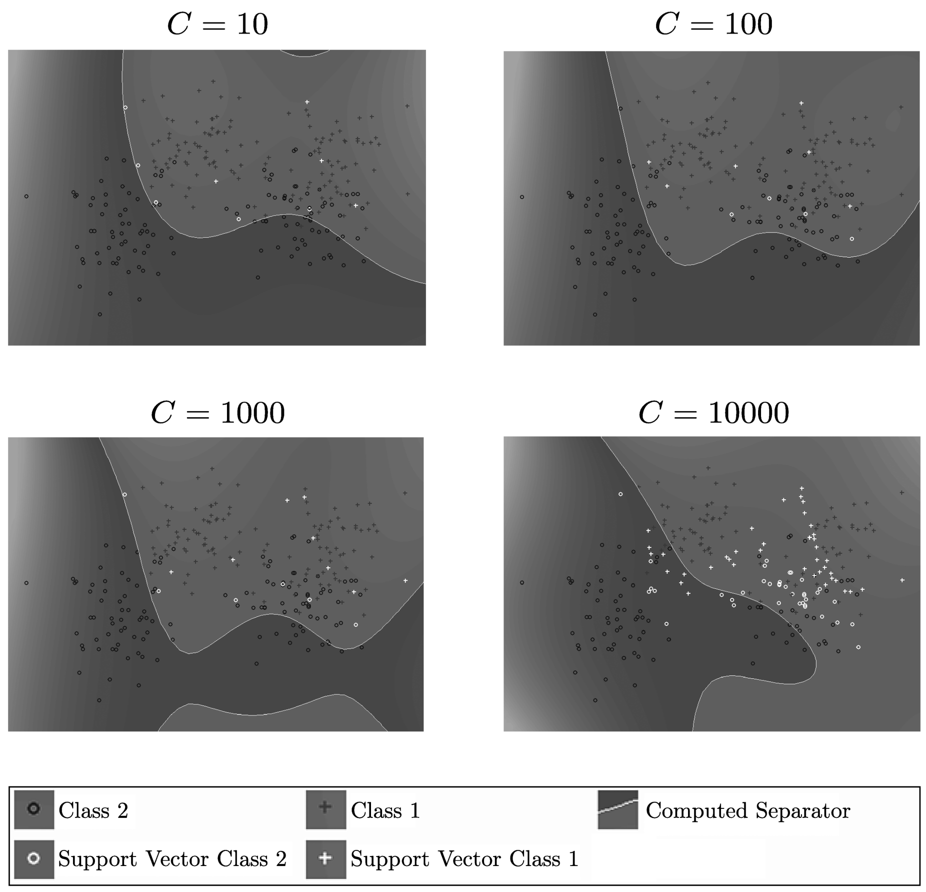

Figure 6 – Classification of samples in database **riply_kp5**.

and **norma2_kg**, the computational behavior of the two competitive strategies is similar, but in these cases the dimension of the samples is two. So, the difficulty on solving the SVM problem (6) also depend on the size of the problem.

Table 6 indicates that the solutions obtained for SPG method with Dai and Fletcher strategy and Projected-KKT scheme are almost equal for each database and each value of C.

Finally, we observe from Figures 1, 2, 3, 4, 5 and 6 that the proposed SPG with the Projected-KKT method classifies the samples on those databases. For **lineal_wk** data (Fig. 1) the real level zero curve is very close to the computed level zero curve and it seems not to depend on the values of the parameter

C. In contrast, for nonlinearly separable data (**sinoid_wk**, **polin3_kp3** and **norm2_wk**), the computed level zero curve highly depends on the values of parameter C and the kernel used. It is possible that in these nonlinearly separable samples the use of different kernels could get a better classification of the data. However, our interest in this work is to show that the proposed SPG with the Projected-KKT can solve the SVM problem (6), and that playing with different kernels and values of C the solution could be improved.

6 Conclusions

In this work we proposed an algorithm for projecting over a single linear constraint and a box constrained set. This algorithm is based on the Karush-Kuhn-Tucker conditions for a quadratic programming problem over the described set. To the best of our knowledge, this is the first time that a KKT-type algorithm has been used in the SVM context for projecting over the convex set. The proposed algorithm has finite termination unless cycling occurs. However, in practice for well-conditioned diagonal positive definite matrices (as the identity) cycling never occurs.

