Large Quadratic Programs in Training Gaussian Support Vector Machines*

T. Serafini[1], G. Zanghirati[2] and L. Zanni[1]

[1]Dipartimento di Matematica, Università di Modena e Reggio Emilia, Modena, Italy.
[2]Dipartimento di Matematica, Università di Ferrara, Ferrara, Italy.

Sommario

We consider the numerical solution of the large convex quadratic program arising in training the learning machines named *support vector machines*. Since the matrix of the quadratic form is dense and generally large, solution approaches based on explicit storage of this matrix are not practicable. Well known strategies for this quadratic program are based on decomposition techniques that split the problem into a sequence of smaller quadratic programming subproblems. For the solution of these subproblems we present an iterative projection-type method suited for the structure of the constraints and very effective in case of Gaussian support vector machines. We develop an appropriate decomposition technique designed to exploit the high performance of the proposed inner solver on medium or large subproblems. Numerical experiments on large-scale benchmark problems allow to compare this approach with another widely used decomposition technique. Finally, a parallel extension of the proposed strategy is described.

Sunto

In questo lavoro si considera la risoluzione numerica del problema di programmazione quadratica convessa di grandi dimensioni che interviene nell'addestramento della metodologia di apprendimento automatico nota come *support vector machine*. Poiché la matrice della forma quadratica è densa e generalmente di grandi dimensioni, per

*This work was supported by the Italian Education, University and Research Ministry (grants FIRB2001/RBAU01JYPN, FIRB2001/RBAU01877P) and by the Italian National Research Council (grant CNRC000E3F_008).

la risoluzione del problema non si possono utilizzare schemi che richiedono l'intera matrice disponibile in memoria. Per questo motivo si utilizzano tecniche di decomposizione che separano il problema in una successione di sottoproblemi quadratici più piccoli. Per la risoluzione di tali sottoproblemi si presenta un metodo iterativo di tipo proiezione, particolarmente adatto alla struttura dei vincoli e molto efficace nel caso di macchine a vettori di supporto con nucleo gaussiano. Si propone una tecnica di decomposizione progettata per sfruttare le buone prestazioni del metodo di proiezione su sottoproblemi di dimensioni medio-grandi. Si riportano i risultati di una sperimentazione numerica su problemi di grandi dimensioni, in cui si confronta l'approccio proposto con un'altra tecnica di decomposizione nota in letteratura. Infine, si descrive l'estensione parallela della strategia qui proposta.

Key words: support vector machines, pattern recognition, convex quadratic programs, decomposition techniques, projection-type methods.

AMS Subject Classification: 65K05, 90C20, 68T05.

1 Introduction

This work is concerned with the numerical solution of the large quadratic programming (QP) problem arising in training the learning machines named *support vector machines* (SVMs) [4, 5, 31].

Given a training set of labelled examples

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

the SVM algorithm performs pattern recognition by finding a classifier $F : R^m \to \{-1, 1\}$, of the form

$$F(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i^* y_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b^*\right), \tag{1}$$

where $K(\cdot, \cdot)$ denotes a kernel function defining the classifier type. In case of linear SVMs, the kernel function is linear and (1) becomes

$$F(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}^T \boldsymbol{x}_i + b^*\right),$$

while nonlinear SVMs can be obtained by choosing special nonlinear kernel functions such as polynomial kernels

$$K(\boldsymbol{s},\boldsymbol{t}) = (1+\boldsymbol{s}^T\boldsymbol{t})^d, \qquad \boldsymbol{s},\boldsymbol{t}\in\mathbb{R}^m,$$

or Gaussian kernels

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

The coefficients α_i^* in (1) are the solution of the following QP problem

$$\begin{array}{rl} \min & f(\boldsymbol{\alpha}) = \dfrac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \displaystyle\sum_{i=1}^N \alpha_i \\ \text{sub. to} & \boldsymbol{y}^T\boldsymbol{\alpha} = 0, \\ & 0\le\alpha_i\le C, \qquad i=1,\ldots,N, \end{array} \tag{2}$$

where $\boldsymbol{y} = (y_1, y_2, \ldots, y_N)^T$, $\boldsymbol{\alpha} = (\alpha_1,\alpha_2,\ldots,\alpha_N)^T$, $C\in\mathbb{R}$ and the entries Q_{ij} of Q are defined as

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

The training examples corresponding to nonzero α_i^* are the only examples used in the classifier definition; they are called *support vectors* (SVs) and their number is usually much smaller than N. The support vector having $\alpha_i^* = C$ are often called *bound support vectors* (BSVs).

