An Improved Gradient Projection-based Decomposition Technique for Support Vector Machines*

L. Zanni

Dipartimento di Matematica, Università di Modena e Reggio Emilia, Modena, Italy.

Abstract

In this paper we propose some improvements to a recent decomposition technique for the large quadratic program arising in training Support Vector Machines. As standard decomposition approaches, the technique we consider is based on the idea to optimize, at each iteration, a subset of the variables through the solution of a quadratic programming subproblem. The innovative features of this approach consist in using a very effective gradient projection method for the inner subproblems and a special rule for selecting the variables to be optimized at each step. These features allow to obtain promising performance by decomposing the problem into few large subproblems instead of many small subproblems as usually done by other decomposition schemes. We improve this technique by introducing a new inner solver and a simple strategy for reducing the computational cost of each iteration. We evaluate the effectiveness of these improvements by solving large-scale benchmark problems and by comparison with a widely used decomposition package.

Keywords: support vector machines, quadratic programs, decomposition techniques, gradient projection methods, large-scale problems.

1 Introduction

We consider the numerical solution of the convex quadratic programming (QP) problem arising in training *Support Vector Machines* (SVMs) [7, 33]:

$$\begin{array}{rl} \min & \mathcal{F}(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G \boldsymbol{x} - \sum_{i=1}^{n} x_i \\ \text{sub. to} & \sum_{i=1}^{n} y_i x_i = 0, \\ & 0 \le x_i \le C, \qquad i = 1, \ldots, n, \end{array} \tag{1}$$

where the size n is the number of labelled examples of the given training set

$$D = \{(\boldsymbol{z}_i, y_i),\ i = 1, \ldots, n, \quad \boldsymbol{z}_i \in \mathbb{R}^m,\ y_i \in \{-1, 1\}\},$$

*This work was supported by the Italian Education, University and Research Ministry via the FIRB Projects *"Statistical Learning: Theory, Algorithms and Applications"* (grant RBAU01877P) and *"Parallel Algorithms and Numerical Nonlinear Optimization"* (grant RBAU01JYPN).

and the entries of G are defined by

$$G_{ij} = y_i y_j K(\boldsymbol{z}_i, \boldsymbol{z}_j), \quad i, j = 1, 2, \ldots, n,$$

in which $K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a special kernel function. Examples of widely used kernel functions are the linear kernels ($K(\boldsymbol{z}_i, \boldsymbol{z}_j) = \boldsymbol{z}_i^T \boldsymbol{z}_j$), the polynomial kernels ($K(\boldsymbol{z}_i, \boldsymbol{z}_j) = (1 + \boldsymbol{z}_i^T \boldsymbol{z}_j)^d,\ d \in \mathbb{N}$) and the Gaussian kernels ($K(\boldsymbol{z}_i, \boldsymbol{z}_j) = \exp(-\|\boldsymbol{z}_i - \boldsymbol{z}_j\|_2^2/(2\sigma^2)),\ \sigma \in \mathbb{R}$). Since these kernel functions make the matrix G dense and in many real life applications the size of the training set is very large ($n \gg 10^4$), problem (1) cannot be generally solved by traditional approaches based on explicit storage of G.

This paper deals with the most popular strategies to overcome this drawback: the decomposition techniques. In short, these techniques split the problem into a sequence of smaller QP subproblems that can be stored in the available memory and efficiently solved [4, 23]. At each decomposition step, a subset of the variables is optimized through the solution of the subproblem in order to obtain a progress towards the minimum of $\mathcal{F}(\boldsymbol{x})$. One of the main differences among the various decomposition approaches proposed in literature is given by the size chosen for the subproblems. In fact, if the subproblems are sized 2, they can be solved analytically [5, 16, 26], while an inner numerical QP solver is necessary in decomposition schemes that use subproblems of larger size [6, 14, 15, 23, 31, 34]. Both the frameworks have been explored and have given rise to decomposition packages widely used within the SVM community.

Here we are interested to introduce some improvements to the decomposition technique recently proposed in [31]. This technique belongs to the class of decomposition schemes based on a numerical solution of the inner subproblems. It uses as inner QP solver the special gradient projection method introduced in [32] and for this reason it is called *Gradient Projection-based Decomposition Technique* (GPDT). The gradient projection method used by GPDT exhibits high performance on SVM-type QP problems and enables the decomposition technique to manage efficiently much larger subproblems in comparison to standard decomposition packages. This ability, combined with an appropriate selection of the variables to optimize at each step and a standard caching strategy for reducing kernel evaluations, allows GPDT to achieve promising performance with respect to the SVM^{light} algorithm, one of the most used and efficient decomposition packages. Furthermore, the possibility to decompose into large subproblems makes the scheme well suited for an easy and effective parallelization; the numerical experiments carried

out in [31] with a parallel version of GPDT give evidence of the importance of this approach in training SVMs on multiprocessor systems.

The improvements to GPDT introduced in this work concern the subproblems solution and a simple trick for a further reduction of the kernel evaluations. For the subproblems solution we consider the recent gradient projection method proposed in [9] and we show that it can be more efficient than the inner solver currently used by GPDT. The GPDT kernel evaluations are reduced by introducing an alternative formula for defining the data of the subproblem at each iteration. Numerical evidence of these improvements are given by solving well known benchmark problems and by comparison with the SVM^{light} package.

The paper is organized as follows: section 2 gives the decomposition technique, section 3 states and analyses the new inner solver, section 4 evaluates the effectiveness of the formula proposed for reducing the kernel evaluations, section 5 deals with the accuracy of the decomposition technique and, finally, section 6 reports the main conclusions.

2 A gradient projection-based decomposition technique

In this section we recall the special version of the decomposition technique proposed in [31] named GPDT2.

By following the classical notation, we split the indices of the variables $x_i,\ i = 1, \ldots, n$, into the set $\mathcal{B}$ of *basic* variables, called the *working set*, and the set $\mathcal{N} = \{1, 2, \ldots, n\} \setminus \mathcal{B}$ of *nonbasic* variables. Furthermore, we denote by $\boldsymbol{x}^*$ a solution of (1) and arrange $\boldsymbol{x}$ and G with respect to $\mathcal{B}$ and $\mathcal{N}$ as follows:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_{\mathcal{B}} \\ \boldsymbol{x}_{\mathcal{N}} \end{bmatrix}, \qquad G = \begin{bmatrix} G_{\mathcal{B}\mathcal{B}} & G_{\mathcal{B}\mathcal{N}} \\ G_{\mathcal{N}\mathcal{B}} & G_{\mathcal{N}\mathcal{N}} \end{bmatrix}.$$

The decomposition techniques for problem (1) are iterative procedures that, at each step, solve (1) with respect to the basic variables only. This leads to a sequence of QP subproblems simpler than the original problem since their size n_{sp} is equal to the size of the working set ($n_{sp} = \#\mathcal{B}$), that can be chosen much smaller than n. At each iteration, the subproblem solution is used to improve the current approximation of $\boldsymbol{x}^*$ and an appropriate updating of the working set is required in order to obtain the convergence.

The special decomposition technique given in [31] can be described as in Algorithm GPDT2.

