Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Parallel Computing 29 (2003) 535–551

PARALLEL
COMPUTING

www.elsevier.com/locate/parco

A parallel solver for large quadratic programs in training support vector machines ☆

G. Zanghirati [a,*], L. Zanni [b]

[a] *Dipartimento di Matematica, Università di Ferrara, via Machiavelli 35, 44100 Ferrara, Italy*
[b] *Dipartimento di Matematica, Università di Modena e Reggio Emilia, via Campi 213/b, 41100 Modena, Italy*

Received 5 February 2002; received in revised form 28 June 2002

Dedicated to Professor Fernando Zironi

Abstract

This work is concerned with the solution of the convex quadratic programming problem arising in training the learning machines named *support vector machines*. The problem is subject to box constraints and to a single linear equality constraint; it is dense and, for many practical applications, it becomes a large-scale problem. Thus, approaches based on explicit storage of the matrix of the quadratic form are not practicable. Here we present an easily parallelizable approach based on a decomposition technique that splits the problem into a sequence of smaller quadratic programming subproblems. These subproblems are solved by a variable projection method that is well suited to a parallel implementation and is very effective in the case of Gaussian support vector machines. Performance results are presented on well known large-scale test problems, in scalar and parallel environments. The numerical results show that the approach is comparable on scalar machines with a widely used technique and can achieve good efficiency and scalability on a multiprocessor system.
© 2003 Elsevier Science B.V. All rights reserved.

Keywords: Support vector machines; Pattern recognition; Convex quadratic programs; Projection-type methods; Parallel computation

☆ This work was supported by the Italian Education, University and Research Ministry (grant 1999/9901102591_004) and by the Italian National Research Council (grant CNRC000E3F_008).

* Corresponding author. Tel.: +39-0532-974025; fax: +39-0532-247292.

E-mail addresses: g.zanghirati@unife.it (G. Zanghirati), zanniluca@unimo.it (L. Zanni).

0167-8191/03/$ - see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0167-8191(03)00021-8

1. Introduction

This work proposes a parallel solver for the large-scale quadratic programming (QP) problem arising in training the learning machines named support vector machines (SVMs) [3,4,30].

The main idea behind the pattern classification algorithm SVM is to separate two point classes of a training set,

$$D = \{(\boldsymbol{x}_i, y_i), \quad i = 1, \dots, N, \ \boldsymbol{x}_i \in \mathbb{R}^m, \ y_i \in \{-1, 1\}\},$$

with a surface that maximizes the margin between them. This separating surface is obtained by solving a convex quadratic program of the form

$$\begin{aligned} \min \quad & f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}} Q \boldsymbol{\alpha} - \sum_{i=1}^{N} \alpha_i \\ \text{s.t.} \quad & \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\alpha} = 0, \quad 0 \leqslant \alpha_j \leqslant C, \ j = 1, \dots, N, \end{aligned} \tag{1}$$

with $\boldsymbol{y} = [y_1, y_2, \dots, y_N]^{\mathrm{T}}$ and $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_N]^{\mathrm{T}}$. The entries Q_{ij} of the symmetric positive semidefinite matrix Q are defined as

$$Q_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j), \quad i, j = 1, 2, \dots, N,$$

where $K(\cdot,\cdot)$ denotes a kernel function depending on the type of the surface considered. Interesting choices for the kernel function are the polynomial kernel

$$K(\boldsymbol{s}, \boldsymbol{t}) = (1 + \boldsymbol{s}^{\mathrm{T}}\boldsymbol{t})^d$$

and the Gaussian kernel

$$K(\boldsymbol{s}, \boldsymbol{t}) = \mathrm{e}^{\frac{-\|\boldsymbol{s}-\boldsymbol{t}\|_2^2}{2\sigma^2}}, \quad \sigma \in \mathbb{R}.$$

The nonzero components in the solution of (1) correspond to the training examples that determine the separating surface; these special examples are called support vectors (SVs) and their number is usually much smaller than N. Since the matrix Q of the quadratic form is dense, being equal in size to the total number of training examples, the quadratic program (1) may be considered a challenging large-scale problem ($N \gg 10\,000$) in many real life applications of the SVMs.

The various approaches proposed in the last years to overcome the difficulties involved in this large and dense optimization problem fall into two main categories. The first category includes algorithms that exploit the special structure of the problem, while the second collects the techniques based on different formulations of the optimization problem that gives rise to the separating surface. The latter reformulations lead to more tractable optimization problems, but use criteria for determining the decision surface which, in some cases, are considerably different from that of the standard SVM [7,10,11,14–16]. Since the numerical results show a remarkable reduction in training time (with test set correctness statistically comparable to that of standard SVM classifiers), these approaches appear an important tool for very large data sets. Among the methods of the first category we recall the interior point method proposed in [5] for training linear SVMs and the decomposition techniques

[8,21,23]. The method in [5] is suitable for both the special definition of Q in the linear case ($Q_{ij} = y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$) and the simple structure of the SVM constraints. This approach, by using an appropriate implementation of the linear algebra and out-of-core computations, can handle massive problems in which the size of the training set is of the order of millions. On the other hand, the decomposition techniques are based on the idea of splitting the problem (1) into a sequence of smaller QP subproblems that can fit into the available memory (first proposed in [1]). The various techniques differ in the strategy employed for identifying the variables to update at each iteration and in the size chosen for the subproblems. In [23] the subproblems have size 2 and can be solved analytically, while in [8,21] the subproblem size is a parameter of the procedure and a numerical QP solver is required.

In this work, following the decomposition scheme named SVM$^{\text{light}}$ and proposed in [8], we develop a parallel solver for problem (1). In Section 2, we analyze the SVM$^{\text{light}}$ technique and focus on the choice of the subproblem size and on the inner QP solver, which are crucial questions for the effectiveness of the method. We introduce an iterative solver for the inner QP subproblems well suited to a parallel implementation. The solver is a variable projection method [26,27] that is very efficient in the case of Gaussian SVMs and able to exploit the structure of the constraints. By using this inner solver, the SVM$^{\text{light}}$ technique may be appropriately implemented to work efficiently with large QP subproblems and few decomposition steps. In Section 3, we describe how this version of the decomposition technique can be easily parallelized and we evaluate its effectiveness by solving several large-scale benchmark problems on a multiprocessor system.

2. Decomposition techniques

In order to describe our parallel implementation more clearly we need to recall in detail the main ideas underlying the decomposition techniques for problem (1).

At each step of the decomposition strategies proposed in [8,21,23], the variables α_i of (1) are split into two categories:

- the set $\mathcal{B}$ of *free* (or *basic*) variables,
- the set $\mathcal{N}$ of *fixed* (or *nonbasic*) variables.