Since the threshold b^* is easily derived from the α_i^*, we may conclude that the training of an SVM consists in solving the convex QP problem (2). The size of this problem is equal to the number of training examples and, consequently, in many real-world SVMs applications we have to solve a very large QP problem. In these cases, since Q is dense, the main trouble in solving (2) is that standard QP solvers based on explicit storage of the quadratic form matrix cannot be used. Among the various approaches proposed to overcome this trouble we may recognize two main classes. The first class includes algorithms that exploit the special structure of (2), while the second collects the techniques based on different formulations of the optimization problem behind the classifier. These last reformulations lead to more tractable optimization problems, but the criteria they use to determine the classifier are, in some cases, considerably different with respect to the one of the standard SVMs [7, 11, 12, 15, 16, 17]. Since the numerical results show a remarkable gain 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 class we recall the interior point method proposed in [6] for training linear SVMs and the decomposition techniques [2, 9, 21, 22, 24]. The method in [6] 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 with training set size larger than a few millions. On the other hand, the decomposition techniques are based on the idea of splitting the problem (2) into a sequence of smaller QP subproblems. These techniques differ from each other in the strategy employed to identify the variables to update at each iteration and in the chosen subproblems size. In particular, the decomposition techniques proposed in [2, 21] involve subproblems whose size scales with the number of support vectors; hence, they may not be able to handle large-scale problems. This disadvantage is avoided in the schemes of [9, 22, 24], where the subproblems size is independent on the expected number of support vectors.

In this work, we present an iterative solver for the decomposition techniques inner QP subproblems and we show that it can be an useful tool for improving their performance. In Section 2, we recall one of the most used decomposition strategy: the SVM^{light} algorithm proposed by Joachims in [9]. In Section 3, we introduce the iterative solver for the inner QP subproblems. It is a variable projection method [26, 27] very efficient in the case of Gaussian SVMs and suited for the constraints structure of this application. A new implementation of the SVM^{light} decomposition strategy, designed to exploit the effectiveness of this inner solver, is described in Section 4. Finally, in Section 5 we compare the behavior of our implementation with Joachims' SVM^{light} package on well known large-scale test problems.

2 Decomposition Techniques

Here we briefly recall the main ideas behind the decomposition techniques for problem (2).

At each step of the decomposition strategies proposed in [9, 22, 24], the variables α_i in (2) are splitted 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 to arrange the arrays $\boldsymbol{\alpha}$, $\boldsymbol{y}$ and Q with respect to $\mathcal{B}$ and $\mathcal{N}$:

$$\boldsymbol{\alpha} = \begin{pmatrix} \boldsymbol{\alpha}_{\mathcal{B}} \\ \boldsymbol{\alpha}_{\mathcal{N}} \end{pmatrix}, \qquad \boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_{\mathcal{B}} \\ \boldsymbol{y}_{\mathcal{N}} \end{pmatrix}, \qquad Q = \begin{bmatrix} Q_{\mathcal{B}\mathcal{B}} & Q_{\mathcal{B}\mathcal{N}} \\ Q_{\mathcal{N}\mathcal{B}} & Q_{\mathcal{N}\mathcal{N}} \end{bmatrix}.$$

