A Parallel Software for Training Large-Scale Support Vector Machines on Multiprocessor Systems

L. Zanni, T. Serafini, G. Zanghirati

Preprint n. 71 (Ottobre 2005)
A Parallel Software for Training Large-Scale Support Vector Machines on Multiprocessor Systems

Luca Zanni* Thomas Serafini* Gaetano Zanghirati**

* Department of Mathematics, University of Modena and Reggio Emilia, ITALY
  zanni.luca@unimo.it, seraﬁni.thomas@unimo.it
** Department of Mathematics, University of Ferrara, ITALY
  g.zanghirati@unife.it
October 5, 2005

Abstract

A parallel software for solving the quadratic program arising in training Support Vector Machines for classification problems is introduced. The software implements an iterative decomposition technique and exploits both the storage and the computing resources available on multiprocessor systems, by distributing the heaviest computational tasks of each decomposition iteration. Based on a wide range of recent theoretical advances, relevant decomposition issues are described, such as the quadratic subproblem solution, the gradient updating, the working set selection, and their parallel implementation is discussed. The behaviour of the proposed software with respect to state-of-the-art packages is evaluated by solving large-scale benchmark problems. The good accuracy and the remarkable time reduction exhibited make this software an useful tool in training large-scale Support Vector Machines on parallel architectures.

Keywords: Support Vector Machines, Large-Scale Quadratic Programs, Decomposition Techniques, Gradient Projection Methods, Parallel Computation

1 Introduction

Training Support Vector Machines (SVM) for binary classification requires to solve the following convex quadratic programming (QP) problem (Vapnik, 1998; Cristianini and Shawe-Taylor, 2000)

\[
\min \quad \mathcal{F}(x) = \frac{1}{2} x^T G x - \sum_{i=1}^{n} x_i \\
\text{sub. to} \quad \sum_{i=1}^{n} y_i x_i = 0, \quad 0 \leq x_i \leq C, \quad i = 1, \ldots, n,
\]

whose size $n$ is equal to the number of examples in the given training set

\[
D = \{(z_i, y_i), \quad i = 1, \ldots, n, \quad z_i \in \mathbb{R}^M, \quad y_i \in \{-1, 1\}\},
\]

*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

$$G_{ij} = y_i y_j K(z_i, z_j), \quad i, j = 1, 2, \ldots, n,$$

where $K : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ denotes the kernel function. The main features of this problem are the density of the quadratic form and the special feasible region defined by box constraints and a single linear equality constraint. In many practical SVM applications, standard QP solvers based on the explicit storage of the Hessian matrix $G$ may be very inefficient or, in the case of large data sets, even not applicable due to excessive memory requirements. For these reasons in recent years a lot of attention has been dedicated to this problem and several ad hoc strategies have been developed, which are able to solve the problem with $G$ out of memory. Among these strategies, the decomposition techniques have been the most investigated approaches and have given rise to the state-of-the-art software for the SVM QP problem. The idea behind the decomposition techniques consists in splitting the problem into a sequence of smaller QP subproblems, sized $n_{sp}$ say, that can be stored in the available memory and efficiently solved (Boser et al., 1992; Chang and Lin, 2001; Collobert and Benjo, 2001; Joachims, 1998; Osuna et al., 1997; Platt, 1998). At each decomposition step, a subset of the variables, usually called working set, is optimized through the solution of the subproblem in order to obtain a progress towards the minimum of the objective function $F(x)$. Effective implementations of this simple idea involve important theoretical and practical issues. From the theoretical point of view, the policy for updating the working set plays a crucial role since it can guarantee the strict decrease of the objective function at each step (Hush and Scovel, 2003). The most used working set selections rely on the violations of the Karush-Kuhn-Tucker (KKT) first order optimality conditions. In case of working sets of minimal size, i.e. sized 2, a proper selection via the “maximal violating pair” principle (or related criteria) is sufficient to ensure asymptotic convergence of the decomposition scheme (Lin, 2002; Chen et al., 2005). For larger working sets, convergence proofs are available under a further condition which ensures that the distance between two successive approximations tends to zero (Lin, 2001a; Palagi and Sciandrone, 2005). Furthermore, based on these working set selections and further assumptions, the linear convergence rate can be also proved (Lin, 2001b). For the practical efficiency of a decomposition technique, the fast convergence and the low computational cost per iteration seem the most important features. Unfortunately, these goals are conflicting since the strategies to improve the convergence rate (as the use of large working sets or the selections based on second order information) usually increase the cost per iteration. Examples of good trade-offs between the two goals are given by the most widely used decomposition packages: LIBSVM (Chang and Lin, 2001) and SVMlight (Joachims, 1998).

The LIBSVM software is developed for working sets sized 2, hence it tends to minimize the computational cost per iteration. In fact, in this case the inner QP subproblem can be analytically solved without requiring a numerical QP solver and the updating of the objective gradient only involves the two Hessian columns corresponding to the updated variables.
On the other hand, if only few components are updated per iteration, slow convergence is generally implied. In the last LIBSVM release (2.8) this drawback is attenuated by a new working set selection that partially exploits the second order information, thus getting only a moderate increase of the computational cost with respect to the standard selections (Fan et al., 2005).

The SVM\textsuperscript{light} algorithm uses a more general decomposition strategy, in the sense that it can also exploit working sets of size larger than 2. By updating more variables per iteration, such an approach is well suited for a faster convergence, but it introduces additional difficulties and costs. A generalized maximal-violating-pair policy for the working set selection and a numerical solver for the inner QP subproblems are needed; furthermore, we must recall that the more variables are changed per iteration, the more expensive is the objective gradient updating. Even if SVM\textsuperscript{light} can run with any working set size, numerical experiences show that it effectively faces the above difficulties only in case of small sized working sets ($n_{\text{sp}} = O(10)$), where it often exhibits comparable performance with LIBSVM.

Following the SVM\textsuperscript{light} decomposition framework, another attempt to reach a good trade-off between convergence rate and cost per iteration is introduced in (Zanghirati and Zanni, 2003). This was the first approach suited for an effective implementation on multiprocessors systems. Unlike SVM\textsuperscript{light}, it is designed to manage medium-to-large sized working sets ($n_{\text{sp}} = O(10^2)$ or $n_{\text{sp}} = O(10^3)$), that allow the scheme to converge in very few iterations, whose most expensive tasks (subproblem solving and gradient updating) can be easily and fruitfully distributed among the available processors. Of course, several issues must be addressed to achieve good performance, such as limiting the overhead for kernel evaluations and, also important, choosing a suitable inner QP solver. In (Zanghirati and Zanni, 2003) an efficient subproblem solution is obtained by a gradient projection-type method: it exploits the simple structure of the constraints, exhibits good convergence rate and is well suited for a parallel implementation. The promising results given by this parallel scheme can be now further improved thanks to some recent studies on both the gradient projection QP solvers (Serafini et al., 2005; Dai and Fletcher, 2005a) and the selection rules for large sized working sets (Serafini and Zanni, 2005).

On the basis of these studies a new Parallel Gradient Projection-based Decomposition Technique (PGPDT) is developed and implemented in a software available at http://www.dm.unife/gpdt. The main purpose of this work is to describe the PGPDT algorithm and to show how it can exploit the computing and storage resources of a multiprocessor system to achieve good time speedup in comparison with the serial training packages.