ALGORITHM GPDT2 (Gradient Projection-based Decomposition Technique)

1. *Initialization.* Let $\boldsymbol{x}^{(1)}$ be a feasible point for (1), let n_{sp} and n_c be two integer values such that $n \geq n_{sp} \geq n_c > 0$, n_c even; let $\mathcal{L}$ be the largest even number such that $\mathcal{L} \leq \frac{n_{sp}}{10}$. Arbitrarily choose n_{sp} indices for the working set $\mathcal{B}$ and set $k \leftarrow 1$.

2. *QP subproblem solution.* Compute by a Gradient Projection Method the solution $\boldsymbol{x}_{\mathcal{B}}^{(k+1)}$ of

$$\begin{array}{rl} \min & \frac{1}{2}\boldsymbol{x}_{\mathcal{B}}^T G_{\mathcal{B}\mathcal{B}} \boldsymbol{x}_{\mathcal{B}} + \left(G_{\mathcal{B}\mathcal{N}} \boldsymbol{x}_{\mathcal{N}}^{(k)} - \mathbf{1}_{\mathcal{B}}\right)^T \boldsymbol{x}_{\mathcal{B}} \\ \text{sub. to} & \sum_{i\in\mathcal{B}} y_i x_i = -\sum_{i\in\mathcal{N}} y_i x_i^{(k)}, \\ & 0 \leq x_i \leq C \quad \forall i \in \mathcal{B}, \end{array} \tag{2}$$

   where $\mathbf{1}_{\mathcal{B}}$ is the n_{sp}–vector of all ones; set $\boldsymbol{x}^{(k+1)} = \left(\boldsymbol{x}_{\mathcal{B}}^{(k+1)^T}, \ \boldsymbol{x}_{\mathcal{N}}^{(k)^T} \right)^T$.

3. *Gradient updating.* Update the gradient

$$\nabla\mathcal{F}(\boldsymbol{x}^{(k+1)}) = \nabla\mathcal{F}(\boldsymbol{x}^{(k)}) + \left[\begin{array}{c} G_{\mathcal{B}\mathcal{B}} \\ G_{\mathcal{N}\mathcal{B}} \end{array} \right] \left(\boldsymbol{x}_{\mathcal{B}}^{(k+1)} - \boldsymbol{x}_{\mathcal{B}}^{(k)}\right) \tag{3}$$

   and terminate if $\boldsymbol{x}^{(k+1)}$ satisfies the KKT conditions.

4. *Updating of $\mathcal{B}$.*

   4.1. Find the indices corresponding to the nonzero components of the solution of

$$\begin{array}{rl} \min & \nabla\mathcal{F}(\boldsymbol{x}^{(k+1)})^T \boldsymbol{d} \\ \text{sub. to} & \boldsymbol{y}^T \boldsymbol{d} = 0, \\ & d_i \geq 0 \quad \text{for } i \text{ such that } \alpha_i = 0, \\ & d_i \leq 0 \quad \text{for } i \text{ such that } \alpha_i = C, \\ & -1 \leq d_i \leq 1, \\ & \#\{d_i \mid d_i \neq 0\} \leq n_c. \end{array} \tag{4}$$

   Let $\bar{\mathcal{B}}$ the set of these indices.

   4.2. Fill $\bar{\mathcal{B}}$ up to n_{sp} entries by adding the most recent indices $j \in \mathcal{B}$ satisfying $0 < x_j^{(k+1)} < C$; if these indices are not enough, then add the most recent indices $j \in \mathcal{B}$ such that $x_j^{(k+1)} = 0$ and, eventually, the most recent indices $j \in \mathcal{B}$ satisfying $x_j^{(k+1)} = C$.

   4.3. Set $n_c = \min\{n_c, \max\{10, \mathcal{L}, n_{new}\}\}$, where n_{new} is the largest even number such that $n_{new} \leq \#\{j,\ j \in \bar{\mathcal{B}},\ j \notin \mathcal{B}\}$; set $\mathcal{B} = \bar{\mathcal{B}}$, $k \leftarrow k+1$ and go to step 2.

The GPDT2 scheme is similar to the well known SVM^{light} algorithm but presents an essential difference in the choice of the inner QP solver. In fact, GPDT2 uses a gradient projection method, the *Generalized Variable Projection Method* (GVPM) proposed in [32], that is much more effective than the inner solvers (the pr_LOQO or the Hildreth and D'Esopo method) on

which SVM^{light} is based. Thus GPDT2 is able to efficiently solve large subproblems that are hardly managed by SVM^{light} due to the inner optimizer overhead. As a consequence, while SVM^{light} has been designed and very well optimized for decomposition processes in which n_{sp} is very small (generally less than 10^2), GPDT2 is appropriately developed for decomposing into large subproblems (generally $n_{sp} > 10^3$) in order to exploit the high performance of the inner solver. The new GPDT2 approach introduce some implementative difficulties but opens interesting scenarios. On one hand, since the entries of $G_{\mathcal{BB}}$ and $G_{\mathcal{NB}}$ are not in memory, large values of n_{sp} increase the request of kernel evaluations per iteration. This means that special strategies for computing the kernels (exploitation of sparseness in training examples) and for reducing their evaluations (caching of G entries and sparseness exploitation in the matrix-vector product (3)) become crucial tricks for the GPDT2 performance. On the other hand, the ability to work with large n_{sp} allows to explore the benefits, in terms of convergence rate, arising from the possibility to optimize many variables at each decomposition iteration. To this end, an effective rule for updating the working set $\mathcal{B}$ is the essential key. Unfortunately, for the case of large sized working sets, this topic is not widely investigated in literature since the most popular decomposition approaches are commonly used with very small n_{sp} or they are designed for $n_{sp} = 2$ only, in order to solve the subproblems analytically. The working set selection described in step 4 of Algorithm GPDT2 has been recently introduced in [31]; it follows the classical SVM^{light} selection rule but introduces new devices useful in case of large working sets. By proceeding as in SVM^{light}, step 4.1 finds at most $n_c \leq n_{sp}$ indices for the new working set by solving the linear problem (4); the aim is to define basic variables that make possible a rapid decrease of the objective function in the new iteration. This selection idea is exploited by many decomposition approaches and is at the basis of the main theoretical studies on the convergence properties of decomposition techniques [17, 19, 20, 21, 24]. Since the parameter n_c is usually recommended to be less than n_{sp} in order to reduce zigzagging phenomena, $(n_{sp} - n_c)$ indices are required to fill up $\mathcal{B}$. The filling criterion used in step 4.2 is similar to that introduced in [34] but it takes into account also how long a variable is in the working set (see also [19] for a discussion about the importance to retain free variables of the previous $\mathcal{B}$). The last step 4.3 introduces an adaptive reduction of the parameter n_c. For large n_{sp}, this trick allows the decomposition procedure to start with large n_c (e.g. $n_c = n_{sp}/2$), so many new variables can be optimized in the first iterations, but avoids zigzagging through the progressive reduction of n_c. More details

on the above working set selection can be found in [31] where its effectiveness is evaluated by an extensive computational study on well known benchmark problems. In [31] a comparison between GPDT2 and SVM^{light} is also performed. The two solvers exhibit opposite behaviour: GPDT2 gets its best performance for large n_{sp} and shows low sensitivity to n_{sp} variations while SVM^{light} is competitive only for very small n_{sp}; if the best performances are compared, GPDT2 is preferable. Finally, one of the most important aspects of GPDT2 needs to be recalled: its easy parallelization. By decomposing into large subproblems, GPDT2 generally requires very few decomposition iterations in which the most expensive tasks are the solution of the subproblem (2) and the kernel evaluations required in step 3. These tasks can be easily performed on a multiprocessor system by using a parallel version of the GVPM and by distributing the kernel evaluations among the available processors. A parallel version of GPDT2, derived by following these ideas and the implementative framework introduced in [34], has been tested in [31] showing promising speedups respect to the serial GPDT2 and the SVM^{light}.