Given a generic $\bar{\boldsymbol{\alpha}} = \left[\bar{\boldsymbol{\alpha}}_{\mathcal{B}}^T, \bar{\boldsymbol{\alpha}}_{\mathcal{N}}^T\right]^T$, the idea behind the decomposition techniques consists in make a progress toward the minimum of $f(\boldsymbol{\alpha})$ by substituting $\bar{\boldsymbol{\alpha}}_{\mathcal{B}}$ with the vector $\widetilde{\boldsymbol{\alpha}}_{\mathcal{B}}$ obtained by solving (2) with respect to the working set variables only. Different implementations of this idea may be found in literature. In [24], $\widetilde{\boldsymbol{\alpha}}_{\mathcal{B}}$ is analytically computed from a 2-elements working set selected by special choice heuristics. In [9, 22] the size of $\mathcal{B}$ is a constant procedure parameter and a numerical QP solver is used for the subproblems solution. Of course, the convergence rate of these techniques is strictly dependent on the variables chosen for updating $\mathcal{B}$ at each iteration. While the procedure in [22] updates the working set by simply including some variables violating the KKT conditions, in the SVM^{light} technique a more promising updating is performed by following an idea similar to the Zoutendijk' steepest feasible descent approach. Since both the discussion and the numerical experiments that follow will be concerned with this last decomposition strategy, we summarize the main steps of the SVM^{light} algorithm:

Step 1. Let $\boldsymbol{\alpha}^{(1)}$ be a feasible point for (2), let N_{sp} and N_c be two integer values such that* $N \geq N_{sp} \geq N_c$; arbitrarily choose N_{sp} indices for the working set $\mathcal{B}$ and set $k = 1$.

Step 2. Compute the elements of $Q_{\mathcal{B}\mathcal{B}}$, $\boldsymbol{q} = Q_{\mathcal{N}\mathcal{B}}^T \boldsymbol{\alpha}_{\mathcal{N}}^{(k)} - (1, 1, \ldots, 1)^T$ and $e = -\boldsymbol{y}_{\mathcal{N}}^T \boldsymbol{\alpha}_{\mathcal{N}}^{(k)}$.

Step 3. Solve the subproblem

$$\begin{aligned} \min \quad & g(\boldsymbol{\alpha}_{\mathcal{B}}) = \frac{1}{2}\boldsymbol{\alpha}_{\mathcal{B}}^T Q_{\mathcal{B}\mathcal{B}} \boldsymbol{\alpha}_{\mathcal{B}} + \boldsymbol{q}^T \boldsymbol{\alpha}_{\mathcal{B}} \\ \text{sub. to} \quad & \boldsymbol{y}_{\mathcal{B}}^T \boldsymbol{\alpha}_{\mathcal{B}} = e, \\ & 0 \leq \alpha_i \leq C, \qquad \text{for } i \in \mathcal{B}, \end{aligned} \tag{3}$$

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

*N_c is the maximum number of new variables entering the working set in each iteration. A value $N_c < N_{sp}$ is generally advisable to prevent zigzagging.

Step 4. Update the gradient

$$\nabla f(\boldsymbol{\alpha}^{(k+1)}) = \nabla f(\boldsymbol{\alpha}^{(k)}) + \begin{bmatrix} Q_{\mathcal{B}\mathcal{B}} \\ Q_{\mathcal{N}\mathcal{B}} \end{bmatrix} \left(\boldsymbol{\alpha}_{\mathcal{B}}^{(k+1)} - \boldsymbol{\alpha}_{\mathcal{B}}^{(k)}\right) \tag{4}$$

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

$$\begin{array}{rl} \min & \nabla f(\boldsymbol{\alpha}^{(k+1)})^T \boldsymbol{d} \\ \text{sub. to} & \boldsymbol{y}^T \boldsymbol{d} = 0, \\ & d_i \geq 0 \quad \text{for } i \text{ such that } \alpha_i = 0, \\ & d_i \leq 0 \quad \text{for } i \text{ such that } \alpha_i = C, \\ & -1 \leq d_i \leq 1, \\ & \#\{d_i \mid d_i \neq 0\} \leq N_c. \end{array} \tag{5}$$