The paper is organized as follows: section 2 gives the decomposition technique and describes its parallelization, section 3 shows the behaviour of PGPDT on large-scale benchmark problems and evaluates its usefulness by comparison with the softwares SVM\textsuperscript{light} and LIBSVM, section 4 reports the main conclusions.
2 The decomposition framework and its parallelization

In order to describe in detail the decomposition technique implemented by PGPDT we need some basic notations. At each decomposition iteration, the indices of the variables \( x_i, \ i = 1, \ldots, n \), are split into the set \( \mathcal{B} \) of basic variables, usually called the working set, and the set \( \mathcal{N} = \{1, 2, \ldots, n\} \setminus \mathcal{B} \) of nonbasic variables. As a consequence, the kernel matrix \( G \) and the vectors \( x = (x_1, \ldots, x_n)^T \) and \( y = (y_1, \ldots, y_n)^T \) can be arranged with respect to \( \mathcal{B} \) and \( \mathcal{N} \) as follows:

\[
G = \begin{bmatrix}
G_{\mathcal{B}\mathcal{B}} & G_{\mathcal{B}\mathcal{N}} \\
G_{\mathcal{N}\mathcal{B}} & G_{\mathcal{N}\mathcal{N}}
\end{bmatrix}, \quad x = \begin{bmatrix}
x_{\mathcal{B}} \\
x_{\mathcal{N}}
\end{bmatrix}, \quad y = \begin{bmatrix}
y_{\mathcal{B}} \\
y_{\mathcal{N}}
\end{bmatrix}.
\]

Furthermore, we denote by \( n_{\text{sp}} \) the size of the working set \( (n_{\text{sp}} = \#\mathcal{B}) \) and by \( x^* \) a solution of (1). Finally, suppose a distributed-memory multiprocessor system equipped with \( N_p \) processors is available for solving the problem (1) and that each processor has a local copy of the training set.

The decomposition strategy used by the PGPDT falls within the general scheme stated in Algorithm PDT. Here, we denote by the label “Distributed task” the steps where the \( N_p \) processors cooperate to perform the required computation; in these steps communications and synchronization are needed. In the other steps, the processors asynchronously perform the same computations on the same input data to obtain a local copy of the expected output data.

It must be observed that algorithm PDT essentially follows the SVM\textsuperscript{light} decomposition scheme proposed by Joachims (1998) but it allows to distribute among the available processors the subproblem solution in step A2 and the gradient updating in step A3. Thus, two important implications can be remarked: from the theoretical viewpoint, the PDT algorithm satisfies the same convergence properties of the SVM\textsuperscript{light} algorithm, but, in practice, it requires new implementation strategies in order to effectively exploit the resources of a multiprocessor system. Here, we state the main convergence results of the PDT algorithm and we will describe in the next subsections how its steps have been implemented in the PGPDT software.

The convergence properties of the sequence \( \{x^{(k)}\} \) generated by the PDT algorithm are mainly based on the special rule (4) for the working set selection. The rule was originally introduced in (Joachims, 1998) following an idea similar to the Zoutendijk’s feasible direction approach, to define basic variables that make possible a rapid decrease of the objective function. The asymptotic convergence of the decomposition schemes based on this working set selection was first proved in (Lin, 2001a) by relating the selection rule with the violation of the KKT conditions and by assuming the following strict block-wise convexity assumption on \( \mathcal{F}(x) \):

\[
\min_{\mathcal{J}} (\lambda_{\text{min}}(G_{\mathcal{J}\mathcal{J}})) > 0, \tag{5}
\]

where \( \mathcal{J} \) is any subset of \( \{1, \ldots, n\} \) with \( \#\mathcal{J} \leq n_{\text{sp}} \) and \( \lambda_{\text{min}}(G_{\mathcal{J}\mathcal{J}}) \) denotes the smallest
Algorithm PDT Parallel Decomposition Technique

A1. Initialization. Set $x^{(1)} = 0$ and let $n_{sp}$ and $n_c$ be two integer values such that $n \geq n_{sp} \geq n_c > 0$, $n_c$ even. Choose $n_{sp}$ indices for the working set $\mathcal{B}$ and set $k \leftarrow 1$.

A2. QP subproblem solution. [Distributed task] Compute the solution $x^{(k+1)}_{\mathcal{B}}$ of

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

(2)

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

A3. Gradient updating. [Distributed task] Update the gradient

$$
\nabla F(x^{(k+1)}) = \nabla F(x^{(k)}) + \begin{bmatrix} G_{\mathcal{BB}} \\ G_{\mathcal{BN}} \end{bmatrix} \left( x^{(k+1)}_{\mathcal{B}} - x^{(k)}_{\mathcal{B}} \right)
$$

(3)

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

A4. Working set updating. Update $\mathcal{B}$ by the following selection rule:

A4.1. Find the indices corresponding to the nonzero components of the solution of

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

(4)

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

A4.2. Fill $\bar{\mathcal{B}}$ up to $n_{sp}$ entries with indices $j \in \mathcal{B}$. Set $\mathcal{B} = \bar{\mathcal{B}}$, $k \leftarrow k + 1$ and go to A2.
eigenvalue of $G_{\mathcal{J}}$. This condition is used to prove that there exists $\tau > 0$ such that

$$
\mathcal{F}(x^{(k+1)}) \leq \mathcal{F}(x^{(k)}) - \frac{\tau}{2}\|x^{(k+1)} - x^{(k)}\|^2, \quad \forall k,
$$

from which the important property $\lim_{k \to \infty} \|x^{(k+1)} - x^{(k)}\| = 0$ can be derived. Of course, the assumption (5) is satisfied when $G$ is positive definite (for example, when the Gaussian kernel is used and all the training examples are distinct), but it may not hold in other instances of the problem (1). Convergence results that do not require the condition (5) are given in (Lin, 2002) and (Palagi and Sciandrone, 2005).

In (Lin, 2002), for the special case $n_{\text{sp}} = 2$, where the selection rule (4) gives only the two indices corresponding to the maximal violation of the KKT conditions (the “maximal violating pair”), it has been shown that the assumption (5) is not necessary to ensure the convergence.

In (Palagi and Sciandrone, 2005), for any working set size, condition (6) is ensured by solving at each iteration the following proximal point modification of the subproblem (2):

$$
\begin{align*}
\min & \quad \frac{1}{2}x_B^T G_B x_B + \left(G_{\mathcal{N}} x_N^{(k)} - 1_B \right)^T x_B + \frac{\tau}{2}\|x_B - x_B^{(k)}\|^2 \\
\text{sub. to} & \quad \sum_{i \in B} y_i x_i = -\sum_{i \in N} y_i x_i^{(k)}, \\
& \quad 0 \leq x_i \leq C, \quad \forall i \in B.
\end{align*}
$$

Unfortunately, this modification affects the behaviour of standard decomposition schemes in a way which is not completely understood yet. Our preliminary experiences suggest that sufficiently large values of $\tau$ can easily allow a better performance of the inner QP solvers, but those values often imply a dangerous decrease in the convergence rate of the decomposition technique. On the other hand, too small values for $\tau$ do not produce essential differences with respect to the schemes where the subproblem (2) is solved.

In the PGPDT software, besides the default setting in which the standard PDT algorithm is followed, two different ways to generate a sequence satisfying the condition (6) are available by user selection: (i) solving the subproblem (7) in place of (2) at each iteration or (ii) solving (7) only as emergency step, when $x_B^{(k+1)}$ obtained via (2) fails to satisfy (6). All the computational experiments of section 3 are carried out with the default setting that generally yields the best performance. For what concerns the practical rule used in the PGPDT to stop the iterative procedure, the fulfilment of the KKT conditions within a prefixed tolerance is checked (with the equality constraint multiplier computed as suggested in (Joachims, 1998)). The default tolerance is $10^{-3}$, as is usual in SVM packages, but different values can be selected by the user.