3 An improved inner QP solver for GPDT2

The inner QP subproblems (2) have the following general form:

$$\min_{\boldsymbol{w}\in\Omega} \quad f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T A\boldsymbol{w} + \boldsymbol{b}^T\boldsymbol{w} \tag{5}$$

where $A \in \mathbb{R}^{n_{sp}\times n_{sp}}$ is symmetric and positive semidefinite, $\boldsymbol{w}, \boldsymbol{b} \in \mathbb{R}^{n_{sp}}$ and the feasible region Ω is defined by

$$\Omega = \{\boldsymbol{w} \in \mathbb{R}^{n_{sp}}, \quad 0 \leq \boldsymbol{w} \leq C, \quad \boldsymbol{c}^T\boldsymbol{w} = d\}. \tag{6}$$

We recall that now the size n_{sp} allows the matrix A to be stored in memory.

The special gradient projection method used by GPDT2 for solving these subproblems is reported in Algorithm GVPM. Gradient projection methods are appealing approaches for problems (5) since they consist in a sequence of projections onto the feasible region, that are non-expensive due to the special constraints (6). In fact, a projection onto Ω can be performed by efficient $O(n_{sp})$ algorithms like those in [9] and [25].

GVPM is characterized by the use of a standard limited minimization rule as linesearch technique [2] and a special selection strategy for the steplength parameter. The steplength selection consists in an adaptive alternation of the two well known Barzilai-Borwein (BB) rules [1]

ALGORITHM GVPM (Generalized Variable Projection Method)

1. *Initialization.* Let $\boldsymbol{w}^{(0)} \in \Omega$, $i_\alpha \in \{1,2\}$, $0 < \alpha_{\min} < \alpha_{\max}$, $\alpha_0 \in [\alpha_{\min}, \alpha_{\max}]$, $n_{\min}, n_{\max} \in \mathbb{N}$, $0 < n_{\min} \leq n_{\max}$, $\lambda_\ell \leq 1 \leq \lambda_u$; set $n_\alpha = 1$, $k = 0$.

2. *Projection.* Terminate if $\boldsymbol{w}^{(k)}$ satisfies a stopping criterion; otherwise compute the descent direction
$$\boldsymbol{d}^{(k)} = P_\Omega(\boldsymbol{w}^{(k)} - \alpha_k(A\boldsymbol{w}^{(k)} + \boldsymbol{b})) - \boldsymbol{w}^{(k)}.$$

3. *Linesearch.* Compute
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \lambda_k \boldsymbol{d}^{(k)}, \qquad \text{with} \qquad \lambda_k = \arg\min_{\lambda \in [0,1]} f(\boldsymbol{w}^{(k)} + \lambda \boldsymbol{d}^{(k)}).$$

4. *Update.* If * ${\boldsymbol{d}^{(k)}}^T A \boldsymbol{d}^{(k)} \leq 0$ then
   set $\alpha_{k+1} = \alpha_{\max}$;
   else
   compute $\alpha_{k+1}^{(1)} = \dfrac{{\boldsymbol{d}^{(k)}}^T \boldsymbol{d}^{(k)}}{{\boldsymbol{d}^{(k)}}^T A \boldsymbol{d}^{(k)}}$, $\alpha_{k+1}^{(2)} = \dfrac{{\boldsymbol{d}^{(k)}}^T A \boldsymbol{d}^{(k)}}{{\boldsymbol{d}^{(k)}}^T A^2 \boldsymbol{d}^{(k)}}$, $\lambda_{\text{opt}} = \arg\min_{\lambda} f(\boldsymbol{w}^{(k)} + \lambda \boldsymbol{d}^{(k)})$.
   If $(n_\alpha \geq n_{\min})$ and $\Big((n_\alpha \geq n_{\max})$ or $(\alpha_{k+1}^{(2)} \leq \alpha_k \leq \alpha_{k+1}^{(1)})$
   or $\Big((\lambda_{\text{opt}} < \lambda_\ell$ and $\alpha_k = \alpha_k^{(1)})$ or $(\lambda_{\text{opt}} > \lambda_u$ and $\alpha_k = \alpha_k^{(2)}) \Big) \Big)$ then
   set $i_\alpha \leftarrow \text{mod}(i_\alpha, 2) + 1$, $n_\alpha = 0$;
   end.
   Compute $\alpha_{k+1} = \min\left\{\alpha_{\max}, \max\left\{\alpha_{\min}, \alpha_{k+1}^{(i_\alpha)}\right\}\right\}$;
   end.
   Set $k \leftarrow k+1$, $n_\alpha \leftarrow n_\alpha + 1$ and go to step 2.

*Since A is positive semidefinite, the condition may reduce to an equality test.

$\left(\alpha_{k+1}^{(1)} = \frac{{\boldsymbol{d}^{(k)}}^T \boldsymbol{d}^{(k)}}{{\boldsymbol{d}^{(k)}}^T A \boldsymbol{d}^{(k)}}, \quad \alpha_{k+1}^{(2)} = \frac{{\boldsymbol{d}^{(k)}}^T A \boldsymbol{d}^{(k)}}{{\boldsymbol{d}^{(k)}}^T A^2 \boldsymbol{d}^{(k)}}\right)$; this alternation avoids the typical slow convergence exhibited by gradient projection methods with monotone linesearches. Furthermore, this selection rule doesn't introduce expensive operations since it involves the matrix-vector product $A\boldsymbol{d}^{(k)}$ already required for computing λ_k in step 3. The stopping criterion used in GVPM is based on the fulfilment within a prefixed tolerance of the KKT conditions, with the equality constraint multiplier computed as suggested in [15]. In this way, the main task of each iteration remains the computation of $A\boldsymbol{d}^{(k)}$ (remember that $\boldsymbol{t} = A\boldsymbol{w}^{(k)}$ can be updated by $\boldsymbol{t} \leftarrow \boldsymbol{t} + \lambda_k A\boldsymbol{d}^{(k)} = A\boldsymbol{w}^{(k+1)}$), whose cost can be substantially reduced by exploiting the sparsity of $\boldsymbol{d}^{(k)}$. A convergence analysis of GVPM is presented in [32] where, for convex QP problems,

the R-linear convergence is derived by proceeding as in [29] for standard variable projection methods. The analysis developed in [32] has shown that GVPM is very effective on the SVM QP subproblems, outperforming QP solvers widely used in this application (e.g. pr_LOQO and MINOS) as well as other gradient projection methods [3, 28, 29].

ALGORITHM ALG. 2

---

1. *Initialization.* Let $\boldsymbol{w}^{(0)} \in \Omega$, $0 < \alpha_{\min} < \alpha_{\max}$, $\alpha_0 \in [\alpha_{\min}, \alpha_{\max}]$, $L \in \mathbb{N}$;
   set $f_{ref} = \infty$, $f_{best} = f_c = f(\boldsymbol{w}^{(0)})$, $l = 0$, $k = 0$, $\boldsymbol{s}^{(k-1)} = \boldsymbol{y}^{(k-1)} = \boldsymbol{0}$.