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

We refer to [9, 13, 14] for a discussion about the convergence properties of the scheme and about other important aspects, such as how to solve the nonexpensive linear program (5) and how to check the KKT conditions for this special QP problem. Here we are interested in discussing how an effective implementation of this strategy may be carried out. In each iteration, the main computational resources are employed for computing the elements of $[\ Q_{\mathcal{B}\mathcal{B}} \quad Q_{\mathcal{N}\mathcal{B}}^T\ ]^T$ and for solving the QP subproblem (3). In fact, the kernel evaluations required to update $Q_{\mathcal{B}\mathcal{B}}$ and $Q_{\mathcal{N}\mathcal{B}}$ may be very expensive if the dimension of the input space is large and the training examples have many nonzero features. Furthermore, when the size N_{sp} is not very small, also the numerical solution of (3) may significantly increase the cost of each iteration. Various efficient tricks are used in the Joachims implementation to reduce the computational cost of these tasks. In particular, a *caching* strategy to avoid the recomputation of previously used elements of Q and a *shrinking* strategy to reduce the problem size are implemented. For the working set size N_{sp}, very small values are suggested in [9] (in the SVM^{light} package the default value is 10). This makes the subproblem (3) efficiently solvable by many QP packages, does not significantly increase 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 of N_{sp} imply many iterations of the SVM^{light} technique. The subproblems QP solvers suggested in the SVM^{light} package are the Hildreth and D'Esopo method and a version of the primal-dual infeasible interior point method of Vanderbei [30], named

pr_LOQO and implemented by Smola [29]. Of course, other robust general solvers like, for example, MINOS [19] may be used (see [21]). Nevertheless, special methods designed for the characteristics of problem (3) and effective also for medium to large N_{sp} values are crucial: this allows to improve the efficiency of general QP-based decomposition schemes and, furthermore, to develop new implementations of the SVM^{light} strategy.

3 The Variable Projection Method as Inner QP Solver

In case of SVMs with Gaussian kernel, we introduce an iterative solver suited for exploiting both the structure of the constraints and the particular Hessian matrix nature. The method is the *variable projection method* (VPM) [26, 27] with a diagonal scaling matrix and a special updating rule for the projection parameter, appropriately studied for the QP problem of this application.

The VPM steps for subproblem (3) are the following:

Step 1. Let $S = diag\{s_1, \ldots, s_{N_{sp}}\}$, $s_i > 0 \ \forall i$, $\boldsymbol{z}^{(0)} \in \mathbb{R}^{N_{sp}}$ arbitrary, $\rho_1 > 0$, $\ell = 1$.

Step 2. Compute the unique solution $\bar{\boldsymbol{z}}^{(\ell)}$ of the subproblem

$$\begin{array}{rl} \min & \frac{1}{2}\boldsymbol{z}^T \frac{S}{\rho_\ell}\boldsymbol{z} + \left(\boldsymbol{q} + \left(Q_{\mathcal{B}\mathcal{B}} - \frac{S}{\rho_\ell}\right)\boldsymbol{z}^{(\ell-1)}\right)^T \boldsymbol{z} \\ \text{sub. to} & \boldsymbol{y}_{\mathcal{B}}^T \boldsymbol{z} = e, \\ & 0 \le z_j \le C, \qquad j = 1, \ldots, N_{sp}. \end{array} \tag{6}$$

Step 3. If $\ell \neq 1$, compute the solution θ_ℓ of

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

else $\theta_\ell = 1$.

Step 4. Compute $\boldsymbol{z}^{(\ell)} = \boldsymbol{z}^{(\ell-1)} + \theta_\ell \boldsymbol{d}^{(\ell)}$.

Step 5. Terminate if $\boldsymbol{z}^{(\ell)}$ satisfies a stopping criterion, otherwise update $\rho_{\ell+1}$ by the rule

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

where $\epsilon > 0$ is a prefixed small tolerance; then $\ell \leftarrow \ell + 1$ and go to step 2.

The R-linear convergence of VPMs for convex quadratic programs is proved in [27]. When VPM is applied to the subproblems (3), the requirement for its convergence is only that the sequence $\{\rho_\ell\}$ is bounded by positive constants. Proceeding as in [28], it's easy to see that the projection parameter satisfies

$$\min\left(\rho_1, \frac{\lambda_{\min}(S)}{\lambda_{\max}(Q_{\mathcal{BB}})}\right) \leq \rho_{\ell+1} \leq \max\left(\rho_1, \frac{\lambda_{\max}(S)\lambda_{\max}(Q_{\mathcal{BB}})}{\epsilon}\right),$$

where $\lambda_{\min}(\cdot)$ and $\lambda_{\max}(\cdot)$ denote the minimum and the maximum eigenvalue of a matrix, respectively. An example of a parameters setting suited for the QP problems arising in training SVMs will be given later.