Before to describe in detail the PGPDT implementation, it must be recalled that this software is designed to be effective in case of sufficiently large $n_{\text{sp}}$, i.e. when iterations with well parallelizable tasks are generated. For this reason, in the sequel the reader may assume $n_{\text{sp}}$ to be of medium-to-large size.
2.1 Parallel Gradient Projection Methods for the Subproblems

The inner QP subproblems (2) and (7) can fit into the following general form:

$$\min_{\mathbf{w} \in \Omega} \ f(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{A} \mathbf{w} + \mathbf{b}^T \mathbf{w}$$  \hspace{1cm} (8)$$

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

$$\Omega = \{ \mathbf{w} \in \mathbb{R}^{n_{sp}}, \ \ell \leq \mathbf{w} \leq \mathbf{u}, \ \mathbf{c}^T \mathbf{w} = \gamma \}, \quad \ell, \mathbf{u}, \mathbf{c} \in \mathbb{R}^{n_{sp}}, \ \ell < \mathbf{u}. \hspace{1cm} (9)$$

We recall that the size $n_{sp}$ is such that $\mathbf{A}$ can fit into the available memory.

Since subproblem (8) appears at each decomposition iteration, an effective inner solver becomes a crucial tool for the performance of a decomposition technique. The standard library QP solvers can be successfully applied only in the small size case ($n_{sp} = O(10)$), since their computational cost easily degrades the performance of a decomposition technique based on medium-to-large $n_{sp}$. For such kind of decomposition schemes, it is essential to design more efficient inner solvers able to exploit the special features of (8) and, for the PGPDT purposes, with the additional property to be easily parallelizable. To this end, the gradient projection methods are very appealing approaches (Bertsekas, 1999). They consist in a sequence of projections onto the feasible region, that are nonexpensive operations in the case of the special constraints (9). In fact, the projection onto $\Omega$ (denoted by $P_\Omega(\cdot)$) can be performed in $O(n_{sp})$ operations by efficient algorithms, like those in (Pardalos and Kovoor, 1990) and (Dai and Fletcher, 2005a). Furthermore, the single iteration core consists essentially in an $n_{sp}$-dimensional matrix-vector product that is suited to be optimized (by exploiting the vector sparsity) and also to be parallelized. Thus, the simple and nonexpensive iteration motivates the interest for these approaches as possible alternative to standard solvers based on expensive factorizations (usually requiring $O(n_{sp}^3)$ operations). A general parallel gradient projection scheme for (8) is depicted in Algorithm PGPM. As in the classical gradient projection methods, at each iteration a feasible descent direction $\mathbf{d}^{(k)}$ is obtained by projecting onto $\Omega$ a point derived by taking a steepest descent step from the current $\mathbf{w}^{(k)}$ with steplength $\alpha_k$. A linesearch procedure is then applied along the direction $\mathbf{d}^{(k)}$ to decide the step size $\lambda_k$ able to ensure the global convergence. The parallelization of this iterative scheme is simply obtained by a block row-wise distribution of $\mathbf{A}$ and by a parallel computation of the heaviest task of each iteration: the matrix-vector product $\mathbf{A}\mathbf{d}^{(k)}$.

Concerning the convergence rate, that is the key element for the PGPM performance, the choices of both the steplength $\alpha_k$ and the linesearch parameter $\lambda_k$ play a crucial role. Recent works have shown that appropriate selection rules for these parameters can significantly improve the typical slow convergence rate of the traditional gradient projection approaches (refer to (Ruggiero and Zanni, 2000b) for the R-linear convergence of PGPM-like schemes). From the steplength viewpoint, very promising results are actually obtained with selection
Algorithm PGPM Parallel Gradient Projection Method for step A2 of Algorithm PDT.

B1. Initialization.

[Data distribution] \( \forall p = 1, \ldots , N_p \): allocate a row-wise slice \( A_p = (a_{ij})_{i \in I_p, j = 1, \ldots , n_p} \) of \( A \), where \( I_p \) is the subset of row indices belonging to processor \( p \):

\[
I_p \subset \{1, \ldots , n\}, \quad \bigcup_{p=1}^{N_p} I_p = \{1, \ldots , n\}, \quad I_i \cap I_j = \emptyset, \quad \text{for} \ i \neq j.
\]

Furthermore, allocate local copy of all the other input data. Initialize the parameters for the steplength selection rule and for the linesearch strategy.

Set \( w(0) \in \Omega, \quad 0 < \alpha_{\min} < \alpha_{\max}, \quad \alpha_0 \in [\alpha_{\min}, \alpha_{\max}], \quad k = 0. \)

[Distributed task] \( \forall p = 1, \ldots , N_p \): compute the local slice \( t_p^{(0)} = A_p w(0) \) and send it to all the other processors; assemble a local copy of the full \( t^{(0)} = A w(0) \) vector.

Set \( g^{(0)} = \nabla f(w(0)) = A w(0) + b = t^{(0)} + b \).


Terminate if \( w^{(k)} \) satisfies a stopping criterion; otherwise compute the descent direction

\[
d^{(k)} = P_{\Omega}(w^{(k)} - \alpha_k g^{(k)}) - w^{(k)}.
\]

B3. Matrix-vector product.

[Distributed task] \( \forall p = 1, \ldots , N_p \): compute the local slice \( z_p^{(k)} = A_p d^{(k)} \) and send it to all the other processors; assemble a local copy of the full \( z^{(k)} = A d^{(k)} \) vector.

B4. Linesearch.

Compute the linesearch step \( \lambda_k \) and \( w^{(k+1)} = w^{(k)} + \lambda_k d^{(k)} \).

B5. Update.

Compute

\[
t^{(k+1)} = A w^{(k+1)} = t^{(k)} + \lambda_k A d^{(k)} = t^{(k)} + \lambda_k z^{(k)},
\]
\[
g^{(k+1)} = \nabla f(w^{(k+1)}) = A w^{(k+1)} + b = t^{(k+1)} + b,
\]

and a new steplength \( \alpha_{k+1} \).

Update the parameters for the linesearch strategy, set \( k \leftarrow k + 1 \) and go to step B2.
Initialization (step B1). Set $i_{\alpha} = 2$, $n_{\min} = 3$, $n_{\max} = 10$, $\lambda_{l} = 0.1$, $\lambda_{u} = 5$, $n_{\alpha} = 1$.

Linesearch (step B4). Compute $\lambda_{k} = \arg \min_{\lambda \in [0,1]} f(w^{(k)} + \lambda d^{(k)})$.

Update (step B5). If $d^{(k)^{T}} A d^{(k)} = 0$ then set $\alpha_{k+1} = \alpha_{\max}$ else compute $\alpha_{BB1}^{k+1}$, $\alpha_{BB2}^{k+1}$, $\lambda_{\text{opt}} = \arg \min_{\lambda} f(w^{(k)} + \lambda d^{(k)})$.

If ($n_{\alpha} \geq n_{\min}$) and ($n_{\alpha} \geq n_{\max}$) or ($\alpha_{BB2}^{k+1} \leq \alpha_{k} \leq \alpha_{BB1}^{k+1}$) or ($\lambda_{\text{opt}} < \lambda_{l}$ and $\alpha_{k} = \alpha_{BB1}^{k}$) or ($\lambda_{\text{opt}} > \lambda_{u}$ and $\alpha_{k} = \alpha_{BB2}^{k}$) then set $i_{\alpha} \leftarrow \text{mod}(i_{\alpha}, 2) + 1$, $n_{\alpha} = 0$; end.