2. *Projection.* Terminate if $\boldsymbol{w}^{(k)}$ satisfies a stopping criterion; otherwise compute the descent direction
$$\boldsymbol{d}^{(k)} = P_\Omega(\boldsymbol{w}^{(k)} - \alpha_k(A\boldsymbol{w}^{(k)} + \boldsymbol{b})) - \boldsymbol{w}^{(k)}.$$

3. *Linesearch.*
   If $\big(k = 0 \text{ and } f(\boldsymbol{w}^{(k)} + \boldsymbol{d}^{(k)}) \geq f(\boldsymbol{w}^{(k)})\big)$ or $\big(k > 0 \text{ and } f(\boldsymbol{w}^{(k)} + \boldsymbol{d}^{(k)}) \geq f_{ref}\big)$ then
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \lambda_k \boldsymbol{d}^{(k)}, \qquad \text{with} \qquad \lambda_k = \underset{\lambda \in [0,1]}{\arg\min}\, f(\boldsymbol{w}^{(k)} + \lambda \boldsymbol{d}^{(k)});$$
   else
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \boldsymbol{d}^{(k)}.$$

4. *Update.* Compute $\boldsymbol{s}^{(k)} = \boldsymbol{w}^{(k+1)} - \boldsymbol{w}^{(k)}$; $\boldsymbol{y}^{(k)} = A(\boldsymbol{w}^{(k+1)} - \boldsymbol{w}^{(k)})$.
   If ${\boldsymbol{s}^{(k)}}^T \boldsymbol{y}^{(k)} \leq 0$ then
   set $\alpha_{k+1} = \alpha_{\max}$;
   else
   If ${\boldsymbol{s}^{(k-1)}}^T \boldsymbol{y}^{(k-1)} \leq 0$ then
   set $\alpha_{k+1} = \min\left\{\alpha_{\max}, \max\left\{\alpha_{\min}, \frac{{\boldsymbol{s}^{(k)}}^T \boldsymbol{s}^{(k)}}{{\boldsymbol{s}^{(k)}}^T \boldsymbol{y}^{(k)}}\right\}\right\}$;
   else
   set $\alpha_{k+1} = \min\left\{\alpha_{\max}, \max\left\{\alpha_{\min}, \frac{{\boldsymbol{s}^{(k)}}^T \boldsymbol{s}^{(k)} + {\boldsymbol{s}^{(k-1)}}^T \boldsymbol{s}^{(k-1)}}{{\boldsymbol{s}^{(k)}}^T \boldsymbol{y}^{(k)} + {\boldsymbol{s}^{(k-1)}}^T \boldsymbol{y}^{(k-1)}}\right\}\right\}$;
   end.
   end.
   If $f(\boldsymbol{w}^{(k+1)}) < f_{best}$ then
   set $f_{best} = f(\boldsymbol{w}^{(k+1)})$, $f_c = f(\boldsymbol{w}^{(k+1)})$, $l = 0$;
   else
   set $f_c = \max\left\{f_c, f(\boldsymbol{w}^{(k+1)})\right\}$, $l = l + 1$;
   If $l = L$ then
   set $f_{ref} = f_c$, $f_c = f(\boldsymbol{w}^{(k+1)})$, $l = 0$;
   end.
   end.
   Set $k \leftarrow k + 1$, and go to step 2.

Recently, Dai and Fletcher [9] have proposed a new gradient projection method for singly linearly constrained QP problems subject to lower and upper bounds. In the computational experiments reported in [9], this method has exhibited promising behaviour and, in particular, has shown better results in comparison to GVPM on some medium-scale SVM test problems. Thus, the Dai and Fletcher method can be a valid alternative to GVPM as the inner solver within GPDT2. Here we are interested to evaluate the GPDT2 performance improvements due to this new inner solver. We recall the Dai and Fletcher method in Algorithm Alg. 2. The method can be described within a scheme similar to GVPM, and the same considerations on the computational cost per iteration still hold. Nevertheless, the linesearch step and the steplength selection rule are very different.

Alg. 2 uses an adaptive nonmonotone linesearch in order to allow the objective function value $f(\boldsymbol{w}^{(k)})$ to increase on some iterations. This is an important strategy for exploiting the benefits in terms of convergence rate given by BB-like steplengths, that typically imply nonmonotone behaviour [10]. The special linesearch strategy used in Alg. 2 has been recently suggested in [8] and its main feature is the adaptive updating of the reference function value f_{ref}. The purpose of the updating rule is to cut down the number of times the linesearch is brought into play and, consequently, to frequently accept the iterate $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \boldsymbol{d}^{(k)}$ obtained through an appropriate steplength α_k. The numerical experience in [8, 9] shows that this technique can perform better than the classical nonmonotone linesearch [12] used by Birgin-Martìnez-Raydan in their gradient projection methods [3].

For the steplength updating in Alg. 2, Dai and Fletcher propose the general rule

$$\alpha_{k+1} = \frac{\sum_{i=0}^{m-1} \boldsymbol{s}^{(k-i)^T} \boldsymbol{s}^{(k-i)}}{\sum_{i=0}^{m-1} \boldsymbol{s}^{(k-i)^T} \boldsymbol{y}^{(k-i)}}, \qquad m \geq 1,$$

and suggest the choice $m = 2$ as the better for the SVM QP problems (observe that the case corresponding to $m = 1$ reduces to the standard BB rule $\alpha_{k+1}^{(1)}$). In particular, in [9] the above steplength is discussed in comparison with a selection rule based on simple alternations of the two standard BB rules ($\alpha_{k+1}^{(1)}$ and $\alpha_{k+1}^{(2)}$). While in different contexts the idea to combine the alternation technique of the two steplengths and a nonmonotone linesearch has given promising results [8, 13, 32], this approach seems useless on the SVM QP problems. On the other hand, a sophisticated alternation technique of the two BB rules is crucial for the effectiveness of a monotone scheme like the GVPM. All these considerations suggest that further study on

the relationship between the steplength selection and the linesearch technique is required for improving existing gradient projection schemes. Nevertheless, this interesting topic is beyond the aim of this work and in the sequel we will concentrate only on the behaviour of GVPM and Alg. 2 within GPDT2. We conclude the introduction to Alg. 2 by recalling that its global convergence in real arithmetic can be proved through a simple argument presented in [8].

In order to analyze the behaviour of the two solvers within GPDT2 we consider three large test problems of the form (1) derived by training Gaussian SVMs on the well known UCI Adult data set [22], WEB data set [26] and MNIST data set [18]. A detailed description of the test problems generation is reported in the Appendix. All the experiments are carried out with standard C code running on a single processor of an IBM SP4 equipped with Power4 1.3GHz CPUs.

We compare GVPM and Alg. 2 on the subproblems (2) that arise when GPDT2 is applied to the above test problems with starting point $\boldsymbol{x}^{(1)} = \mathbf{0}$ and tolerance 0.001 for the stopping rule. The parameters n_{sp} (the subproblems size) and n_c differ for each test problem and will be reported in the sequel.