We studied how to solve the SVM optimization problem by using the Spectral Projected Gradient (SPG) method using the Projected-KKT method for solving the projection problem, as well as the secant-based algorithm proposed by Dai and Fletcher and a version of the Dykstra's method. The results obtained indicate that the Projected-KKT algorithm is competitive with the Dai and Fletcher projecting strategy within the SPG method, and that both strategies outperform the SPG method with Dykstra's algorithm. Using the Projected-KKT projection strategy more problems were solved than using the other two methods, indicating that it has a slightly higher efficiency. The number of support vectors attained by the two competitive methods is the same for almost all the problems tested, implying that both methods compute basically the same optimal separator. The computational behavior of these two methods highly depends on the size of the problem, the condition number of the Hessian matrix, the value of the parameter C, and the kernel used.

The results presented in this work are a contribution to the study of the behavior of optimization methods for solving large scale SVM problems.

Acknowledgements. The authors are grateful to M.A. Francisco and F. Vargas from Universidad Simón Bolívar for programming assistence and useful contributions.

REFERENCES

[1] Neural Information Processing Systems (NIPS) Conference: Feature Selection Challenge, 2003. http://www.nipsfsc.ecs.soton.ac.uk/datasets/.

[2] IEEE World Congress on Computational Intelligence: Performance Prediction Challenge, 2006. http://www.modelselect.inf.ethz.ch/.

[3] E. Birgin, J.M. Martínez and M. Raydan, *Inexact spectral projected gradient methods on convex sets.* IMA Journal of Numerical Analysis, **23** (2003), 539–559.

[4] E.G. Birgin and M. Raydan, *Robust stopping criteria for Dykstra's algorithm.* SIAM Journal on Scientific Computing, **26** (2005), 1405–1414.

[5] E.G. Birgin, J.M. Martínez and M. Raydan, *Nonmonotone spectral projected gradient methods on convex sets.* SIAM Journal on Optimization, **10** (2000), 1196–1211.

[6] V. Blanz, B. Schölkopf, H. Bülthoffand, C. Burges, V.N. Vapnik and T. Vetter, *Comparison of view-based object recognition algorithms using realistic 3d models.* In: J.C. Vorbrüggen, C. von der Malsburg, W. von Seelen and B. Sendhoff, editors, *Artificial Neural Networks*, ICIANN'96, pages 251–256, Berlin 1996. Springer Lecture Notes in Computer Science, **1112** (1996).

[7] B.E. Boser, I.M. Guyon and V.N. Vapnik, *A training algorithm for optimal margin classifiers.* In: *Proceedings of the 5th ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992.

[8] J.P. Boyle and R.L. Dykstra, *A method for finding projections onto the intersections of convex sets in Hilbert spaces.* Lecture Notes in Statistics, 37 (1986), 28–47.

[9] C.J.C. Burges, *A tutorial on support vector machines for pattern recognition.* In: U. Fayyad, editor, *Data Mining on Knowledge Discovery*, pages 121–167. Kluwer Academic Publishers, Netherlands (1998).

[10] C.J.C. Burges and B. Schölkopf, *Improving the accuracy and speed of support vector machines.* In: M. Jordan M. Mozer and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381. MIT Press, Cambridge, MA (1997).

[11] W. Cheney and A. Goldstein, *Proximity maps for convex sets.* Proc. Amer. Math. Soc., **10** (1959), 448–450.

[12] C. Cortes and V. Vapnik, *Support vector networks.* Machine Learning, **20** (1995), 1–25.

[13] N. Cristianini and J.S. Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge University Press, United Kingdom (2000).

[14] Y.H. Dai and R. Fletcher, *New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds.* Math. Program., **106** (2006), 403–421.

[15] J.E. Dennis Jr. and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, NJ (1983).

[16] R.L. Dykstra, *An algorithm for restricted least-squares regression.* J. Amer. Stat. Assoc., **78** (1983), 837–842.

[17] R. Escalante and M. Raydan, *Dykstra's algorithm for a constrained least-squares matrix problem.* Num. Linear Algebra Appl., **3**(6) (1996), 459–471.