Compute $\alpha_{k+1} = \min \{\alpha_{\max}, \max \{\alpha_{\min}, \alpha_{BBi_{\alpha}}^{k+1}\}\}$ and set $n_{\alpha} \leftarrow n_{\alpha} + 1$.

Figure 1: linesearch and steplength rule for the GVPM method.

strategies based on the Barzilai-Borwein (BB) rules (Barzilai and Borwein, 1988):

$$
\alpha_{BB1}^{k+1} = \frac{d^{(k)^{T}} d^{(k)}}{d^{(k)^{T}} A d^{(k)}}, \quad \alpha_{BB2}^{k+1} = \frac{d^{(k)^{T}} A d^{(k)}}{d^{(k)^{T}} A^2 d^{(k)}}.
$$

The importance of these rules has been observed in combination with both monotone and nonmonotone linesearch strategies (Birgin et al., 2000; Dai and Fletcher, 2005b; Ruggiero and Zanni, 2000a). In particular, for the SVM applications, the special BB steplength selections proposed for the monotone scheme in (Serafini et al., 2005) and the nonmonotone method in (Dai and Fletcher, 2005a) seem very efficient.

The Generalized Variable Projection Method (GVPM) in (Serafini et al., 2005) uses a standard limited minimization rule as linesearch technique and an adaptive alternation of the two BB formulae. It outperforms the monotone gradient projection scheme used in (Zanghirati and Zanni, 2003), that was simply based on an alternation of the BB rules every three iterations. Furthermore, the numerical experiments reported in (Serafini et al., 2005) show that the GVPM is much more efficient than the pr-LOQO (Smola, 1997) and MINOS (Murtagh and Saunders, 1998) solvers, two softwares widely used within the machine learning community. GVPM steplength selection and linesearch are described in Fig. 1.

The Dai-Fletcher scheme is based on the following steplength selection:

$$
\alpha_{DF}^{k+1} = \frac{\sum_{i=0}^{m-1} s^{(k-i)^{T}} s^{(k-i)}}{\sum_{i=0}^{m-1} s^{(k-i)^{T}} v^{(k-i)}}, \quad m \geq 1,
$$

where $s^{(j)} = w^{(j+1)} - w^{(j)}$ and $v^{(j)} = g^{(j+1)} - g^{(j)}$, ($g^{(j)} = \nabla f(w^{(j)})$), $j = 0, 1, \ldots$. Observe that the case $m = 1$ reduces to the standard BB rule $\alpha_{BB1}^{k+1}$. In order to frequently accept the
Initialization (step B1). Set $L = 2$, $f_{\text{ref}} = \infty$, $f_{\text{best}} = f_c = f(w(0))$, $h = 0$, $k = 0$, $s^{(k-1)} = v^{(k-1)} = 0$.

Linesearch (step B4). If $(k = 0$ and $f(w^{(k)} + d^{(k)}) \geq f(w^{(k)}))$ or $(k > 0$ and $f(w^{(k)} + d^{(k)}) \geq f_{\text{ref}})$ then
\[ w^{(k+1)} = w^{(k)} + \lambda_k d^{(k)} \] with $\lambda_k = \arg \min_{\lambda \in [0,1]} f(w^{(k)} + \lambda d^{(k)})$
else
\[ w^{(k+1)} = w^{(k)} + d^{(k)} \]
end. Update (step B5). Compute $s^{(k)} = w^{(k+1)} - w^{(k)}$, $v^{(k)} = g^{(k+1)} - g^{(k)}$.
If $s^{(k)}^T v^{(k)} = 0$ then
set $\alpha_{k+1} = \alpha_{\text{max}}$
else If $s^{(k-1)}^T v^{(k-1)} = 0$ then
set $\alpha_{k+1} = \min\left\{ \alpha_{\text{max}}, \max\left\{ \frac{s^{(k)}^T s^{(k)}}{s^{(k)}^T v^{(k)}}, \frac{s^{(k-1)}^T s^{(k-1)}}{s^{(k-1)}^T v^{(k-1)}} \right\} \right\}$
else
set $\alpha_{k+1} = \min\left\{ \alpha_{\text{max}}, \max\left\{ \frac{s^{(k)}^T s^{(k)}}{s^{(k)}^T v^{(k)}}, \frac{s^{(k-1)}^T s^{(k-1)}}{s^{(k-1)}^T v^{(k-1)}} \right\} \right\}$
end.
end.
If $f(w^{(k+1)}) < f_{\text{best}}$ then
set $f_{\text{best}} = f(w^{(k+1)})$, $f_c = f(w^{(k+1)})$, $h = 0$;
else set $f_c = \max\{f_c, f(w^{(k+1)})\}$, $h = h + 1$;
If $h = L$ then
set $f_{\text{ref}} = f_c$, $f_c = f(w^{(k+1)})$, $h = 0$;
end.
end.

Figure 2: linesearch and steplength rule for the Dai-Fletcher method.

full step $w^{(k+1)} = w^{(k)} + d^{(k)}$ generated with the above steplength, a special nonmonotone linesearch is used. Fig. 2 describes the version of the Dai-Fletcher method corresponding to the parameters setting suggested in (Zanni, 2005) for the SVM applications. It may be observed that the linesearch parameter $\lambda_k = \arg \min_{\lambda \in [0,1]} f(w^{(k)} + \lambda d^{(k)})$ is used only if $f(w^{(k)} + d^{(k)}) \geq f_{\text{ref}}$ and not at each iteration, as in the GVPM. The steplength selection corresponds to the rule (10) with $m = 2$ and, for what concerns the iteration cost, no significant additional tasks are required in comparison to the GVPM ($g^{(k+1)}$ is already available in step B5).

The PGPDT software can run the PGPM with either the GVPM or the Dai-Fletcher scheme, the latter being the default due to better experimental convergence rate (Zanni,
We end this subsection with some further details about the PGPM implementation used within the PGPDT software. The starting point $w^{(0)}(x_{k}^{(k)})$ is used if the stopping rule of the decomposition procedure is nearly satisfied, otherwise $w^{(0)} = P_{\Omega}(0)$ is used. This aims to start the PGPM with sparse vectors in the first decomposition steps, and to save inner solver iterations at the end of the decomposition, where slight changes in $x_{k}^{(k)}$ are expected. At the beginning we also set $\alpha_{\text{min}} = 10^{-10}$, $\alpha_{\text{max}} = 10^{10}$ and $\alpha_{0} = \min\{\alpha_{\text{max}}, \max\{\alpha_{\text{min}}, \bar{\alpha}_{0}\}\}$, where $\bar{\alpha}_{0} = \|P_{\Omega}(w^{(0)}(0) - (A w^{(0)} + b)) - w^{(0)}\|_{\infty}^{-1}$. For the computation of $P_{\Omega}(\cdot)$ in step B2, by default if $n_{\text{sp}} \leq 20$ the bisection-like method described in (Pardalos and Kovoor, 1990) is used, else the secant-based algorithm proposed in (Dai and Fletcher, 2005a) is chosen, that usually is faster for large size. However, the user can select one of the two projector. Finally, we remark that the PGPM stopping rule is the same used for the decomposition technique: the fulfilment of the KKT conditions within a prefixed tolerance. In the PGPDT, the tolerance required to the inner solver depends on the quality of the outer iterate $x^{(k)}$: in the first iterations the same tolerance as the decomposition scheme is used, while a progressive lower tolerance is imposed when $x^{(k)}$ nearly satisfies the outer stopping criterion. In our experience, a more accurate inner solution just from the beginning doesn’t imply remarkable increase of the overall performance.