The set $\mathcal{B}$ is usually referred to as the *working set*. Suppose we arrange the arrays $\boldsymbol{\alpha}$, $\boldsymbol{y}$ and Q with respect to $\mathcal{B}$ and $\mathcal{N}$:

$$\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha}_{\mathcal{B}} \\ \boldsymbol{\alpha}_{\mathcal{N}} \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_{\mathcal{B}} \\ \boldsymbol{y}_{\mathcal{N}} \end{bmatrix}, \quad Q = \begin{bmatrix} Q_{\mathcal{BB}} & Q_{\mathcal{BN}} \\ Q_{\mathcal{NB}} & Q_{\mathcal{NN}} \end{bmatrix}.$$

Given a generic $\bar{\boldsymbol{\alpha}} = [\bar{\boldsymbol{\alpha}}_{\mathcal{B}}^T, \bar{\boldsymbol{\alpha}}_{\mathcal{N}}^T]^T$, the idea behind the decomposition techniques consists in progressing toward the minimum of $f(\boldsymbol{\alpha})$ by substituting $\bar{\boldsymbol{\alpha}}_{\mathcal{B}}$ with the vector $\tilde{\boldsymbol{\alpha}}_{\mathcal{B}}$ obtained by solving (1) with respect only to the variables in the working set. Of course, in order to achieve a rapid decrease in the objective function, an appropriate choice of the working set is required. One of the main contributions in this respect is afforded by the effective updating rule, based on Zoutendijk's method, introduced by the SVM$^{\text{light}}$ algorithm of Joachims [8].

The SVM$^{\text{light}}$ decomposition technique can be stated in this way:

Step 1. Let $\boldsymbol{\alpha}^{(1)}$ be a feasible point for (1), let N_{sp} and N_c be two integer values such that $N \geqslant N_{\text{sp}} \geqslant N_c$; arbitrarily choose N_{sp} indices for the working set $\mathcal{B}$ and set $k = 1$.

Step 2. Compute the elements of $Q_{\mathcal{BB}}$, $\boldsymbol{q} = Q^{\text{T}}_{\mathcal{NB}}\boldsymbol{\alpha}^{(k)}_{\mathcal{N}} - [1, 1, \dots, 1]^{\text{T}}$ and $e = -\boldsymbol{y}^{\text{T}}_{\mathcal{N}}\boldsymbol{\alpha}^{(k)}_{\mathcal{N}}$.

Step 3. Solve the subproblem

$$\begin{aligned} &\min \quad g(\boldsymbol{\alpha}_{\mathcal{B}}) = \frac{1}{2}\boldsymbol{\alpha}^{\text{T}}_{\mathcal{B}}Q_{\mathcal{BB}}\boldsymbol{\alpha}_{\mathcal{B}} + \boldsymbol{q}^{\text{T}}\boldsymbol{\alpha}_{\mathcal{B}} \\ &\text{s.t.} \quad \boldsymbol{y}^{\text{T}}_{\mathcal{B}}\boldsymbol{\alpha}_{\mathcal{B}} = e, \quad 0 \leqslant \alpha_i \leqslant C, \quad \text{for } i \in \mathcal{B}, \end{aligned} \tag{2}$$

and let $\boldsymbol{\alpha}^{(k+1)}_{\mathcal{B}}$ denote an optimal solution. Set $\boldsymbol{\alpha}^{(k+1)} = \begin{bmatrix} \boldsymbol{\alpha}^{(k+1)}_{\mathcal{B}} \\ \boldsymbol{\alpha}^{(k)}_{\mathcal{N}} \end{bmatrix}$.

Step 4. Update the gradient

$$\nabla f(\boldsymbol{\alpha}^{(k+1)}) = \nabla f(\boldsymbol{\alpha}^{(k)}) + \begin{bmatrix} Q_{\mathcal{BB}} \\ Q_{\mathcal{NB}} \end{bmatrix}\left(\boldsymbol{\alpha}^{(k+1)}_{\mathcal{B}} - \boldsymbol{\alpha}^{(k)}_{\mathcal{B}}\right) \tag{3}$$

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

Step 5. Find the indices corresponding to the nonzero components of the solution of the following problem:

$$\begin{aligned} &\min \quad \nabla f(\boldsymbol{\alpha}^{(k+1)})^{\text{T}}\boldsymbol{d}, \\ &\text{s.t.} \quad \boldsymbol{y}^{\text{T}}\boldsymbol{d} = 0, \\ &\qquad d_i \geqslant 0 \quad \text{for } i \text{ such that } \alpha_i = 0, \\ &\qquad d_i \leqslant 0 \quad \text{for } i \text{ such that } \alpha_i = C, \\ &\qquad -1 \leqslant d_i \leqslant 1, \\ &\qquad \#\{d_i | d_i \neq 0\} \leqslant N_c. \end{aligned} \tag{4}$$

Update $\mathcal{B}$ to include these indices; set $k \leftarrow k + 1$ and go to step 2.

We refer to [8,12,13] for a discussion about the convergence properties of the scheme and about other important aspects, such as how to solve the nonexpensive linear program (4) and how to check the KKT conditions for this special QP problem. Here we concentrate on steps 2, 3 and 4, which require the main computational resources and on which is based the proposed parallel implementation. These steps involve kernel evaluations (for computing the elements of $Q_{\mathcal{BB}}$ and $Q_{\mathcal{NB}}$) that may be very expensive if the dimension of the input space is large and the training examples have many nonzero features. Furthermore, we have to solve the QP subproblem (2) of size N_{sp} at each iteration. Various efficient tricks are used in the implementation of Joachims to reduce the computational cost of these tasks. In particular, a *caching* strategy to avoid recomputation of elements of Q previously used and a *shrinking* strategy for reducing the size of the problem are implemented. For the size $N_{\rm sp}$ of the working set, very small values are suggested in [8]. As a result, the subproblem (2) can be efficiently solved by many QP packages, does not increase significantly the cost of each iteration and, in addition to the caching and shrinking strategies, reduces the total number of kernel evaluations required by the scheme. On the other hand, in general, small values for $N_{\rm sp}$ imply many iterations of the SVM$^{\rm light}$ technique. In short, the implementation proposed by Joachims is organized to produce better performance when it works with small values of $N_{\rm sp}$, that is, with many nonexpensive iterations.