GVPM and Alg. 2 solve each GPDT2 subproblem with the same $\boldsymbol{w}^{(0)}$ ($\boldsymbol{w}^{(0)} = P_\Omega(\boldsymbol{x}_{\mathcal{B}}^{(k)})$ when the GPDT2 stopping rule is nearly satisfied or $\boldsymbol{w}^{(0)} = P_\Omega(\mathbf{0})$ otherwise) and stopping rule (KKT fulfilment within 0.001). For the other parameters the following setting is used: $\alpha_{\min} = 10^{-30}$, $\alpha_{\max} = 10^{30}$, $\alpha_0 = \|P_\Omega(\boldsymbol{w}^{(0)} - (A\boldsymbol{w}^{(0)} + \boldsymbol{b})) - \boldsymbol{w}^{(0)}\|_\infty^{-1}$; moreover, $i_\alpha = 2$, $n_{\min} = 3$, $n_{\max} = 10$, $\lambda_\ell = 0.5$, $\lambda_u = 5$, in GVPM and $L = 2$ in Alg. 2. The parameters setting used in GVPM is derived by the wide experimentation developed in [32]; for Alg. 2, the results obtained with different values of L between 1 and 10 suggest that $L = 2$ is preferable on these problems. We have also tested the version Alg. 2^* proposed in [9], for different values of its parameter M, but we have obtained worse results in comparison with Alg. 2 with $L = 2$. In these experiments, both GVPM and Alg. 2 compute the projection onto Ω by the algorithm described in [25]. In future versions of the solvers, we will compute this projection with the more efficient Alg. 1 given in [9]; however, remember that the cost of the projection is not relevant with respect to the cost of the matrix-vector multiplication $A\boldsymbol{d}^{(k)}$ and the use of Alg. 1 will not significantly change the subsequent numerical results.

In Table 1 the behaviour of the two solvers is shown in terms of total number of inner iterations (total it.) and total optimization time in seconds (total time) needed to solve the subproblems. For each test problem, we report also the values used for the parameters n_{sp}, n_c and the

corresponding GPDT2 decomposition iterations (decomp. it.). From Table 1 we can observe that, in each test problem, Alg. 2 shows a reduced number of inner iterations; this reduction is more evident in case of Web and MNIST data sets where it implies a significant time saving. The behaviour of the two solvers is better emphasized by Figure 1-3 where the inner iterations required for each decomposition step are plotted. Finally, for the MNIST problem only, in Table 2 the number of step reduction ($\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \lambda_k \boldsymbol{d}^{(k)}$ with $\lambda_k < 1$) produced by the linesearch techniques is reported in the column denoted by "ls.". The two linesearches behave in a similar way: on average the ratio between it. and ls. is 5.1 for GVPM and 6.5 for Alg. 2. However, the linesearches exhibit this similar behaviour by working in combination with different steplength selection rules, that imply remarkable differences in the convergence rate. In particular, we can conclude that the adaptive nonmonotone linesearch and the steplength selection used by Alg. 2. combine successfully on these test problems.

Table 1: Total inner iterations and total optimization time given by GVPM and Alg. 2.

				GVPM		Alg. 2	
Test problems	n_{sp}	n_c	decomp. it.	total it.	total time	total it.	total time
UCI Adult $n = 32561$	1300	650	30	8482	20.3	6820	17.9
Web $n = 49749$	1500	750	24	17844	84.9	12587	61.2
MNIST $n = 60000$	2000	300	22	10228	82.9	6623	52.0

4 Reducing kernel evaluations in GPDT2

As observed in section 2, the main tasks of each GPDT2 iteration are to solve the QP subproblem and to compute the kernels for updating the gradient (remember that we are considering the case in which G is not in memory). Thus, the decomposition scheme can benefit not only from an effective inner QP solver but also by optimizing the kernel computation and by reducing the number of kernel evaluations.

In order to optimize the kernel computation, GPDT2 uses sparse representation of the training examples and exploits their sparseness during the computation, as commonly done by other decomposition approaches [15, 27].

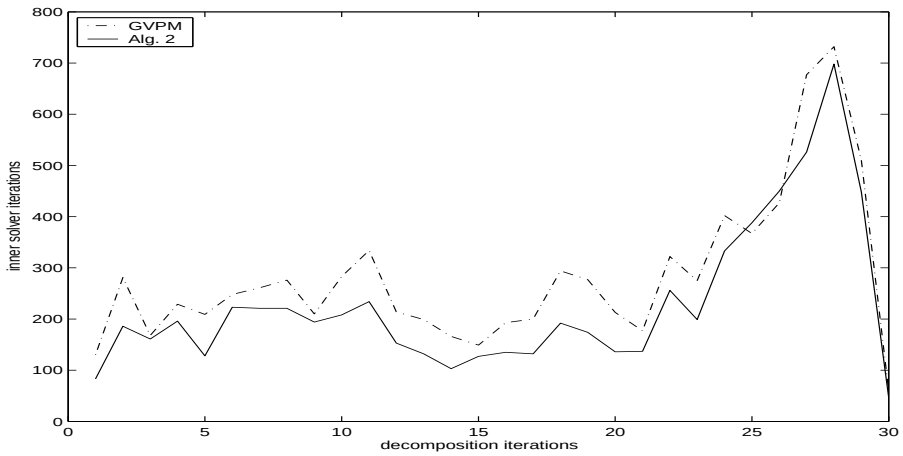

Figure 1: Inner solvers behaviour: UCI Adult test problems ($n = 32561,\ n_{sp} = 1300,\ n_c = 650$).

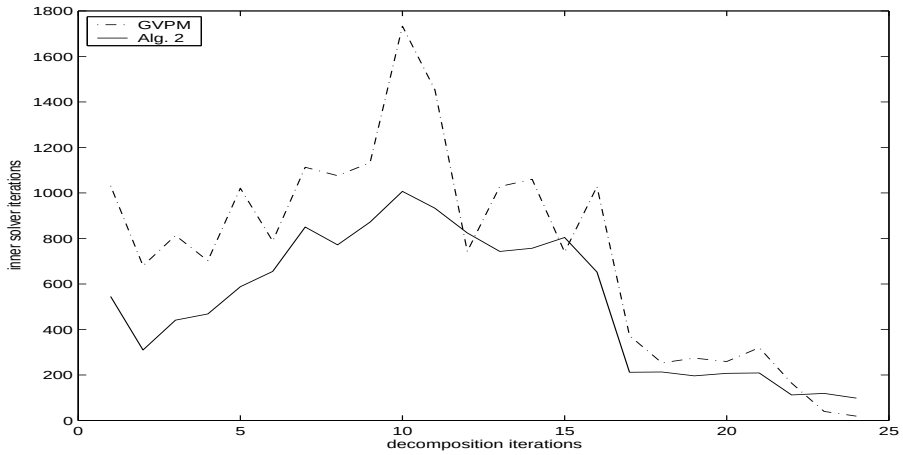

Figure 2: Inner solvers behaviour: WEB test problems ($n = 49749,\ n_{sp} = 1500,\ n_c = 750$).