2.2 Parallel Gradient Updating

The gradient updating in step A3 is usually the most expensive task of a decomposition iteration. Since the matrix $G$ is assumed out of memory, in order to obtain $\nabla F(x^{(k+1)})$ some entries of $G$ need to be computed and, consequently, some kernel evaluations are involved that can be very expensive in case of large sized input space and not much sparse training examples. Thus, any strategy able to save kernel evaluations or to optimize their computation is crucial for minimizing the time consumption for updating the gradient. The updating formula (3) allows to obtain $\nabla F(x^{(k+1)})$ by involving only the columns of $G$ corresponding to the indices for which $(x_{i}^{(k+1)} - x_{i}^{(k)}) \neq 0, i \in B$. Further improvements in the number of kernel evaluations can be obtained by introducing some kind of caching strategy, consisting in using an area of the available memory to store some elements of $G$ in order to avoid their recomputation in subsequent iterations. PGPDT fills the caching area with the columns of $G$ involved in (3); when the cache is full, the current columns substitute those that have not been used for the greatest number of iterations. This simple trick seems to well combine with the working set selection used in step A4, which forces some indices of the current $B$ to remain in the new working set (see the next section for more details), and remarkable reduction of the kernel evaluations are often observed. Nevertheless, the improvements implied by a caching strategy are obviously dependent on the size of the caching area. To this regard, the large amount of memory available on modern multiprocessor systems is an appealing resource for improving the performance of a decomposition technique. One of the innovative features of PGPDT is to implement a parallel gradient updating where both the matrix-vector
Algorithm PGU Parallel Gradient Updating in step A3 of Algorithm PDT

i) Denote by $W_p$, $p = 1, 2, \ldots, N_P$, the caching area of the processor $p$ and by $G_i$ the i-th column of $G$. Let

$$B_1 = \{i \in B \mid (x_i^{(k+1)} - x_i^{(k)}) \neq 0\},$$
$$B_n = \{i \in B_1 \mid G_i \notin W_p, \ p = 1, 2, \ldots, N_P\}, \quad B_c = B_1 \setminus B_n.$$ 

Distribute among the processors the set $B_c$ and $B_n$ and denote by $B_{c,p}$ and $B_{n,p}$ the set of indices assigned to processor $p$. Make the distribution in such a way that

$$B_c = \bigcup_{i=1}^{N_P} B_{c,i}, \quad B_{c,i} \cap B_{c,j} = \emptyset \text{ for } i \neq j, \quad \forall i \in B_{c,p} \Rightarrow G_i \in W_p,$$

$$B_n = \bigcup_{i=1}^{N_P} B_{n,i}, \quad B_{n,i} \cap B_{n,j} = \emptyset \text{ for } i \neq j.$$ 

and by trying to obtain a well balanced workload among the processors.

ii) $\forall p = 1, 2, \ldots, N_P$: compute, by using the columns $G_i$, $i \in B_{c,p}$, available in the caching area $W_p$,

$$r_p = \begin{bmatrix} G_{BB_{c,p}} \\ G_{NB_{c,p}} \end{bmatrix} \begin{pmatrix} x_{B_{c,p}}^{(k+1)} - x_{B_{c,p}}^{(k)} \end{pmatrix},$$

then compute the columns $G_i$, $i \in B_{n,p}$, necessary to obtain

$$r_p \leftarrow r_p + \begin{bmatrix} G_{BB_{n,p}} \\ G_{NB_{n,p}} \end{bmatrix} \begin{pmatrix} x_{B_{n,p}}^{(k+1)} - x_{B_{n,p}}^{(k)} \end{pmatrix}$$

and store in $W_p$ as much as possible of these columns, eventually by substituting those less recently used.

iii) $\forall p = 1, 2, \ldots, N_P$: send $r_p$ to all the other processors and assemble a local copy of

$$\nabla F(x^{(k+1)}) = \nabla F(x^{(k)}) + \sum_{i=1}^{N_P} r_i.$$
multiplication and the caching strategy are distributed among the processors. This is done by asking each processor to perform a part of the column combinations required in (3) and to make available its local memory for caching the columns of \( G \). In this way, the gradient updating benefits not only from a computations distribution, but also from a reduction of the kernel evaluations due to much larger caching areas. Of course, these features are not shared by standard decomposition packages, designed to exploit the resources of only one processor. The main steps of the above parallel updating procedure are summarized in Algorithm PGU.

Concerning the reduction of the kernel evaluations, it is worth to recall that the entries of \( G \) stored in the caching area can be used also for \( G_{BB} \) in step A2. Moreover, for the computation of the linear term in (2), the equality

\[
G_{BN}x^{(k)} - 1_B = \nabla F_B(x^{(k)}) - G_{BB}x^{(k)}
\]

can avoid additional kernel evaluations by exploiting already computed quantities.

The gradient updating overhead within each decomposition iteration can be further reduced by optimizing the kernel computation. Even if a caching strategy can limit the number of kernel evaluations, large problems often require millions of them and their optimization becomes a need. PGPDT uses sparse vector representation of the training examples and exploits the sparseness in the dot products required by the kernel evaluations. Three kernels are available: linear, polynomial and Gaussian. The interested reader is referred to the available code for more details on their practical implementation. We end this section by remarking that, in case of linear kernel, the updating formula (3) can be simplified in

\[
t = \sum_{i \in B_1} y_i z_i \left( x^{(k+1)}_i - x^{(k)}_i \right), \quad B_1 = \{ i \in B \mid (x^{(k+1)}_i - x^{(k)}_i) \neq 0 \},
\]

\[
\nabla F(x^{(k+1)})_j = \nabla F(x^{(k)})_j + y_j z^T_j t, \quad j = 1, 2, \ldots, n,
\]

and the importance of a caching strategy is generally negligible. Consequently, PGPDT faces linear SVMs without any caching strategy and performs the gradient updating by simply distributing the \( n \) tasks (11) among the processors.

### 2.3 Working Set Selection

In this section we describe how the working set updating in step A4 of the PDT algorithm is implemented within PGPDT. It consists in two phases: in the first phase at most \( n_c \) indices are chosen for the new working set by solving the problem (4), while in the second phase at least \( n_{sp} - n_c \) entries are selected from the current \( B \) to complete the new working set. The selection procedure in step A4.1 was first introduced in (Joachims, 1998) and then rigorously justified in (Lin, 2001a). In short, by using the notation

\[
I_{\text{top}}(x) \equiv \{ i \mid (x_i < C \text{ and } y_i = -1) \text{ or } (x_i > 0 \text{ and } y_i = 1) \},
\]

\[
I_{\text{bot}}(x) \equiv \{ j \mid (x_j > 0 \text{ and } y_j = -1) \text{ or } (x_j < C \text{ and } y_j = 1) \},
\]
Algorithm SP1: Selection procedure for step A4.1 of algorithm PDT.

i) Sort the indices of the variables according to $y_i \nabla F(x^{(k+1)})_i$ in decreasing order and let $I \equiv (i_1, i_2, \ldots, i_n)^T$ be the sorted list (i.e., $y_{i_1} \nabla F(x^{(k+1)})_1 \geq y_{i_2} \nabla F(x^{(k+1)})_2 \geq \ldots \geq y_{i_n} \nabla F(x^{(k+1)})_n$).

ii) Repeat the selection of a pair $(i_t, i_b) \in I \times I$, with $t < b$, as follows:

- moving down from the top of the sorted list, choose $i_t \in I_{\text{top}}(x^{(k+1)})$,
- moving up from the bottom of the sorted list, choose $i_b \in I_{\text{bot}}(x^{(k+1)})$,

until $n_c$ indices are selected or a pair with the above properties cannot be found.

iii) Let $B$ be the set of the selected indices.