[18] R. Escalante and M. Raydan, *Dykstra's algorithm for constrained least-squares rectangular matrix problems.* Computers Math. Applic., **35**(6) (1998), 73–79.

[19] M. Ferris and T. Munson, *Interior-point methods for massive support vector machines.* SIAM Journal on Optimization, **13**(3) (2003), 783–804.

[20] J.C. Gilbert and J. Nocedal, *Global convergence properties of conjugate gradient methods for optimization.* SIAM J. Optim., **2**, 21–42.

[21] R.P. Gorman and T.J. Sejnowski, *Analysis of hidden units in a layered network trained to classify sonar targets.* Neural Networks, **1** (1988), 75–89.

[22] S.P. Han, *A successive projection method.* Math. Program, **40** (1988), 1–14.

[23] H. Lin and C. Lin, *A study on sigmoid kerrnels for svm and the training of non-psd kernels by smo-type methods.* Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei Taiwan (2003).

[24] T. Joachims, *Text categorization with support vector machines: Learning with many relevant features.* In: C. Nédellec and C. Rouveirol, editors, *Proceedings of the* 10^{th} *European Conference on Machine Learning (ECML-98)*, pages 137–142. Springer, April 1998.

[25] J.J. Júdice and F.M. Pires, *Solution of large-scale separable strictly convex quadratic programs on the simplex.* Linear Algebra and its applications, (1989), 215–220.

[26] J. Nocedal and S.J. Wright, *Numerical Optimization*. Springer, New York (1999).

[27] E.E. Osuna, R. Freund and F. Girosi, *Support Vector Machines: Training and Applications.* Technical Report A.I. Memo No. 1602, C.B.C.L. Paper No. 144, Massachusetts Institute of Technology, Cambridge, MA, USA (1997).

[28] E.E. Osuna, R. Freund and F. Girosi, *Training support vector vector machines: An application to face detection.* In: *IEEE Conference on Computer Vision and Pattern Recognition*, (1997), 130–136.

[29] P.M. Pardalos and N. Kovoor, *An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds.* Mathematical Programming, 46.

[30] M. Raydan, *The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem.* SIAM J. Opt., **7** (1997), 26–33.

[31] B.D. Riply, *Neural networks and related methods for classification (with discussion).* J. Royal Statistical Soc. Series B, **56** (1994), 409–456.

[32] M. Schmidt, *Identifying speaker with support vector networks.* In: *Interface'96 Proceedings*. Sydney, Australia (1996).

[33] B. Schölkopf, C. Burges and V.N. Vapnik, *Extracting support data for a given task.* In: U.M. Fayyad and R. Uthurusamy, editors, *Proceedings First International Conference on Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, CA (1995).

[34] B. Schölkopf, C. Burges and V.N. Vapnik, *Incorporating invariances in support vector learning machines.* In: J.C. Vorbrüuggen C. von der Malsburg, W. von Seelen and B. Sendhoff, editors, *Artificial Neural Networks*, ICIANN'96, pages 47–52, Berlin 1996. Springer Lecture Notes in Computer Science, **1112** (1996).

[35] T. Serafini, G. Zanghirati and L. Zanni, *Gradient projection methods for quadratic programs and applications in training support vector machines.* Optimization Methods and Software, **20** (2003), 353–378.

[36] D.F. Shanno and K.H. Phua, *Remark on algorithm 500: Minimization of unconstrained multivariate functions.* ACM Trans. Math. Software, **6** (1980), 618–622.

[37] K. Sung and T. Poggio, *Example-based learning for view-based human face detection.* Technical Report A.I. Memo No. 1521, C.B.C.L. Paper No. 112, Massachusetts Institute of Technology, Cambridge, MA, USA (1994).

[38] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer-Verlag, New York (1995).

[39] J. von Neumann, *Functional operators, Vol. II. The geometry of orthogonal spaces.* Annals of Math. Studies, **22** (1950). Princeton University Press.

[40] J. Weston, A. Gammerman, M.O. Stitson, V. Vapnik, V. Vork and C. Watkins, *Density estimation using support vector machines.* Technical Report CSD-TR-97-23, Royal Holloway College (1997).

[41] L. Zanni, T. Serafini and G. Zanghirati, *Parallel software for training large scale support vector machines on multiprocessor systems.* Journal of Machine Learning Research, **7** (2006), 1467–1492.