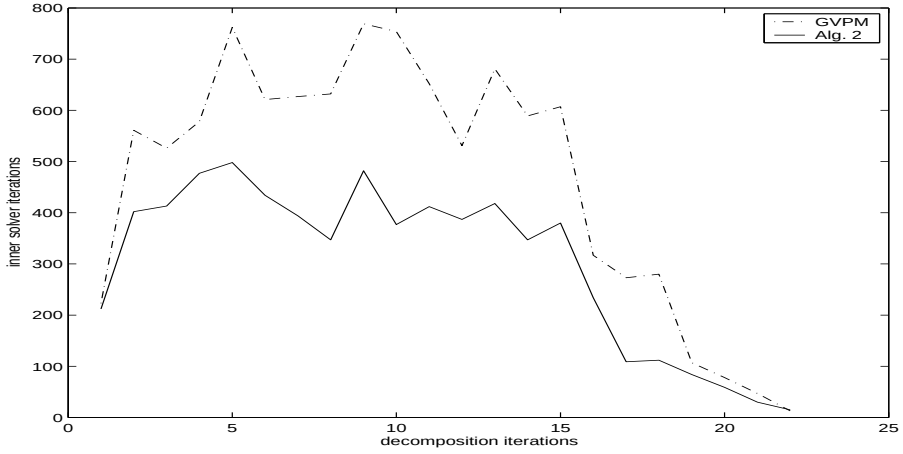

Figure 3: Inner solvers behaviour: MNIST test problems ($n = 60000,\ n_{sp} = 2000,\ n_c = 300$).

Table 2: Inner solvers behaviour on the MNIST test problem ($n = 60000,\ n_{sp} = 2000,\ n_c = 300$).

decomposition	GVPM			Alg. 2		
iterations	it.	ls.	time	it.	ls.	time
1	223	41	0.6	**212**	23	0.7
2	561	93	2.7	**402**	54	1.9
3	526	105	2.6	**413**	62	2.3
4	578	111	3.5	**477**	71	2.9
5	762	156	4.8	**498**	73	3.3
6	621	129	4.8	**434**	68	3.3
7	627	123	5.5	**394**	62	3.2
8	632	118	5.1	**347**	49	2.9
9	769	151	6.4	**482**	81	4.0
10	754	147	7.0	**377**	53	3.2
11	652	132	6.2	**412**	68	4.0
12	531	103	4.6	**387**	54	3.7
13	681	136	6.9	**418**	68	4.1
14	589	124	5.3	**347**	54	3.0
15	607	124	6.1	**380**	68	3.3
16	317	63	3.1	**234**	40	2.2
17	273	56	2.6	**109**	21	1.1
18	280	48	2.7	**112**	17	1.0
19	107	24	0.9	**84**	13	0.8
20	78	14	0.8	**59**	11	0.6
21	47	4	0.5	**30**	3	0.3
22	**13**	3	0.2	15	4	0.2

Concerning the reduction of kernel evaluations, the crucial trick consists in using an area of the available memory as a caching area, where some elements of G can be stored in order to avoid their recomputation in the subsequent iterations. The caching strategy used in GPDT2 is based on the simple idea to fill the caching area with the columns of G involved in the updating formula (3), that is the columns of G corresponding to the indices of $\mathcal{B}$ for which $(x_i^{(k+1)} - x_i^{(k)}) \neq 0,\ i \in \mathcal{B}$. When the cache is full, the current columns substitute those that have not been used for the greatest number of iterations. This strategy, combined with the working set updating used in step 4 which forces some indices of $\mathcal{B}$ to remain in the new working set, seems enough to avoid an undesirable increase in the number of kernel evaluations.

The entries of G stored in the caching area are exploited by GPDT2 not only for the gradient

updating in step 3 but also for the computation of $G_{\mathcal{BB}}$ and $G_{\mathcal{BN}}\boldsymbol{x}_{\mathcal{N}}^{(k)}$ in step 2.

Here we are interested to show that a simple alternative way to derive the coefficients of the linear terms in (2) can reduce, sometimes significantly, the number of kernel evaluations. In fact, taking into account that

$$G_{\mathcal{BN}}\boldsymbol{x}_{\mathcal{N}}^{(k)} - \mathbf{1}_B = \nabla\mathcal{F}_{\mathcal{B}}(\boldsymbol{x}^{(k)}) - G_{\mathcal{BB}}\boldsymbol{x}_{\mathcal{B}}^{(k)}, \tag{7}$$

with $\nabla\mathcal{F}_{\mathcal{B}}(\boldsymbol{x}^{(k)})$ coming from the gradient computed in the previous iteration and $G_{\mathcal{BB}}$ already computed in the current step 2, we can obtain the required coefficients without additional kernel evaluations (the same trick is used also in [5]).

The GPDT2 improvements due to the kernel evaluations reduction obtained in this way are emphasized by the following experiments. We solve the UCI Adult, the Web and the MNIST test problems on the previously used testing platform by the package SVM^{light} (version 5.0), the method GPDT2 described in [31] and the modified version of GPDT2 in which $G_{\mathcal{BN}}\boldsymbol{x}_{\mathcal{N}}^{(k)} - \mathbf{1}_B$ is computed as in (7) and the subproblems (2) are solved by the Alg. 2 discussed in section 3. This new GPDT2 version is called *Improved Gradient Projection-based Decomposition Technique* (IGPDT). The three algorithms are run with a caching area of 512MB and a tolerance of 0.001 on the KKT-based stopping rule. The SVM^{light} is used with the pr_LOQO inner QP solver and default setting for other parameters; this means that the *shrinking* strategy for reducing the problem size and, consequently, the number of kernel evaluations, is enabled [15]. Many experiments with different values for the parameters n_{sp} and n_c are carried out for each method. Table 3 shows the results corresponding to some meaningful values of these parameters; in particular the values for which the methods have given the best computational time are reported. A more detailed analysis of the SVM^{light} and GPDT2 performance on these test problems is available in [31]. In Table 3 the behaviour of the three decomposition techniques is shown in terms of the number of decomposition iterations, the computational time in seconds (without including the time for data reading) and the number of kernel evaluations in millions; the symbol "*" means that the 2000 seconds time limit is reached, $\mathrm{N_{SV}}$ is the number of support vectors, i.e. the training examples corresponding to the nonzero components of $\boldsymbol{x}^*$ and $\mathrm{N_{BSV}}$ denotes the number of bound support vectors, i.e. the training examples corresponding to the components of $\boldsymbol{x}^*$ at the upper bound.

We may observe that, for given n_{sp} and n_c, IGPDT always outperforms significantly GPDT2,

Table 3: SVM^{light} , GPDT2 and IGPDT for different n_{sp} and n_c.