In order to develop a parallel implementation of this decomposition scheme it is useful to analyze its behavior also for large values of $N_{\rm sp}$. In fact, in this case we expect few expensive iterations whose complexity may be reduced by facing in parallel their expensive tasks. To this end, the choice of the solver for the QP subproblems (2) is crucial. The solver must be efficient for this dense problem when the size is medium or large and well suited to parallel implementations. The robust solvers suggested in the machine learning literature [3,8,21], like MINOS [19] and LOQO [28,29], are not designed for the special characteristics of problem (2) and appear hardly parallelizable. In the case of SVMs with Gaussian kernels, an iterative solver suited to exploit both the structure of the constraints and the particular nature of the Hessian matrix is introduced in [31]. The method is the *variable projection method* (VPM) [26,27] with a special updating rule for its projection parameter, appropriately studied for the QP problem of this application.

The VPM for the subproblem (2) consists in the following steps: [1]

Step I. Let $S = \mathrm{diag}\{s_1, \dots, s_{N_{\rm sp}}\}$, $s_i > 0\ \forall i$, $\boldsymbol{z}^{(0)} \in \mathbb{R}^{N_{\rm sp}}$ arbitrary, $\rho_1 > 0$, $\tilde{\ell} \in \mathbb{N} \setminus \{0\}$, $\ell \leftarrow 1$.

Step II. Compute the unique solution $\bar{\boldsymbol{z}}^{(\ell)}$ of the subproblem

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{z}^{\rm T}\frac{S}{\rho_\ell}\boldsymbol{z} + \left(\boldsymbol{q} + \left(Q_{\mathcal{BB}} - \frac{S}{\rho_\ell}\right)\boldsymbol{z}^{(\ell-1)}\right)^{\rm T}\boldsymbol{z} \\ \text{s.t.} \quad & \boldsymbol{y}_{\mathcal{B}}^{\rm T}\boldsymbol{z} = e, \quad 0 \leqslant z_j \leqslant C, \ j = 1, \dots, N_{\rm sp}. \end{aligned} \tag{5}$$

Step III. If $\ell \neq 1$, compute the solution θ_ℓ of

$$\min_{\theta \in (0,1]} g(\boldsymbol{z}^{(\ell-1)} + \theta \boldsymbol{d}^{(\ell)}) \quad \text{where} \quad \boldsymbol{d}^{(\ell)} = \bar{\boldsymbol{z}}^{(\ell)} - \boldsymbol{z}^{(\ell-1)}$$

else $\theta_\ell = 1$.

Step IV. Compute $\boldsymbol{z}^{(\ell)} = \boldsymbol{z}^{(\ell-1)} + \theta_\ell \boldsymbol{d}^{(\ell)}$.

[1] Here, as usual, $\mathrm{diag}\{a_1, \dots, a_n\}$ denotes a diagonal matrix with entries a_i, $i = 1, \dots, n$ and $\mathrm{mod}(m, n)$ is the remainder of the integer ratio m/n.

Step V. Terminate if $\boldsymbol{z}^{(\ell)}$ satisfies a stopping criterion, otherwise update $\rho_{\ell+1}$ by the rule

$$\rho_{\ell+1} = \begin{cases} \rho_\ell & \text{if } \|Q_{\mathcal{BB}}\boldsymbol{d}^{(\ell)}\|^2 \leqslant \epsilon\|\boldsymbol{d}^{(\ell)}\|^2, \\ \begin{cases} \dfrac{\boldsymbol{d}^{(\ell)\mathrm{T}}Q_{\mathcal{BB}}\boldsymbol{d}^{(\ell)}}{\boldsymbol{d}^{(\ell)\mathrm{T}}Q_{\mathcal{BB}}S^{-1}Q_{\mathcal{BB}}\boldsymbol{d}^{(\ell)}} & \text{if } \mathrm{mod}(\ell,\tilde{\ell}) < \frac{\tilde{\ell}}{2}, \\ \dfrac{\boldsymbol{d}^{(\ell)\mathrm{T}}S\boldsymbol{d}^{(\ell)}}{\boldsymbol{d}^{(\ell)\mathrm{T}}Q_{\mathcal{BB}}\boldsymbol{d}^{(\ell)}} & \text{otherwise,} \end{cases} \end{cases}$$

where $\epsilon > 0$ is a prefixed small tolerance; then $\ell \leftarrow \ell + 1$ and go to step II.

Each iteration of this scheme essentially requires the matrix-vector product $Q_{\mathcal{BB}}\bar{\boldsymbol{z}}^{(\ell)}$ and the solution of the separable QP problem (5) with the same constraints as (2). The vector $Q_{\mathcal{BB}}\bar{\boldsymbol{z}}^{(\ell)}$ is necessary for computing θ_ℓ in step III, $\rho_{\ell+1}$ in step V and for updating the vector in which is stored $Q_{\mathcal{BB}}\boldsymbol{z}^{(\ell)}$:

$$\boldsymbol{t}^{(\ell)} \leftarrow Q_{\mathcal{BB}}\boldsymbol{z}^{(\ell)} = Q_{\mathcal{BB}}(\boldsymbol{z}^{(\ell-1)} + \theta_\ell\boldsymbol{d}^{(\ell)}) = \boldsymbol{t}^{(\ell-1)} + \theta_\ell(Q_{\mathcal{BB}}\bar{\boldsymbol{z}}^{(\ell)} - \boldsymbol{t}^{(\ell-1)}).$$

Because of the special structure of the constraints, for the solution of (5) very efficient algorithms are available, suitable for both scalar and parallel computation [2,20,22]. Currently, we are using the $O(N)$ algorithm proposed in [22]. Thus, since the matrix $Q_{\mathcal{BB}}$ is dense, the main computational cost of each iteration is due to the matrix-vector product $Q_{\mathcal{BB}}\bar{\boldsymbol{z}}^{(\ell)}$. However, when N_{sp} is large and the solution $\boldsymbol{\alpha}_{\mathcal{B}}^{(k+1)}$ of (2) has few nonzero components, this cost may be significantly reduced by exploiting the expected sparsity of $\bar{\boldsymbol{z}}^{(\ell)}$. Finally, as numerically shown in [31], the particular updating rule for the projection parameter ρ_ℓ implies a remarkable increase in the linear convergence rate of the scheme, when the Hessian matrix $Q_{\mathcal{BB}}$ derives from training SVMs with Gaussian kernels. Unfortunately, the proposed updating rule in not so effective in the case of polynomial kernels. The reader is referred to [31] for a detailed analysis of VPM performance in this kind of problems and to [25,26] for VPM behavior in more general QP problems.