The most expensive operations in each iteration of this scheme are the matrix-vector product $Q_{\mathcal{BB}}\bar{\boldsymbol{z}}^{(\ell)}$ and the solution of the QP subproblem (6), having the same constraints of (3). The vector $Q_{\mathcal{BB}}\bar{\boldsymbol{z}}^{(\ell)}$ is necessary for computing θ_ℓ in step 3, $\rho_{\ell+1}$ in step 5 and for updating the vector storing $Q_{\mathcal{BB}}\boldsymbol{z}^{(\ell)}$:

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

About the subproblem (6), we observe that by selecting a diagonal scaling matrix S we make (6) a separable QP problem (refer to [1] for an introduction to scaled projection methods). Because of the special constraints structure, for the solution of this separable problem very efficient algorithms are available, suitable for both scalar and parallel computation [3, 20, 23]. Currently we are using the $O(N_{sp})$ algorithm proposed in [23]. 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 (3) has few nonzero components, this cost may be significantly reduced by exploiting the expected sparsity of $\bar{\boldsymbol{z}}^{(\ell)}$: if N_{nz} is the number of nonzero components of $\bar{\boldsymbol{z}}^{(\ell)}$, the product $Q_{\mathcal{BB}}\bar{\boldsymbol{z}}^{(\ell)}$ can be performed with $O(N_{sp}N_{nz})$ operations. Finally, the particular updating rule for the projection parameter ρ_ℓ implies a remarkable improvement in the linear convergence rate of the scheme when the Hessian matrix $Q_{\mathcal{BB}}$ comes from training SVM with Gaussian kernels. Unfortunately, this updating rule is not equally effective in the case of polynomial kernels. The reader may refer to [32] for further discussion of VPM performance on this kind of problems and to [8] for the VPM behavior on more general

Tabella 1: test problems from the MNIST data set.

	N	iter.	sec.	SV	BSV
pr_LOQO	200	13	0.15	76	0
	400	14	1.2	120	0
	800	15	10.0	176	0
	1600	16	111.5	239	0
VPM	200	57	0.02	76	0
	400	82	0.1	120	0
	800	124	0.3	176	0
	1600	232	1.7	238	0

QP problems. Here, in order to show that this method is a promising inner solver for decomposition techniques, we report the results of a comparison with the pr_LOQO solver on some small to medium problems of the form (2) (note that, between the two solvers suggested in the SVM^{light} package, pr_LOQO appears the most effective when the problem size increases).

Our test problems are obtained by training Gaussian SVMs on the handwritten digits MNIST database from AT&T Research Labs [10] and on the UCI Adult data set [18]. In the MNIST database the inputs are 784-dimensional non-binary sparse vectors; the sparsity level is 81% and the database size is 60000. For these experiments we construct reduced test problems of size $N = 200, 400, 800, 1600$, by considering the first $N/2$ inputs of the digits 8 and 9. The UCI Adult data set we consider allows to train a SVM to predict whether a household has an income greater than \$50000. After appropriate discretization of the continuous attributes [24], the inputs are 123-dimensional binary sparse vectors with a sparsity level equal to 89%. We consider the versions of the data set with size $N = 1605, 2265, 3185$. We use $C = 10$, $\sigma = 1800$ for the MNIST data set and $C = 1$, $\sigma^2 = 10$ for the UCI Adult data set.

The results in tables 1 and 2 are obtained on a Compaq XP1000 workstation at 667MHz with 1GB of RAM, running standard C code. In VPM, the stopping rule consists in the fulfillment of the KKT conditions within a tolerance of 0.001 (we follow [9] for the computation of the equality constraint multiplier). In general, a higher accuracy does not significantly improve the SVM performance [9, 24]. The VPM parameters are set as follows: S is equal to the identity matrix, $\rho_1 = 1$, $\epsilon = 10^{-16}$, $\tilde{\ell} = 6$ and $\boldsymbol{z}^{(0)} = \mathbf{0}$.