		SVM^{light}			GPDT2			IGPDT		
n_{sp}	n_c	it.	time	kernels	it.	time	kernels	it.	time	kernels
UCI Adult data set ($n = 32561$, $\text{N}_{\text{SV}} = 11698$, $\text{N}_{\text{BSV}} = 10605$)										
20	10	4407	278.7	415.1	3897	213.2	560.7	3739	175.7	460.4
600	300	125	505.2	444.9	90	202.5	649.8	81	**143.0**	493.0
1300	650	48	1703.4	468.2	32	215.9	656.3	30	**158.5**	497.2
Web data set ($n = 49749$, $\text{N}_{\text{SV}} = 3199$, $\text{N}_{\text{BSV}} = 919$)										
8	4	7235	184.6	188.9	9094	230.7	225.3	8601	217.0	212.9
600	300	137	580.2	432.9	190	207.6	498.8	204	**175.8**	416.2
1500	750	40	1932.4	512.8	25	168.1	258.4	24	**142.8**	251.6
MNIST data set ($n = 60000$, $\text{N}_{\text{SV}} = 3159$, $\text{N}_{\text{BSV}} = 160$)										
8	4	9226	901.6	252.8	10649	1045.5	731.7	10110	966.3	669.0
200	100	320	1723.1	475.3	491	952.6	747.1	542	910.6	714.2
1000	200		$*$		55	778.7	483.1	55	**649.2**	474.5
2000	300		$*$		22	662.8	404.4	22	**562.4**	403.9
3700	1000		$*$		8	704.1	443.1	8	**630.6**	440.8

confirming the effectiveness of the two improvements introduced in this work. In particular, by considering the columns concerning the kernel evaluations in Table 3 and the total time for optimization reported in Table 1, it is possible to evaluate the impact of the two improvements in the different test problems. If the best performance only is considered, the new IGPDT allows remarkable time reduction in comparison with SVM^{light} , especially on the UCI Adult and MNIST data sets. Furthermore, it is interesting to observe the lower number of kernel evaluations exhibited by SVM^{light} when small values of n_{sp} are used; this is due to the default shrinking strategy that is very effective in these situations (for example, in case of the MNIST test problems with $n_{sp} = 8$ and $n_c = 4$, SVM^{light} needs 559.1×10^6 kernel evaluations without shrinking). On the other hand, IGPDT doesn't adopt shrinking and further improvements can be expected by introducing this strategy. Nevertheless, when IGPDT works with sufficiently large n_{sp}, its caching strategy and kernel optimization avoid unsatisfactory overhead for kernel management and allow to successfully exploit the high performance of the inner QP solver.

Finally, we remark that, as for previous gradient projection-based decomposition approaches [31, 34], the IGPDT is very well suited to be efficiently implemented in parallel. We will

deal with a parallel version of IGPDT in a future work dedicated to the training of SVMs on multiprocessor systems.

5 On the accuracy of IGPDT

We now briefly discuss on the accuracy of the approximate solution given by IGPDT. The IGPDT algorithm, like the SVM^{light} technique, is implemented in double precision, but it keeps the G_{ij} entries in single precision for memory saving. All the previous numerical experiments are performed by stopping the decomposition technique when the KKT conditions of (1) are satisfied within a tolerance of 10^{-3}. This stopping rule is the same used as the default in SVM^{light} and it is generally considered sufficient to ensure a good performance of the SVM methodology.
In order to study the IGPDT accuracy, we consider a small strictly convex test problem sized $n = 800$ generated by training a Gaussian SVM (with the parameters described in the Appendix) on a subset of the MNIST handwritten digit database, built with the first $n/2$ inputs of the digit "8" and the first $n/2$ inputs of the other digits. A very accurate solution of this small problem can be obtained by using the QL Fortran routine of Schittkowski [30], that implements, in double precision, the primal-dual active set method of Goldfarb and Idnani [11]. Denoting by $\boldsymbol{x^*}$ the QL solution and by $\bar{\boldsymbol{x}}$ the solution given by a decomposition technique, we analyse the accuracy of IGPDT and SVM^{light} by evaluating the errors $Er_{\bar{\boldsymbol{x}}} = \frac{\|\bar{\boldsymbol{x}}-\boldsymbol{x^*}\|_2}{\|\boldsymbol{x^*}\|_2}$ and $Er_{\mathcal{F}} = \frac{|\mathcal{F}(\bar{\boldsymbol{x}})-\mathcal{F}(\boldsymbol{x}^*)|}{|\mathcal{F}(\boldsymbol{x}^*)|}$.

Table 4: IGPDT and SVM^{light} accuracy on an MNIST test problem sized $n = 800$.

stopping rule	SVM^{light}		($n_{sp} = 80$, $n_c = 40$)			IGPDT		($n_{sp} = 160$, $n_c = 80$)		
tolerance	N_{SV}	N_{BSV}	$Er_{\bar{\boldsymbol{x}}}$	$Er_{\mathcal{F}}$	N_{er}	N_{SV}	N_{BSV}	$Er_{\bar{\boldsymbol{x}}}$	$Er_{\mathcal{F}}$	N_{er}
10^{-1}	286	1	0.6 e− 1	0.2 e− 03	2540	277	1	0.2 e− 1	0.2 e− 03	2554
10^{-2}	280	1	0.5 e− 2	0.1 e− 04	2510	281	1	0.4 e− 2	0.6 e− 05	2512
10^{-3}	281	1	0.5 e− 3	0.1 e− 06	2510	281	1	0.6 e− 3	0.9 e− 07	2509
10^{-4}	281	1	0.6 e− 4	0.2 e− 08	2508	281	1	0.7 e− 4	0.1 e− 08	2510
10^{-5}	281	1	0.6 e− 5	0.1 e− 10	2509	281	1	0.5 e− 5	0.7 e− 11	2509
10^{-6}	281	1	0.7 e− 6	0.1 e− 12	2509	281	1	0.6 e− 6	0.1 e− 12	2509

In Table 4 the values of $Er_{\bar{\boldsymbol{x}}}$ and $Er_{\mathcal{F}}$ corresponding to different tolerances in the KKT residual are reported for both SVM^{light} and IGPDT. Furthermore, in order to show how, in this particular application, the training algorithm accuracy affects the SVM performance, we evaluate the SVM classifiers trained with different accuracy on a test set of 39200 MNIST examples not included

in the training set. The column N_{er} reports the number of classification errors. The results corresponding to the QL solution $\boldsymbol{x}^*$ are: $\mathrm{N_{SV}} = 281$, $\mathrm{N_{BSV}} = 1$ and $N_{er} = 2509$. The QL computational time is about five times the one required by the decomposition techniques.

These experiments show that IGPDT is very well comparable with SVM^{light} in terms of numerical accuracy. The same conclusion held for the first version of the gradient projection-based decomposition technique introduced and extensively tested in [34]. Finally, the results in Table 4 emphasize that a good performance of the SVM methodology can be reached by stopping the training algorithm with a sufficiently large tolerance on the KKT residual. This behaviour is observed in many other SVM applications and suggests the default tolerance of 10^{-3} commonly used by the main training algorithms.

6 Conclusions

This work introduces two improvements to the gradient projection-based decomposition technique proposed in [31] for training Support Vector Machines.

The first improvement consists in a new efficient gradient projection method for the inner QP subproblems of the decomposition technique. The new method has been recently proposed by Dai and Fletcher in [9] and first tested in this work as inner solver within the decomposition technique. By an appropriate parameters setting, this solver has shown a better convergence rate and, consequently, a significant time reduction in comparison with the gradient projection method previously used by the decomposition technique.

The second improvement consists in a simple formula for deriving the data of the subproblems. This formula allows the linear term of the subproblem function to be obtained from quantities already computed, without requiring additional kernel evaluations. In this way, the total number of kernel evaluations required by the decomposition technique is generally reduced. Our numerical experiments have shown that in some situations this simple trick can imply much better performance.

Finally, a comparison with the SVM^{light} package in terms of both computational time and numerical accuracy confirms the effectiveness of the improved decomposition technique.

A software based on the proposed approach is available at http://dm.unife.it/gpdt/, for both serial and parallel architectures.

Acknowledgment

The author is most grateful to the referees for their helpful suggestions.