By using the VPM as efficient inner solver, we can test the SVM$^{\text{light}}$ decomposition technique with large values of the subproblem size. We develop an implementation of the SVM$^{\text{light}}$ algorithm appropriately designed for large values of the parameter N_{sp} and with the further aim of obtaining an easily parallelizable scheme. For this reason, we use a very simple caching strategy suitable for implementations on distributed memory systems and avoid other sophisticated tricks like those proposed by Joachims. In practice, we fill the caching area with the columns of $Q_{\mathcal{NB}}$ involved in (3), that is, the columns corresponding to the nonzero components of $\left(\boldsymbol{\alpha}_{\mathcal{B}}^{(k+1)} - \boldsymbol{\alpha}_{\mathcal{B}}^{(k)}\right)$. This simple strategy is justified by the fact that, in the updating of the working set at the end of each decomposition iteration, a part of the nonzero variables of the current working set will be included in the new one. In detail, in our updating procedure we first include in the new working set the indices given by (4), then, to fill the set up to N_{sp} entries, we add the indices satisfying $0 < \alpha_j^{(k+1)} < C$, $j \in \mathcal{B}$. If these indices are not enough, we then add those such that

$\alpha_j^{(k+1)} = 0$, $j \in \mathcal{B}$, and, eventually, those satisfying $\alpha_j^{(k+1)} = C$, $j \in \mathcal{B}$. Of course, this procedure may not be optimal for all problems; sometimes, for example, we find convenient to exchange the last two criteria.

In the following numerical experiments we evaluate the behavior of this implementation named *variable projection decomposition technique* (VPDT). The VPDT algorithm is coded in standard C and the experiments are carried out on a workstation Compaq XP1000 at 667 MHz with 768 MB of RAM. We consider two real-world data sets: the MNIST database of handwritten digits from AT&T Research Labs [9] and the UCI Adult data set [18]. In the MNIST database the inputs are 784-dimensional nonbinary sparse vectors; the sparsity level of the inputs is 81% and the size of the database is 60 000. For these experiments we construct a reduced test problem of size 20 000 by considering the first 5000 inputs of the digit 8 and the first 15 000 of the other digits. The UCI Adult data set allows us to train a SVM to predict whether a household has an income greater than \$50 000. After appropriate discretization of the continuous attributes [23], the inputs are 123-dimensional binary sparse vectors with a sparsity level equal to 89%. We use the version of the data set with size 16101. For both the data sets we train a Gaussian SVM, with $C = 10$, $\sigma = 1800$ for the MNIST data set and with $C = 1$, $\sigma^2 = 10$ for the UCI Adult data set.

In Tables 1 and 2 we compare the results obtained by training the SVM with Joachims' package SVM$^{\text{light}}$ (version 3.50) and the VPDT algorithm. The methods use the same stopping rule, based on the fulfillment of the KKT conditions within a tolerance of 0.001, the same size for the caching area (500 MB) and sparse vector representation, crucial for optimizing kernel evaluations [24] and for reducing memory consumption. The VPM is used with S equal to the identity matrix, the null vector as $z^{(0)}$, $\rho_1 = 1$, $\tilde{\ell} = 6$ and a stopping rule based on the fulfillment of KKT conditions with the same tolerance used for the VPDT. In the SVM$^{\text{light}}$ package

Table 1
MNIST data set, $N = 20\,000$

	N_{sp}	N_c	Iter.	Sec.	SV	BSV	err($\tilde{\boldsymbol{\alpha}}$)	err($f(\tilde{\boldsymbol{\alpha}})$)
SVM$^{\text{light}}$	10*	10*	6719	562.8	2235	83	1.73e−3	8.08e−7
	4	2	11 449	529.4	2234	83	1.74e−3	7.09e−7
	8	4	6066	516.5	2235	83	1.64e−3	6.23e−7
	30	10	1935	525.7	2233	83	1.25e−3	3.93e−7
	42	14	1337	527.8	2234	83	1.13e−3	3.16e−7
	90	30	556	548.9	2235	83	9.64e−4	2.27e−7
	240	80	167	611.4	2235	83	7.42e−4	1.56e−7
VPDT	2000	400	12	672.1	2234	83	7.66e−4	1.76e−7
	2100	500	9	539.7	2234	83	6.70e−4	1.73e−7
	2200	700	7	504.4	2234	83	6.54e−4	1.68e−7
	2300	750	6	468.7	2235	83	9.11e−4	1.59e−7
	2400	800	6	495.1	2234	83	4.76e−4	1.93e−7
	2500	800	6	515.6	2233	83	8.41e−4	6.11e−8
	2600	700	6	520.4	2235	83	7.44e−4	2.26e−7

Table 2
UCI adult data set, $N = 16\,101$

	N_{sp}	N_c	Iter.	Sec.	SV	BSV	err($\tilde{\boldsymbol{\alpha}}$)	err($f(\tilde{\boldsymbol{\alpha}})$)
SVM$^{\text{light}}$	10*	10*	4175	120.9	5943	5349	2.28e−2	2.74e−7
	20	10	1877	114.0	5949	5363	3.37e−2	1.20e−7
	28	14	1403	114.3	5958	5365	3.42e−2	1.29e−7
	40	20	1072	115.7	5957	5353	3.86e−2	1.03e−7
	80	40	502	121.7	5963	5344	3.27e−2	7.33e−8
	160	80	240	133.1	5947	5350	3.37e−2	4.87e−8
	280	140	124	158.3	5968	5337	3.42e−2	2.14e−8
VPDT	600	250	46	121.7	5956	5343	3.65e−2	1.13e−8
	700	400	35	111.9	5960	5351	3.63e−2	3.70e−8
	800	500	31	125.2	5967	5358	3.17e−2	3.44e−8
	900	450	28	135.0	5964	5354	3.68e−2	2.96e−8
	1000	550	27	143.7	5962	5353	3.27e−2	1.95e−8
	1100	500	27	150.4	5965	5351	3.45e−2	3.66e−8
	1200	700	23	159.8	5960	5343	3.58e−2	1.24e−8

two solvers are available for the inner QP subproblems: the first is based on the method of Hildreth and D'Esopo, the second is a version of the primal-dual infeasible interior point method of Vanderbei [29], named pr_LOQO and implemented by Smola [28]. On these test problems, the performance of the solvers differ slightly for very small values of N_{sp}, while pr_LOQO appears more effective when N_{sp} increases. The reported results concern SVM$^{\text{light}}$ combined with pr_LOQO. In the Tables we show the number of iterations, the training time in seconds, the number of support vectors and bound support vectors (BSV) (i.e. support vectors with $\alpha_i = C$) corresponding to different values of the subproblem size. Furthermore, in order to evaluate the numerical accuracy of VPDT in comparison to SVM$^{\text{light}}$, besides the number of SVs and BSVs, we report the relative errors $\text{err}(\tilde{\boldsymbol{\alpha}}) = \|\tilde{\boldsymbol{\alpha}} - \boldsymbol{\alpha}^*\|_2 / \|\boldsymbol{\alpha}^*\|_2$ and $\text{err}(f(\tilde{\boldsymbol{\alpha}})) = |f(\tilde{\boldsymbol{\alpha}}) - f(\boldsymbol{\alpha}^*)| / |f(\boldsymbol{\alpha}^*)|$, where $\tilde{\boldsymbol{\alpha}}$ denotes the solution provided by the decomposition techniques with the above stopping rule and $\boldsymbol{\alpha}^*$ is the solution obtained by running SVM$^{\text{light}}$ with default settings, but tolerance 10^{-6} in the stopping rule. For SVM$^{\text{light}}$, N_{sp} and N_c indicate the values assigned to its optimization options `q` and `n`, respectively, while the marker "*" denotes the default parameter setting. For each value of N_{sp} we report the results corresponding to an empirical approximation of the optimal value of N_c, that is, the value of N_c that gave us the lowest computational time.