Tabella 2: test problems from the UCI Adult data set.

	N	iter.	sec.	SV	BSV
pr_LOQO	1605	15	131.6	691	584
	2265	15	383.9	1011	847
	3185	15	1081.4	1300	1109
VPM	1605	136	2.3	691	584
	2265	171	5.8	1011	847
	3185	237	14.6	1299	1113

The pr_LOQO solver was run with $sigfig\_max = 8$ because lower values gave classification accuracies too worse than VPM.

In tables 1 and 2, "N" denotes the problem size, "iter." is the number of iterations and "sec." the seconds required by the solvers. Furthermore, SV and BSV denote the number of support vectors ($0 < \alpha_i^* < C$) and bound support vectors ($\alpha_i^* = C$), respectively. From the tables we may observe the higher effectiveness of VPM, especially when the problem size increases. In particular, VPM allows the solution of medium size problems ($N > 1000$) in few seconds. Thus, this method may be an useful inner QP solver to improve the performances of decomposition techniques for large-scale SVMs. Furthermore, we emphasize that the method is easily parallelizable, since each iteration consists essentially in a matrix-vector product; hence, new parallel decomposition schemes can be based on VPM [32].

Remark. Since the previous experiments are aimed to compare the two solvers, the test problems are solved without decomposition. Of course, this may not be the best way to proceed; as an example, if $N_{sp} = 20$ and $N_c = 10$ the problem sized 3185 in table 2 is solved in 4.0 seconds by the SVM^{light} package combined with pr_LOQO.

4 A VPM-based Decomposition Technique

From the previous discussion it could be devised that a simple way to improve the performance of the SVM^{light} package is to introduce VPM as inner solver and to use decompositions with larger subproblems. This is not generally true because, as previously explained, this package is specifically designed to give the best performance when small size subproblems are used in the decomposition. In particular, the computation of N_{sp}

rows of the Hessian matrix is required at each iteration (see [9]) and, consequently, when N_{sp} is not sufficiently small the additional computational cost for the larger number of kernel evaluations degrades the performance. This disadvantage is partially reduced by the caching strategy, but it is enough to vanish the benefits, in terms of number of iterations, coming from the use of large N_{sp} values.

These reasons motivate our implementation of the SVM^{light} strategy, more suited than Joachims' package to handle decompositions based on large subproblems and thus able to exploit the high VPM performance. The main feature of our implementation is related to the matrix-vector products involving the submatrix $Q_{\mathcal{NB}}$. At each iteration, instead to compute the whole array as suggested in [9], the expected sparsity of the vectors involved in the products is exploited to compute only the strictly needed elements of $Q_{\mathcal{NB}}$. In practice, we fill the caching area with the $Q_{\mathcal{NB}}$ columns involved in (4), that is those columns corresponding to the nonzero components of $(\boldsymbol{\alpha}_{\mathcal{B}}^{(k+1)} - \boldsymbol{\alpha}_{\mathcal{B}}^{(k)})$. The rational for this simple strategy is the following: in the working set updating at the end of each decomposition iteration, some of the current working set nonzero variables will be included in the new one. Hence, our updating procedure first includes in the new working set the indices given by (5), then, to fill the set up to N_{sp} entries, it adds the indices satisfying $0 < \alpha_j^{(k+1)} < C$, $j \in \mathcal{B}$. If these indices are not enough, it adds 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 we find convenient to exchange the last two criteria. When the size of the subproblems is sufficiently large these simple tricks produce an appreciable saving in the number of kernel evaluations and, combined with an efficient inner solver like VPM, allow good performance.

Furthermore, since this VPM-based decomposition technique can work with large N_{sp} values, it is well suited for an efficient implementation on multiprocessor systems [32]. In fact, for large N_{sp} we expect few expensive iterations, which can be easily faced in parallel by solving the QP subproblems with a parallel version of VPM and by performing distributed kernel evaluations. We show some results of the parallel approach in the next section.

5 Numerical Experiments on Large Problems

In this section we compare on some large-scale problems the Joachims' package with our SVM^{light} implementation, named *variable projection decomposition technique* (VPDT).