This procedure can be stated as in Alg. SP1.

It is interesting to recall how this selection procedure is related to the violation of the first order optimality conditions. For the convex problem (1) the KKT conditions can also be written as

$$
\text{a feasible } x^\ast \text{ is optimal } \iff \max_{i \in I_{\text{top}}(x^\ast)} y_i \nabla F(x^\ast)_i \leq \min_{j \in I_{\text{bot}}(x^\ast)} y_j \nabla F(x^\ast)_j .
$$

It means that, given a non-optimal feasible $x$, there exists at least a pair $(i, j) \in I_{\text{top}}(x) \times I_{\text{bot}}(x)$ satisfying

$$
y_i \nabla F(x)_i > y_j \nabla F(x)_j .
$$

Following (Keerthi and Gilbert, 2002), these pairs are called “KKT violating pairs” and, from this point of view, the above selection procedure chooses indices $(i, j) \in I_{\text{top}}(x^{(k+1)}) \times I_{\text{bot}}(x^{(k+1)})$ by giving priority to those pairs which most violate the optimality conditions. In particular, at each iteration the maximal violating pair is included in the working set: this property is crucial for the asymptotic convergence of a decomposition technique.

From the practical viewpoint, the indices selected via problem (4) identify steepest-like feasible descent directions: this is aimed to get a quick decrease of the objective function $F(x)$. Nevertheless, for fast convergence, both $n_c$ and the updating phase in step A4.2 have a key relevance. In fact, as it is experimentally shown in (Serafini and Zanni, 2005), values of $n_c$ equal or close to $n_{sp}$ often yield a dangerous zigzagging phenomenon (i.e., some variables enter and leave the working set many times), which can heavily degrade the convergence rate especially for large $n_{sp}$. This drawback suggests to set $n_c$ sufficiently smaller than $n_{sp}$ and then it opens the problem of how to select the remaining indices to fill up the new working set. The studies available in literature on this topic (see Hsu and Lin (2002); Serafini and Zanni (2005); Zanghirati and Zanni (2003) and also the SVMlight code) suggest that an efficient approach consists in selecting these indices from the current working set. We recall in Alg. SP2 the filling strategy recently proposed in (Serafini and Zanni, 2005) and used by the PGPDT software.
Algorithm SP2 Selection procedure for step A4.2 of algorithm PDT.

i) Let $\bar{B}$ be the set of indices selected in step A4.1.

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

iii) Set $n_c = \min\{ n_c, \max\{10, J, n_{new}\}\}$, where $J$ is the largest even integer such that $J \leq \frac{n_{sp}}{10}$ and $n_{new}$ is the largest even integer such that $n_{new} \leq \#\{ j, j \in \bar{B} \setminus B\}$; set $B = \bar{B}$, $k \leftarrow k + 1$ and go to step A2.

†We mean the indices that are in the working set $B$ since the lowest number of consecutive iterations.

The selection policy used by Alg. SP2 is based on two criteria: the first accords priority to the free variables over the variables at either the lower or the upper bound, the second takes into account how long (i.e., how many consecutive decomposition iterations) a variable has been into the working set. Roughly speaking, both the criteria aim to preserve into the working set the variables which are likely to need further optimization. The interested reader can find in (Hsu and Lin, 2002) and (Serafini and Zanni, 2005) a deeper discussion on these criteria and the computational evidence of their benefits in terms of convergence rate.

Finally, Alg. SP2 also introduces an adaptive reduction of the parameter $n_c$, useful in case of large-sized working sets. This trick allows the decomposition technique to start with $n_c$ close to $n_{sp}$, in order to optimize many new variables in the very first iterations, and avoids zigzagging through the progressive reduction of $n_c$. The reduction takes place only if $n_c$ is larger than an empirical threshold and it is controlled via the number of those new indices selected in step A4.1 that do not belong to the current working set.

3 Computational Experiments

The aim of this computational study is to analyse the PGPDT performance. To this end, it is worth to also show that the serial version of the proposed software (called GPDT) can train SVMs with effectiveness comparable to that of the state-of-the-art softwares LIBSVM and SVMlight.

Our implementation is an object oriented C++ code and its parallel version uses standard MPI communication routines (Message Passing Interface Forum, 1995), hence it is easily portable on many multiprocessor systems. Most of the experiments are carried out on an IBM SP5, which is an IBM SP Cluster 1600 equipped with 64 nodes p5-575 interconnected by a high performance switch (HPS). Each node owns 8 IBM SMP Power5 processors at 1.9GHz and 16GB of RAM (2GB per CPU). The serial packages run on this computer by exploiting only a single CPU. PGPDT has been tested also on different parallel architectures and, for completeness, we report the results obtained on a system where less memory than in the IBM SP5 is available for each CPU: the IBM CLX/1024 Linux Cluster, that owns 512 nodes equipped with two Intel Xeon processors at 3.0GHz and 1GB of RAM per CPU.
Both the systems are available at CINECA Supercomputing center (Bologna, Italy, http://www.cineca.it).

The considered softwares are compared on several medium-to-large test problems generated from well known benchmark data sets, described in the next subsection.

3.1 Test problems

We trained Gaussian and polynomial SVMs with kernel functions $K(z_i,z_j) = \exp(-\|z_i-z_j\|^2/(2\sigma^2))$ and $K(z_i,z_j) = (s(z_i^Tz_j) + 1)^d$, respectively.

In what follows we give some details on the databases used for the generation of the training sets, as well as on the SVM parameters we have chosen.

The UCI Adult data set (available at http://www.research.microsoft.com/~jplatt/smo.html) allows to train an SVM to predict whether a household has an income greater than $50000. The inputs are 123-dimensional binary sparse vectors with sparsity level $\approx 89\%$. We use the largest version of the data set, sized 32561. We train a Gaussian SVM with training parameters according to the database technical documentation: $C = 1$, $\sigma = \sqrt{10}$.

The Web data set (available at http://www.research.microsoft.com/~jplatt/smo.html) 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. We train a Gaussian SVM with the parameters suggested in the data set documentation: $C = 5$ and $\sigma = \sqrt{10}$.

The MNIST database of handwritten digits (available at http://yann.lecun.com/exdb/mnist) contains 784-dimensional nonbinary sparse vectors; the data set size is 60000 and the data sparsity is $\approx 81\%$. The provided test set is sized 10000. We train two SVM classifiers for the digit “8” with the following parameters: $C = 10$, $\sigma = 1800$ for the Gaussian kernel and $C = 3000$, $d = 4$, $s = 3 \cdot 10^{-9}$ for the polynomial kernel. This setting gives the following error rates on the test set: 0.55\% for the Gaussian kernel and 0.60\% for the polynomial kernel.

The KDDCUP-99 Intrusion Detection data set (available at http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html) consists in binary TCP dump data from seven weeks of network traffic. Each original pattern has 34 continuous features and 7 symbolic features. As in (Tseng et al., 2005) we normalize each continuous feature to the range $[0,1]$ and transform each symbolic feature to multiple binary features. The original training set consists in 4898431 examples containing repeated vectors. We work with a subset of size 400000, obtained by random sampling the original database. We use a Gaussian kernel with parameters $\sigma^2 = (1.2)^{-1}$, $C = 2$. This choice yields an error rate of about 7.4\% on the test set of 311029 examples available in the database.