The first conclusion that can be drawn from these experiments concerns the behavior of the SVM$^{\text{light}}$. The results confirm that the best performances are obtained when SVM$^{\text{light}}$ works with QP subproblems of small size. On the other hand, the VPDT allows comparable performances and numerical accuracy with larger values of N_{sp} and very few iterations. This means that VPDT can be considered a strategy as efficient as SVM$^{\text{light}}$, but easily parallelizable. In fact, as discussed in the next section, the few iterations of the VPDT are suited to being performed on multiprocessor systems using a parallel version of the VPM and distributed kernel evaluations. Finally, we recall that VPDT does not benefit from an efficient caching strategy, but simply exploits the effectiveness of the VPM as inner QP solver. Of course, better performances can be expected by improving the caching strategy.

3. A parallel decomposition technique

In this section we will see how a parallel version of the VPDT works in practice. Our implementation is designed for a distributed memory system, is coded in standard C and uses standard MPI communication routines [17], hence it is easily portable on many multiprocessor systems.

The aim is to parallelize steps 2–4 of the decomposition technique, which include the heaviest computations. To this end, we distribute blockwise the rows of the matrices $Q_{\mathcal{BB}}$ and $Q_{\mathcal{BN}}$ among the available processors (Fig. 1). This data distribution is well suited to

- designing a parallel version of the VPM, since its computational core is given by the matrix-vector product $Q_{\mathcal{BB}}\bar{\mathbf{z}}^{(\ell)}$;
- exploiting, through the caching strategy, the large total memory usually available on multiprocessor systems.

Note that the solution of the separable QP subproblem (5) in the VPM could also be parallelized [20]; however, as observed in the previous section about the cost of each VPM iteration, this step is much less time-consuming than the matrix-vector product and this parallelization is not currently performed. As in the scalar version, the caching strategy is devoted to the management of $Q_{\mathcal{BN}}$. In detail, once all the other local and global arrays are stored, our parallel caching strategy uses the remaining memory of each processing element (PE) to store locally as much of the rows of $Q_{\mathcal{BN}}$

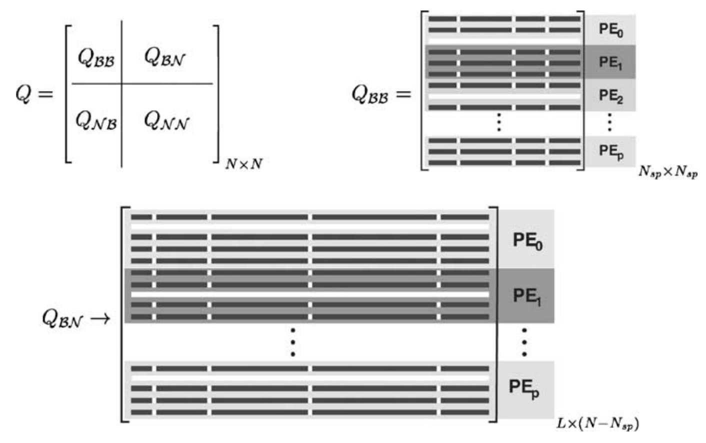
$Q = \begin{bmatrix} Q_{\mathcal{BB}} & Q_{\mathcal{BN}} \\ Q_{\mathcal{NB}} & Q_{\mathcal{NN}} \end{bmatrix}_{N\times N}$
$Q_{\mathcal{BB}} =$
PE$_0$
PE$_1$
PE$_2$
PE$_p$
$N_{sp}\times N_{sp}$
$Q_{\mathcal{BN}} \rightarrow$
$L\times(N-N_{sp})$

Fig. 1. Data distribution for the parallel caching strategy. Light lines/points show the current updated elements.

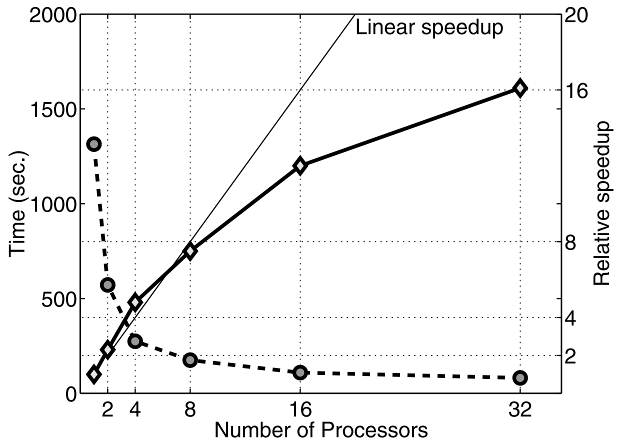

Fig. 2. Relative speedup on Cray T3E for MNIST database with $N = 20\,000$.

as possible, among those used by the PE in the current iteration. If one of these rows was already in the local memory of the PE, only the elements corresponding to the new indices of the current set $\mathcal{N}$ will be computed (light points in Fig. 1), otherwise entire computation of new entering rows will be required (light lines in Fig. 1). When the cache is full, the new rows enter by substituting the least-recently-used rows. Remark that the total number L of stored rows is both a memory-dependent and problem-dependent value.

Besides the matrices distribution, local copies of all the vectors involved in the matrix-vector products are maintained on each PE. In this way, the products $Q_{\mathcal{N}\mathcal{B}}^{\mathrm{T}}\boldsymbol{\alpha}_{\mathcal{N}}^{(k)}$ and $[Q_{\mathcal{B}\mathcal{B}}\ Q_{\mathcal{B}\mathcal{N}}]^{\mathrm{T}}(\boldsymbol{\alpha}_{\mathcal{B}}^{(k+1)} - \boldsymbol{\alpha}_{\mathcal{B}}^{(k)})$ in the decomposition technique and $Q_{\mathcal{B}\mathcal{B}}\bar{\mathbf{z}}^{(\ell)}$ in the VPM are computed in parallel in two phases. First, a local computation is performed by each PE to obtain the local part of the result; this phase does not need any synchronization. Second, a collective operation is performed to update the results on all the PEs. Of course, this phase involves implicit synchronizations.