7 Appendix: test problems

All the numerical results of this work are obtained by solving the quadratic programs arising in training Gaussian SVMs on the following data sets:

- **UCI Adult data set**

  This data set [22], allows to train an SVM to predict whether a household has an income greater than $50000. After appropriate discretization [26], the inputs are 123-dimensional binary sparse vectors with sparsity level $\approx 89\%$. We train a Gaussian SVM with $C = 1$ and $\sigma = \sqrt{10}$ on the largest version of the data set, sized 32561.

- **Web data set**

  The data set [26], (available at <http://www.research.microsoft.com/~jplatt>), concerns a web page classification problem with a binary representation based on 300 keyword features. On average, the sparsity level of the examples is about 96%. We use the largest version of the data set sized 49749. The Gaussian SVM parameters are: $C = 5$ and $\sigma = \sqrt{10}$.

- **MNIST data set**

  The MNIST database of handwritten digits [18] contains 784-dimensional nonbinary sparse vectors; the size of the database is 60000 and the sparsity level of the inputs is $\approx 81\%$. A Gaussian SVM for the class "8" is trained with parameters $C = 10$ and $\sigma = 1800$.

References

[1] J. Barzilai, J.M. Borwein (1988), Two Point Step Size Gradient Methods, *IMA Journal of Numerical Analysis*, **8**, 141–148.

[2] D.P. Bertsekas (1999), Nonlinear Programming, Athena Scientific, Belmont, MA.

[3] E.G. Birgin, J.M. Martìnez, M. Raydan (2000), Nonmonotone Spectral Projected Gradient Methods on Convex Sets, *SIAM Journal on Optimization*, **10**(4), 1196–1211.

[4] B.E. Boser, I.M. Guyon, V.N. Vapnik (1992), A Training Algorithm for Optimal Margin Classifiers, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, D. Haussler, eds., ACM Press, Pittsburgh, PA, 144–152.

[5] C.C. Chang, C.J. Lin (2001), LIBSVM: a library for support vector machines, Software available at <http://www.csie.ntu.edu.tw/∼cjlin/libsvm>.

[6] R. Collobert, S. Benjo (2001), SVMTorch: Support Vector Machines for Large-Scale Regression Problems, *Journal of Machine Learning Research*, **1**, 143–160.

[7] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and other Kernel-Based Learning Methods, Cambridge University Press, (2000).

[8] Y.H. Dai, R. Fletcher (2003), Projected Barzilai-Borwein methods for Large-Scale Box-Constrained Quadratic Programming, Research Report NA/215, Department of Mathematics, University of Dundee.

[9] Y.H. Dai, R. Fletcher (2003), New Algorithms for Singly Linearly Constrained Quadratic Programs Subject to Lower and Upper Bounds, Research Report NA/216, Department of Mathematics, University of Dundee.

[10] R. Fletcher (2001), On the Barzilai-Borwein Method, Research Report NA/207, Department of Mathematics, University of Dundee.

[11] D. Goldfarb, A. Idnani (1983), A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs, *Mathematical Programming* **27**, 1–33.

[12] L. Grippo, F. Lampariello, S. Lucidi (1986), A Nonmonotone Line Search Technique for Newton's Method, *SIAM Journal on Numerical Analysis*, **23**, 707–716.

[13] L. Grippo, M. Sciandrone (2002), Nonmonotone Globalization Techniques for the Barzilai-Borwein Gradient Method, *Computational Optimization and Applications*, **23**, 143–169.

[14] C.W. Hsu, C.J. Lin (2002), A Simple Decomposition Method for Support Vector Machines, *Machine Learning*, **46**, 291–314.

[15] T. Joachims (1998), Making Large-Scale SVM Learning Practical, *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C.J.C. Burges and A. Smola, eds., MIT Press, Cambridge, MA.

[16] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy (2001), Improvements to Platt's SMO Algorithm for SVM Classifier Design, *Neural Computation* **13**, 637–649.

[17] S.S. Keerthi, E.G. Gilbert (2002), Convergence of a Generalized SMO Algorithm for SVM Classifier Design, *Machine Learning* **46**, 351-360.

[18] Y. LeCun, MNIST Handwritten Digit Database, available at <http://www.research.att.com/∼yann/ocr/mnist>.

[19] C.J. Lin (2001), On the Convergence of the Decomposition Method for Support Vector Machines, *IEEE Transactions on Neural Networks* **12**, 1288-1298.

[20] C.J. Lin (2001), Linear Convergence of a Decomposition Method for Support Vector Machines, Technical Report, Department of Computer Science and Information Engineering, National Taiwan University.

[21] C.J. Lin (2002), Asymptotic Convergence of an SMO Algorithm Without any Assumptions, *IEEE Transactions on Neural Networks* **13**, 248-250.

[22] P.M. Murphy, D.W. Aha (1992), UCI Repository of Machine Learning Databases, available at <http://www.ics.uci.edu/∼mlearn/MLRepository.html>.

[23] E. Osuna, R. Freund, F. Girosi (1997), Training Support Vector Machines: an application to face detection, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR97)*, Puerto Rico, 130–136.

[24] L. Palagi, M. Sciandrone (2005), On the Convergence of a Modified Version of SVM^{light} Algorithm, *Optimization Methods and Software*, **20**, 317-334.

[25] P.M. Pardalos, N. Kovoor (1990), An Algorithm for a Singly Constrained Class of Quadratic Programs Subject to Upper and Lower Bounds, *Mathematical Programming* **46**, 321–328.

[26] J.C. Platt (1998), Fast Training of Support Vector Machines using Sequential Minimal Optimization, *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C. Burges and A. Smola, eds., MIT Press, Cambridge, MA.

[27] J.C. Platt (1999), Using Analytic QP and Sparseness to Speed Training of Support Vector Machines, *Advances in Neural Information Processing Systems 11*, M.S. Kearns et al., eds., MIT Press, Cambridge, MA.

[28] V. Ruggiero, L. Zanni (2000), A Modified Projection Algorithm for Large Strictly Convex Quadratic Programs, *Journal of Optimization Theory and Applications*, **104**(2), 281–299.

[29] V. Ruggiero, L. Zanni (2000), Variable Projection Methods for Large Convex Quadratic Programs, *Recent Trends in Numerical Analysis*, D. Trigiante, ed., Advances in the Theory of Computational Mathematics 3, Nova Science Publ., 299–313.

[30] K. Schittkowski (2003), QL: A Fortran code for convex quadratic programming - User's guide, Report, Department of Mathematics, University of Bayreuth.

[31] T. Serafini, L. Zanni (2004), On the Working Set Selection in Gradient Projection-based Decomposition Techniques for Support Vector Machines, Technical Report N. 57, Department of Mathematics, University of Modena and Reggio Emilia, to appear on *Optimization Methods and Software*.

[32] T. Serafini, G. Zanghirati, L. Zanni (2005), Gradient Projection Methods for Quadratic Programs and Applications in Training Support Vector Machines, *Optimization Methods and Software*, **20**, 353-378.

[33] V.N. Vapnik (1998), Statistical Learning Theory, John Wiley and sons, New York.

[34] G. Zanghirati, L. Zanni (2003), A Parallel Solver for Large Quadratic Programs in Training Support Vector Machines, *Parallel Computing*, **29**, 535–551.