The Forest Cover Type data set (available at http://ftp://ftp.ics.uci.edu/pub/machine-learning-databases/covtype) has 581012 samples with 54 attributes, distributed in 8 classes. We train some SVM classifiers for separating class 2 from the other classes. The training sets, sized up to 300000, are generated by randomly sampling the data set. We use a Gaussian kernel with $\sigma^2 = 2.5 \cdot 10^4$, $C = 10$. For the largest training set the error rate is
Table 1: performance of the serial packages on different test problems.

<table>
<thead>
<tr>
<th>Data set</th>
<th>n</th>
<th>n&lt;sub&gt;sp&lt;/sub&gt;</th>
<th>n&lt;sub&gt;c&lt;/sub&gt;</th>
<th>sec.</th>
<th>it.</th>
<th>MKernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCI Adult</td>
<td>32561</td>
<td>400</td>
<td>200</td>
<td>93.6</td>
<td>129</td>
<td>498.5</td>
</tr>
<tr>
<td>MNIST (poly)</td>
<td>60000</td>
<td>600</td>
<td>200</td>
<td>345.3</td>
<td>221</td>
<td>324.4</td>
</tr>
<tr>
<td>MNIST (Gauss)</td>
<td>60000</td>
<td>2000</td>
<td>300</td>
<td>341.2</td>
<td>22</td>
<td>396.4</td>
</tr>
<tr>
<td>Web Pages</td>
<td>49749</td>
<td>600</td>
<td>200</td>
<td>62.2</td>
<td>101</td>
<td>252.9</td>
</tr>
<tr>
<td>Cover Type</td>
<td>300000</td>
<td>500</td>
<td>80</td>
<td>21561.4</td>
<td>5018</td>
<td>99880.0</td>
</tr>
<tr>
<td>KDDCUP-99</td>
<td>400000</td>
<td>180</td>
<td>60</td>
<td>9190.3</td>
<td>2248</td>
<td>51336.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data set</th>
<th>n</th>
<th>n&lt;sub&gt;sp&lt;/sub&gt;</th>
<th>n&lt;sub&gt;c&lt;/sub&gt;</th>
<th>sec.</th>
<th>it.</th>
<th>MKernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCI Adult</td>
<td>32561</td>
<td>2</td>
<td>2</td>
<td>165.9</td>
<td>15388</td>
<td>452.1</td>
</tr>
<tr>
<td>MNIST (poly)</td>
<td>60000</td>
<td>2</td>
<td>2</td>
<td>2154.4</td>
<td>452836</td>
<td>792.0</td>
</tr>
<tr>
<td>MNIST (Gauss)</td>
<td>60000</td>
<td>2</td>
<td>2</td>
<td>1081.8</td>
<td>20533</td>
<td>409.4</td>
</tr>
<tr>
<td>Web Pages</td>
<td>49749</td>
<td>2</td>
<td>2</td>
<td>64.0</td>
<td>13237</td>
<td>170.3</td>
</tr>
<tr>
<td>Cover Type</td>
<td>300000</td>
<td>2</td>
<td>2</td>
<td>17271.7</td>
<td>274092</td>
<td>53152.6</td>
</tr>
<tr>
<td>KDDCUP-99</td>
<td>400000</td>
<td>2</td>
<td>2</td>
<td>11220.8</td>
<td>40767</td>
<td>50773.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data set</th>
<th>n</th>
<th>n&lt;sub&gt;sp&lt;/sub&gt;</th>
<th>n&lt;sub&gt;c&lt;/sub&gt;</th>
<th>sec.</th>
<th>it.</th>
<th>MKernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCI Adult</td>
<td>32561</td>
<td>20</td>
<td>10</td>
<td>278.8</td>
<td>4648</td>
<td>436.8</td>
</tr>
<tr>
<td>MNIST (poly)</td>
<td>60000</td>
<td>8</td>
<td>4</td>
<td>4212.1</td>
<td>230784</td>
<td>1282.1</td>
</tr>
<tr>
<td>MNIST (Gauss)</td>
<td>60000</td>
<td>8</td>
<td>4</td>
<td>705.2</td>
<td>56921</td>
<td>263.1</td>
</tr>
<tr>
<td>Web Pages</td>
<td>49749</td>
<td>8</td>
<td>4</td>
<td>96.9</td>
<td>7465</td>
<td>175.3</td>
</tr>
<tr>
<td>Cover Type</td>
<td>300000</td>
<td>20</td>
<td>10</td>
<td>29983.9</td>
<td>61362</td>
<td>56065.6</td>
</tr>
<tr>
<td>KDDCUP-99</td>
<td>400000</td>
<td>10</td>
<td>10</td>
<td>10865.6</td>
<td>27808</td>
<td>27755.0</td>
</tr>
</tbody>
</table>

about 3.6% on the test set given by the remaining 281012 examples.

3.2 Serial behaviour

In the first experiments set, we analyse the behaviour of the serial code on the test problems just described. In Tab. 1 we report the time in seconds (sec.), the iteration count (it.) and the number of kernel evaluations in millions (MKernel) required for each one of the considered SVM training packages. The values we use for the working set parameters n<sub>sp</sub> and n<sub>c</sub> are also reported: as mentioned, the LIBSVM software works only with n<sub>sp</sub> = n<sub>c</sub> = 2, whilst both SVM<sup>light</sup> and GPDT accept larger values. We report those values that gave us the best training time and we remark that the default values (n<sub>sp</sub> = n<sub>c</sub> = 10 for SVM<sup>light</sup>, n<sub>sp</sub> = 400, n<sub>c</sub> = ⌊n<sub>sp</sub>/3⌋ = 132 for GPDT) do not imply large differences in the performance. We run the codes assigning 768MB to the caching area and requiring the default threshold \( \epsilon = 10^{-3} \) for the termination criterion, except for the two largest KDDCUP-99 and Cover Type test problems, where the stopping tolerance \( \epsilon \) is set to \( 10^{-2} \). All the other parameters are assigned default values.

Tab. 1 well emphasizes the different approach of the three softwares. In particular we see how GPDT, by exploiting large working sets, converges in far less iterations than the other softwares, but its iterations are much heavier. Looking at the computational time, GPDT seems to be very competitive with respect to both LIBSVM and SVM<sup>light</sup>. Furthermore,
Figure 3: training time for different test problems from the Cover Type data set.

Table 2: accuracy of the serial solvers.

<table>
<thead>
<tr>
<th>Solver</th>
<th>SV</th>
<th>BSV</th>
<th>$\mathcal{F}_{\text{opt}}$</th>
<th>$b$</th>
<th>test error</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MNIST (poly) test problem</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPDT</td>
<td>2712</td>
<td>640</td>
<td>-2555033.8</td>
<td>3.54283</td>
<td>0.63%</td>
</tr>
<tr>
<td>LIBSVM</td>
<td>2715</td>
<td>640</td>
<td>-2555033.6</td>
<td>3.54231</td>
<td>0.63%</td>
</tr>
<tr>
<td>SVM\text{light}</td>
<td>2714</td>
<td>640</td>
<td>-2555033.0</td>
<td>3.54213</td>
<td>0.62%</td>
</tr>
<tr>
<td><strong>Cover Type test problem</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPDT</td>
<td>50853</td>
<td>32683</td>
<td>-299399.7</td>
<td>0.22083</td>
<td>3.62%</td>
</tr>
<tr>
<td>LIBSVM</td>
<td>51131</td>
<td>32573</td>
<td>-299396.0</td>
<td>0.22110</td>
<td>3.63%</td>
</tr>
<tr>
<td>SVM\text{light}</td>
<td>51326</td>
<td>32511</td>
<td>-299393.9</td>
<td>0.22149</td>
<td>3.62%</td>
</tr>
</tbody>
</table>

The kernel column highlights how GPDT benefits from a good optimization of the kernel computation (compare, for instance, the results for the MNIST Gaussian test).