We tested the code on the Cray T3E present at CINECA Supercomputing Center (Bologna, Italy): it is an MPP distributed memory system equipped with 256 DEC Alpha EV5 processing elements (PEs) at 600 MHz, grouped in two 128-PEs sets: in the first set there are the ones with 128 MB RAM, while in the other set there are PEs with 256 MB RAM. To evaluate the effectiveness of the proposed implementation we solve some test problems derived by the MNIST and the UCI Adult data sets previously described. The test problems are obtained by training Gaussian SVMs with the setting for the parameters C and σ used in Section 2. In the parallel VPDT the stopping rule consists again in verifying that the KKT conditions are fulfilled within 0.001.

Let's start with the results on the MNIST data set. The graphs in Figs. 2–4 present the behavior on three test problems of size 20 000, 40 000 and 60 000, respectively, generated as previously explained. In the graphs, the solid line shows the relative speedup, while the dashed one is for the overall solution time. The relative speedup

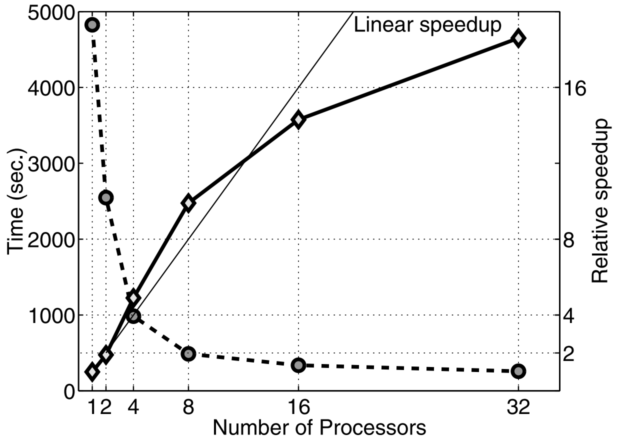

Fig. 3. Relative speedup on Cray T3E for MNIST database with $N = 40000$.

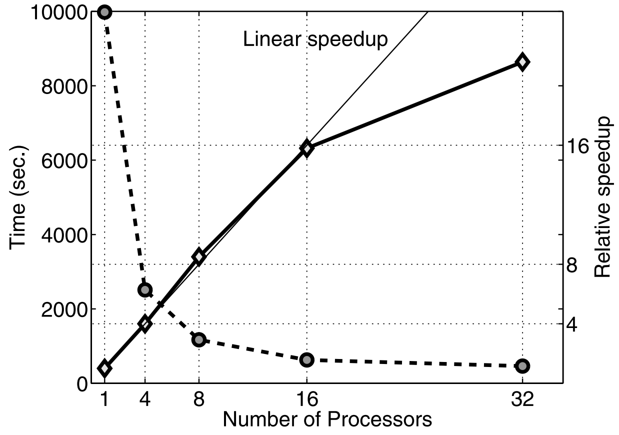

Fig. 4. Relative speedup on Cray T3E for MNIST database with $N = 60000$.

reported is computed as the ratio of the time needed by the program in a "sequential" setting of the parallel machine (T_s) to the time needed by the "same" program on p PEs (T_p):

$$sp_r(p) = \frac{T_s}{T_p}.$$

Here we note that, for a consistent comparison on the Cray T3E, the sequential setting is "simulated" by running the program on two processors, one of which does nothing but the initialization of the parallel environment: this is necessary because this machine has the so-called "production PEs", which are used to run parallel programs, and "command PEs", which are used to run pure scalar programs. However, the latter are also devoted to the external user interface, to the complete I/O management, and are loaded with compilers and debuggers, hence the timing on these PEs cannot be considered reliable.

In all three graphs, for each fixed number of processors we report the results corresponding to the "optimal" values (empirically determined) of the parameters N_{sp} and N_c, that is, the values that allow the available computing resources to be best exploited. These optimal values depend on the number of processors, therefore the parallel VPDT might require a different number of decomposition iterations and kernel evaluations in comparison with the serial VPDT. This is the reason for the superlinear speedup, which in turn demonstrates how well the presented implementation can exploit the available computational resources even with very few PEs.

Consider the smaller case first (Fig. 2). As mentioned, a very good behavior is shown for 4 and 8 PEs, while it becomes worse for a larger number of processors. To understand why this happens, recall that the aim of the decomposition technique is to reduce the problem size; when this size is not large enough, the suitable values for N_{sp} may be too small to allow an efficient use of many PEs. This important observation is confirmed by the graphs in Figs. 3 and 4, where it is clearly evident how, for increasing problem size, 16 PEs may also be very well exploited (we note that, on these test problems, the best performance of the parallel VPDT is obtained with values of N_{sp} around 2400 for $N = 20\,000$, 2800 for $N = 40\,000$, 3600 for $N = 60\,000$). The computational time shown by the sequential runs is seen to be quite high, which is due to two main factors: first, the single processing element of the Cray T3E has at most 256 MB RAM, which is quite a limited amount of memory for this kind of application and severely restricts the performance of our caching strategy; second, the DEC Alpha EV5 processor mounted on the machine is no longer one of the fastest available. For instance, the time taken to solve the test problems is about two thirds less on the more recent DEC Alpha EV6/7 at 667 MHz of a Compaq XP1000 with 768 MB RAM (see Table 1). Nevertheless, the behavior reported is relative, so that the conclusions are still reliable.

Fig. 5 reports the relative efficiency ($\mathrm{eff}_r(p) = sp_r(p)/p$), which explains how the proposed parallel decomposition technique is effective on a small number of processors and, on the other hand, how much the behavior corresponding to a large number of PEs improves when the problem size increases.