As in section 3, the test problems comes from Gaussian SVMs trained on the cited MNIST and UCI Adult data sets, with the same C and σ as before. We consider the three UCI Adult data sets sized 16101, 22697, 32562. On the other hand, by training a classifier for digit 8, we generate from MNIST two test problems sized 40000 and 60000: for the former we consider the first 5000 inputs of the digit 8 and the first 35000 of the other digits, while for the latter we used the whole database. All the experiments are carried out on the Compaq XP1000 workstation previously described and standard C code.

Tables 3–5 report the results obtained by running the SVM^{light} package and the VPDT program with the same stopping rule, based on the fulfillment of the KKT condition within 10^{-3}. Both codes use sparse vector representation, which is crucial to optimize kernel evaluations [25] and to reduce memory consumption. All those tables show the number of decomposition procedure iterations (iter.), the training time in seconds (sec.), the number of support vectors (SV) and the number of bound support vectors (BSV).

The results in table 3 are obtained with a 350MB caching area and different N_{sp}, N_c values. We observe that the SVM^{light} package obtains its best performance for a very small value of the subproblem size ($N_{sp} = 8$), while for increasing N_{sp} values each iteration becomes too expensive and its effectiveness decreases. In these cases, using VPM as inner QP solver can improve the performances. However, since in this setting the QP subproblem solution is a cheap task of each iteration, the improvement due to VPM is not enough to compensate for the increased training time.

On the other hand, VPDT shows the best behavior for large N_{sp} values. In particular, as previously explained, when N_{sp} is close to (or larger than) the support vectors number, this decomposition scheme requires very few expensive iterations. This, combined with the high VPM performances, gives training times lower than those of the SVM^{light} package.

The same conclusions hold for the MNIST test problem of size $N = 60000$. For this problem, table 4 reports the results obtained by varying the caching area size (the SVM^{light} inner solver used is always pr_LOQO). The two implementations benefit in a similar way from increasing caching space.

Tabella 3: results on the MNIST test problem of size $N = 40000$.

	N_{sp}	N_c	iter.	sec.	SV	BSV	inner solver
SVM^{light}	8	4	7724	1710.6	2714	135	pr_LOQO
	40	10	2111	1790.2	2714	135	pr_LOQO
	40	20	1596	1848.6	2716	135	pr_LOQO
	100	20	865	1828.7	2713	135	pr_LOQO
	100	50	575	2138.3	2717	135	pr_LOQO
	200	20	685	2059.9	2714	135	pr_LOQO
	200	100	250	2624.8	2717	135	pr_LOQO
	200	20	716	1915.6	2714	135	VPM
	200	100	257	2511.3	2716	135	VPM
VPDT	2000	500	25	2551.0	2715	135	VPM
	2000	600	21	2385.5	2716	135	VPM
	2000	700	20	2343.1	2714	135	VPM
	2600	600	8	1036.1	2714	135	VPM
	2600	800	8	1150.2	2715	135	VPM
	2600	1000	7	1043.7	2714	135	VPM
	3000	800	7	1139.4	2713	136	VPM
	3000	1000	7	1197.8	2715	135	VPM
	3000	1200	6	1053.7	2715	136	VPM

Table 5 shows the behavior of the two packages on the UCI Adult data set. For each test problem, we report the results corresponding to the values of the parameters N_{sp} and N_c producing the best performance. A 150MB caching area is used. In these tests, the kernel evaluations are less expensive than in the case of the MNIST database (due to the lower dimension of the input space and to the higher sparsity of the examples), but there are much more support vectors (about 36% of N). This last feature prevents VPDT from working with subproblems of size close to the number of support vectors, thus requiring more decomposition iterations. However, considerable improvements are still observed with respect to the SVM^{light} package.