The next experiments are intended to underline how very similar performance are observed, even in the worst case for GPDT. Fig. 3 shows for the Cover Type test problem the training time with respect to both the problem size (Fig. 3a) and the number of support vectors (Fig. 3b): all the packages exhibit almost the same dependence.

Also from the accuracy viewpoint, in our experience the three packages always behave essentially in the same way. This is shown here for two of the considered test problems, by reporting in Tab. 2 the number of support vectors (SV) and bound support vectors (BSV), the computed optimal value $\mathcal{F}_{\text{opt}}$ of the objective function, the bias $b$ of the separating surface expression (Cristianini and Shawe-Taylor, 2000) and the error rate on the test set.

### 3.3 Parallel behaviour

The second experiments set concerns with the behaviour of PGPDT. Due to the little training time required by the UCI Adult and the Web pages test problems, we evaluate PGPDT on the other four large-scale problems.
For a meaningful comparison against the serial version, PGPDT is run with the same $n_{sp}$, $n_c$ and $\epsilon$ parameters as in the previous experiments; furthermore, the same amount of caching area (768MB) is now allocated on each CPU. Default values are assigned to the other parameters.

Tab. 3 and Fig. 4 summarize the results obtained by running PGPDT on different numbers of processors. We evaluate the parallel performance by the relative speedup, defined as $sp_r = T_{serial}/T_{parallel}$, where $T_{serial}$ is the training time spent on a single processor, while $T_{parallel}$ denotes the training time on $N_P$ processors.

Seeking clearness, in Tab. 3 we also report additional information on the overall PGPDT behaviour. In particular, we can see an essentially constant number of decomposition iterations (recall that only the computational burden within the decomposition iteration is distributed) and the same solution accuracy as the serial run (compare the numbers in SV, BSV and $F_{opt}$ columns). Moreover, remark the lower number of total kernel evaluations needed by the parallel version, due to the growing amount of global caching memory available, which our parallel caching strategy is able to exploit. This is the motivation of the superlinear speedup observed in some situations like the MNIST (Gaussian) test problem (Fig. 4a). Unfortunately, there may be cases where the benefits due to the parallel caching strategy are not sufficient to ensure optimal speedups. For instance, sometimes the $n_{sp}$ values that give satisfactory serial performance are not suited for good PGPDT scaling. This is the case of the KDDCUP-99 test problem (Fig. 4b), where the small working sets sized $n_{sp} = 180$ imply many decomposition iterations and consequently the fixed costs of the nondistributed tasks (working set selection and stopping rule) become very heavy. Another example is provided by the MNIST (polynomial) test problem: here the subproblem solution is a dominant task in comparison to the gradient updating and the suboptimal scaling of the PGPM solver on 16 processors leads to poor speedups. However, also in these cases remarkable time reductions are observed compared to serial softwares.

As a final remark, it can be underlined that all these considerations are quite dependent on the underlying parallel architecture. In particular, on multiprocessor systems where less memory than in the SP5 platform is available for each CPU, even better results can be expected due to the effectiveness of the parallel caching strategy. For instance, we report in Fig. 5 what we get for the KDDCUP-99 test problem on the IBM CLX/1024 Linux Cluster, where only 400MB of caching area can be allocated on each CPU. Due to both the worse performance of this machine and the reduced caching area, larger training time is required, but an optimal PGPDT speedup is now observed up to 16 processors.

4 Conclusions

A parallel software to train linear and nonlinear SVMs for classification problems is presented, which is suitable for distributed memory multiprocessors systems. It implements an iterative decomposition technique based on a gradient projection solver for the inner subproblems. At
Table 3: PGPDT scaling on the IBM SP5 system.

<table>
<thead>
<tr>
<th>$N_P$</th>
<th>sec.</th>
<th>$sp_r$</th>
<th>it.</th>
<th>MKernel</th>
<th>SV</th>
<th>BSV</th>
<th>$J_{opt}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>345.3</td>
<td>221</td>
<td>324.2</td>
<td>2712</td>
<td>640</td>
<td>−2555033.8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>158.6</td>
<td>2.18</td>
<td>212</td>
<td>249.2</td>
<td>2710</td>
<td>640</td>
<td>−2555033.8</td>
</tr>
<tr>
<td>4</td>
<td>100.5</td>
<td>3.44</td>
<td>214</td>
<td>253.9</td>
<td>2711</td>
<td>641</td>
<td>−2555033.8</td>
</tr>
<tr>
<td>8</td>
<td>59.7</td>
<td>5.78</td>
<td>212</td>
<td>259.8</td>
<td>2711</td>
<td>641</td>
<td>−2555033.7</td>
</tr>
<tr>
<td>16</td>
<td>47.3</td>
<td>7.30</td>
<td>217</td>
<td>271.4</td>
<td>2711</td>
<td>641</td>
<td>−2555033.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$N_P$</th>
<th>sec.</th>
<th>$sp_r$</th>
<th>it.</th>
<th>MKernel</th>
<th>SV</th>
<th>BSV</th>
<th>$J_{opt}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21561</td>
<td>5018</td>
<td>99880</td>
<td>50853</td>
<td>32683</td>
<td>−299399.7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>11123</td>
<td>1.94</td>
<td>5047</td>
<td>98925</td>
<td>50786</td>
<td>32673</td>
<td>−299399.8</td>
</tr>
<tr>
<td>4</td>
<td>5715</td>
<td>3.77</td>
<td>5059</td>
<td>93597</td>
<td>50786</td>
<td>32668</td>
<td>−299399.9</td>
</tr>
<tr>
<td>8</td>
<td>3016</td>
<td>7.15</td>
<td>5086</td>
<td>82853</td>
<td>50832</td>
<td>32664</td>
<td>−299399.9</td>
</tr>
<tr>
<td>16</td>
<td>1673</td>
<td>12.89</td>
<td>5029</td>
<td>59439</td>
<td>50826</td>
<td>32697</td>
<td>−299399.9</td>
</tr>
</tbody>
</table>

![Figure 4: PGPDT scaling on the IBM SP5 system.](image)

5 Acknowledgments

The authors are grateful to the staff of the CINECA Supercomputing Center (Bologna, Italy) for supporting us with their valuable expertise.
Figure 5: PGPDT scaling on the CLX/1024 system: the KDDCUP-99 test problem.

References


Pai-Hsuen Chen, Rong en Fan, and Chih-Jen Lin. A study on SMO-type decomposition methods for support vector machines. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2005.


Rong-En Fan, Pai-Hsuen Chen, and Chih-Jen Lin. Working set selection using second order information for training SVM. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2005.


Chih Jen Lin. Linear convergence of a decomposition method for support vector machines. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2001b.


B. Murtagh and M. Saunders. MINOS 5.5 user’s guide. Technical report, Department of Operation Research, Stanford University, Stanford CA, 1998.