Moreover, let's consider the last graph (Fig. 6): here, for all three tested problems, we report the values of the Kuck's function, which for p processors is defined as

$$f_K(p) = \mathrm{eff}_r(p)sp_r(p) = \frac{sp_r^2(p)}{p}.$$

It is well known that $f_K(p)$ achieves its maximum in the optimal number of processors for a given problem. The graph summarizes all the previous observations and emphasizes that the parallel VPDT may be effective for solving very large problems on multiprocessor systems, since it appears well suited to exploit efficiently a large

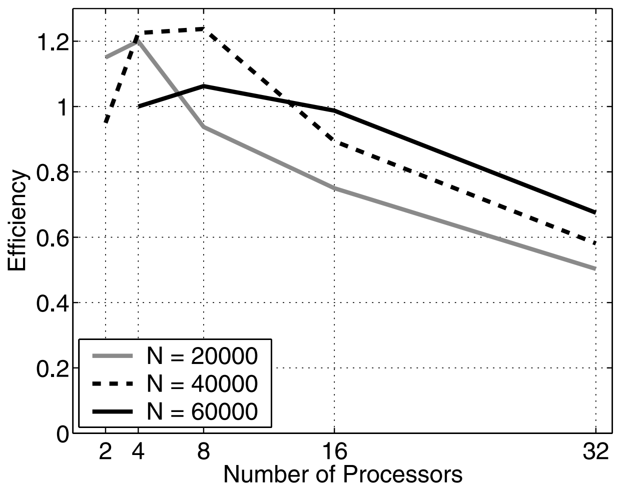

Fig. 5. Relative efficiency on Cray T3E for the three tests on the MNIST database.

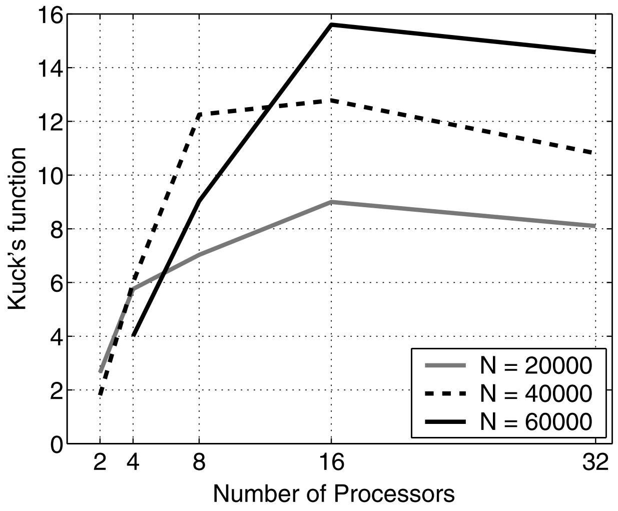

Fig. 6. Kuck's function for the three tests on the Cray T3E.

number of processors as well. Note, in fact, that for all three test problems the $f_K(p)$ maximum is obtained for 16 PEs but that, in the largest case, the maximum is close to the "optimal" one; we may therefore expect that for larger problems more than 16 PEs could be efficiently exploited.

To better analyze the scalability of the parallel VPDT with respect to increasing computational complexity, we evaluate the scaled speedup of the code on the considered test problems. Here we use the definition of scaled speedup given in [6]:

Table 3
Scaled speedup $S_s(\gamma, n, w)$ for the MNIST test problems

	Number of PEs		
	$n = 2$	$n = 4$	$n = 8$
$\gamma = 3$	2.3	2.0	1.8
$\gamma = 7$	5.2	3.8	2.7

$$S_s(\gamma, n, w) = \gamma \frac{T(n, w)}{T(\gamma n, \gamma w)}, \quad w > 0, \ n \geqslant 1, \ \gamma \geqslant 1, \tag{6}$$

where $T(n, w)$ is the running time required by a parallel code to solve a problem of complexity w on n processors, while $T(\gamma n, \gamma w)$ is the running time required by the program to solve a problem of complexity γw on γn processors. Named w the number of elementary computations for the problem of size $N = 20\,000$, the computational complexity for the problems of size $N = 40\,000$ and $60\,000$ is about $3w$ and $7w$, respectively. Thus, we report in Table 3 the scaled speedup (6) for the two cases $\gamma = 3$ (with running times for the problems of size $N = 20\,000$ and $40\,000$) and $\gamma = 7$ (with running times for the problems of size $N = 20\,000$ and $60\,000$).

We recall that a parallel code is scaling well when the scaled speedup is close to γ. Looking at Table 3, the parallel VPDT achieves good scaled speedups as far as the number of processors γn in S_s does not exceed a critical value that, as also highlighted by the Kuck's function, for these test problems is around 16.

Let us now consider the results on the UCI Adult data set. As shown in Tables 1 and 2, the test problems arising in this case are very different from the previous ones. Here, we have many SVs (about 37% of N) and many BSVs (about 90% of the number of SVs). This feature implies that a rapid convergence of the VPDT can be guaranteed only with very large QP subproblems, whose solution is expensive even for an effective solver like the VPM. Thus, to obtain the best performance with VPDT we are forced to work with suboptimal subproblem sizes and more decomposition iterations than in the case of MNIST data set. In this situation, the parallel VPDT can efficiently exploit only a small number of PEs. Table 5 reports the best results obtained on the largest version of the UCI Adult data set for different numbers of

Table 4
Numerics of Fig. 3 and number of support vectors

	PEs	N_{sp}	N_c	Iter.	Sec.	sp_r	SV	BSV
VPDT	1	2800	800	7	4825.6		2717	134
Parallel	2	2800	800	7	2547.9	1.9	2717	134
VPDT	4	2800	800	7	984.7	4.9	2717	134
	8	2800	800	7	487.5	9.9	2717	134
	16	2800	800	7	337.3	14.3	2717	134
	32	4000	1000	6	258.9	18.6	2720	133

Table 5
Relative speedup on Cray T3E for UCI Adult data set with $N = 32\,562$

	PEs	N_{sp}	N_c	Iter.	Sec.	sp_r	SV	BSV
VPDT	1	1300	750	40	2494.9		11 741	10 553
Parallel	2	1600	800	31	1358.5	1.8	11 766	10 553
VPDT	4	1600	800	31	856.9	2.9	11 766	10 553
	8	1600	800	31	559.3	4.5	11 766	10 553
	16	1600	800	31	412.2	6.1	11 766	10 553

PEs: as expected, we observe a good relative speedup only for 2 and 4 PEs. However, an appreciable time reduction is also achieved with 8 and 16 PEs.

Finally, we remark that in all the experiments the solution computed by the parallel VPDT satisfies the same stopping rule of the serial VPDT, hence it has a comparable numerical accuracy (the number of SVs obtained with the serial and parallel algorithms are reported in Tables 4 and 5).

Further developments about both the porting of the code and the performance analysis will be discussed in a future work.

4. Conclusions

In this paper we have proposed a parallel decomposition technique for solving the large quadratic program arising in training SVMs. The scheme is based on the decomposition strategy SVM^{light} of Joachims [8], which requires to solve a sequence of smaller QP subproblems. In the case of Gaussian SVMs, a very effective and easily parallelizable solver for these subproblems can be derived by the variable projection method introduced in [26,27]. By using this method as inner solver, we have developed an implementation that works efficiently with subproblems large enough to produce few iterations of the decomposition scheme. Conversely, in Joachims' approach, the best results are obtained by working in an opposite way, i.e. with very small QP subproblems and then many decomposition iterations.

The numerical experiments in training Gaussian SMVs on the MNIST and UCI Adult data sets show that our approach, named variable projection decomposition technique, has scalar performance comparable with that of the original SVM^{light} package. Nevertheless, our approach is better suited to a parallel implementation, since the expensive tasks of the few iterations can be easily performed in parallel. In fact, the large QP subproblems may be solved with a parallel version of the variable projection method, while the data updating phase, which requires expensive kernel evaluations, can be performed by distributing the computations among the available processors. This parallel decomposition technique proves to be well scalable and very efficient in the case of MNIST classification, where the solution has few nonzero components, i.e. there are few support vectors. In the case of UCI Adult classification, where many support vectors are involved, suboptimal performances are observed, since the subproblem size convenient for the decomposition technique is not so large as to allow a very effective use of the parallel resources.

To sum up, the parallel solver introduced in this work may be an useful tool for reducing the computational time in training Gaussian SVMs on distributed memory multiprocessor systems.

Acknowledgements

The authors are most grateful to Alessandro Verri for bringing the problem to their attention and for useful suggestions. The authors are also indebted to the referees for several remarks leading to improvements of the paper.

References

[1] B.E. Boser, I.M. Guyon, V.N. Vapnik, A training algorithm for optimal margin classifiers, in: D. Haussler (Ed.), Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, ACM Press, Pittsburgh, PA, 1992, pp. 144–152.
[2] P. Brucker, An $O(n)$ algorithm for quadratic knapsack problems, Oper. Res. Lett. 3 (1984) 163–166.
[3] C.J.C. Burges, A tutorial on support vector machines for pattern recognition, Data Mining and Knowledge Discovery 2 (1998) 121–167.
[4] C. Cortes, V.N. Vapnik, Support vector network, Machine Learning 20 (1995) 1–25.
[5] M.C. Ferris, T.S. Munson, Interior Point Methods for Massive Support Vector Machines, Technical Report 00-05, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.
[6] H.P. Flatt, K. Kennedy, Performance of parallel processors, Parallel Comput. 12 (1989) 1–20.
[7] G. Fung, O.L. Mangasarian, Proximal Support Vector Machines Classifiers, Technical Report 01-02, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2001.
[8] T. Joachims, Making large-scale SVM learning practical, in: B. Schölkopf, C.J.C. Burges, A. Smola (Eds.), Advances in Kernel Methods—Support Vector Learning, MIT Press, Cambridge, MA, 1998.
[9] Y. LeCun, MNIST Handwritten Digit Database, available at www.research.att.com/~yann/ocr/mnist.
[10] Y.J. Lee, O.L. Mangasarian, SSVM: A Smooth Support Vector Machines, Technical Report 99-03, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1999.
[11] Y.J. Lee, O.L. Mangasarian, RSVM: Reduced Support Vector Machines, Technical Report 00-07, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.
[12] C.J. Lin, On the convergence of the decomposition method for support vector machines, IEEE Trans. Neural Networks 12 (2001) 1288–1298.
[13] C.J. Lin, Linear Convergence of a Decomposition Method for Support Vector Machines, Technical Report, Department of Computer Science and Information Engineering, National Taiwan University, Taiwan, 2002.
[14] O.L. Mangasarian, D.R. Musicant, Successive overrelaxation for support vector machines, IEEE Trans. Neural Networks 10 (1999) 1032–1037.
[15] O.L. Mangasarian, D.R. Musicant, Active Support Vector Machine Classification, Technical Report 00-04, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.

[16] O.L. Mangasarian, D.R. Musicant, Lagrangian Support Vector Machines, Technical Report 00-06, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.
[17] Message Passing Interface Forum, MPI 1.2: A Message-Passing Interface standard (version 1.2), Technical Report, 1995. Available at www.mpi-forum.org.
[18] P.M. Murphy, D.W. Aha, UCI Repository of Machine Learning Databases, 1992. Available at www.ics.uci.edu/~mlearn/MLRepository.html.
[19] B. Murtagh, M. Saunders, MINOS 5.4 User's Guide, System Optimization Laboratory, Stanford University, 1995.
[20] S.N. Nielsen, S.A. Zenios, Massively parallel algorithms for single constrained convex programs, ORSA J. Comput. 4 (1992) 166–181.
[21] E. Osuna, R. Freund, F. Girosi, An improved training algorithm for support vector machines, in: J. Principe, L. Giles, N. Morgan, E. Wilson, (Eds.), Proceedings of the IEEE Workshop on Neural Networks for Signal Processing, Amelia Island, FL, 1997, pp. 276–285.
[22] P.M. Pardalos, N. Kovoor, An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds, Math. Prog. 46 (1990) 321–328.
[23] J.C. Platt, Fast training of support vector machines using sequential minimal optimization, in: B. Schölkopf, C. Burges, A. Smola (Eds.), Advances in Kernel Methods—Support Vector Learning, MIT Press, Cambridge, MA, 1998.
[24] J.C. Platt, Using analytic QP and sparseness to speed training of support vector machines, in: M.S. Kearns, S.A. Solla, D.A. Cohn (Eds.), Advances in Neural Information Processing Systems, vol. 11, MIT Press, Cambridge, Mass, 1999.
[25] V. Ruggiero, L. Zanni, On the efficiency of splitting and projection methods for large strictly convex quadratic programs, in: G. Di Pillo, F. Giannessi (Eds.), Nonlinear Optimization and Related Topics, Applied Optimization, vol. 36, Kluwer Academic Publishers, 1999, pp. 401–413.
[26] V. Ruggiero, L. Zanni, A modified projection algorithm for large strictly convex quadratic programs, J. Optim. Theory Appl. 104 (2000) 281–299.
[27] V. Ruggiero, L. Zanni, Variable projection methods for large convex quadratic programs, in: D. Trigiante (Ed.), Recent Trends in Numerical Analysis, vol. 3, Nova Science Publishers, 2000, pp. 299–313.
[28] A.J. Smola, pr_LOQO, www.kernel-machines.org/code/prloqo.tar.gz.
[29] R.J. Vanderbei, LOQO: An Interior Point Code for Quadratic Programming, Technical Report SOR-94-15 Revised, Princeton University, 1998.
[30] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, New York, 1995.
[31] G. Zanghirati, L. Zanni, The Variable Projection Methods for Large Quadratic Programs in Training Support Vector Machines, Technical Report, Department of Mathematics, University of Modena and Reggio Emilia, Italy, 2003.