We remarked that the proposed VPM-based approach is well suited for parallel implementation on distributed memory multiprocessor systems, since the few expensive iterations required can be efficiently faced in parallel. In fact, recall that steps 2, 3 and 4 of the decomposition technique involve the heaviest computations (which are essentially due

Tabella 4: results on the MNIST test problem of size $N = 60000$.

	caching area in MB	iter.	sec.	SV	BSV
SVM^{light} $N_{sp} = 8$ $N_c = 4$	50	9699	4405.6	3153	158
	150	9527	4063.6	3151	158
	250	9430	3764.9	3153	159
	350	9334	3434.6	3155	160
	450	9397	3247.0	3154	160
VPDT $N_{sp} = 3100$ $N_c = 1200$	50	7	3602.4	3156	159
	150	7	3370.1	3156	159
	250	7	2972.4	3154	159
	350	7	2607.3	3156	159
	450	7	2333.6	3156	159

Tabella 5: results on test problems from the UCI Adult data set.

	N	N_{sp}	N_c	iter.	sec.	SV	BSV	inner solver
SVM^{light}	16101	20	10	1857	138.2	5949	5359	pr_LOQO
	22697	20	10	3002	337.4	8346	7495	pr_LOQO
	32562	20	10	4457	742.9	11699	10615	pr_LOQO
VPDT	16101	700	400	34	94.0	5959	5357	VPM
	22697	900	500	40	241.4	8377	7482	VPM
	32562	1300	850	39	593.3	11767	10558	VPM

to matrix-vector products) and that the VPM computational core is the product $Q_{\mathcal{BB}}\bar{\boldsymbol{z}}^{(\ell)}$. Hence, a suitable data distribution of $Q_{\mathcal{BB}}$ and $Q_{\mathcal{BN}}$ rows among the available processors allows to design a parallel version of VPM and to exploit, through the caching strategy, the large total memory usually available on multiprocessor systems. The solution of the separable QP subproblem (6) in VPM could also be parallelized [20], but this step is much less time-consuming than the matrix-vector product. In [32] a parallel version of the VPDT is coded in standard C with MPI 1.2 communication routines and tested on a Cray T3E MPP system with 256 DEC Alpha EV5 processors (PEs) at 600 MHz, equipped with 256 MB of RAM each.

In order to give an idea of the effectiveness of the proposed parallel approach, in table 6 we report the results obtained on the largest MNIST test problem ($N = 60000$) for

Tabella 6: performances of parallel VPDT on CRAY T3E for the MNIST data set with $N = 60000$.

	PEs	N_{sp}	N_c	iter.	sec.	SV	BSV	sp_r	eff_r
VPDT	1	3200	800	8	9979.1	3159	158		
	4	3200	800	8	2511.1	3159	158	4.0	1.000
Parallel	8	3200	800	8	1170.0	3159	158	8.5	1.063
VPDT	16	4000	1200	6	629.1	3161	159	15.9	0.994
	32	4000	1200	6	461.7	3161	159	21.6	0.675

different number of processors. In column sp_r we show the relative speedup of parallel VPDT, defined as the ratio of the time T_s needed by the program in a sequential setting of the parallel machine to the time T_p needed by the same program on p PEs, that is to say $sp_r(p) = T_s/T_p$. Column eff_r shows the relative efficiency $eff_r = sp_r(p)/p$.

Empirically determined "optimal" values was used for parameters N_{sp} and N_c, which allowed the best exploitation of the available computing resources. These optimal values and the storage resources depend on the number of processors; hence, parallel VPDT might behaves differently with respect to serial VPDT (as far as number of decomposition iterations and kernel evaluations are concerned). This is the reason for the superlinear speedup.

The results clearly show how the parallel approach can benefit from large subproblem size and how well the presented implementation scales. It exploits very well the available computational resources, even with very few PEs. The performance becomes suboptimal for 32 PEs, since the problem size is no longer large enough to fully exploit the machine power, as confirmed by the relative efficiency value. However, this allows to expect very good performance on larger (even huge) test problems.

We refer the reader to [32] for a deeper analysis of parallel VPDT and a wider numerical experimentation.

Conclusions

In this work we considered decomposition techniques for solving the large quadratic program arising in training SVMs. These decomposition schemes require to solve a sequence of smaller QP subproblems. For these subproblems, we proposed an iterative projection-

type method well suited to exploit the constraints structure and very effective in case of Gaussian SVMs. Furthermore, this solver is easily parallelizable. By using this method as inner solver, we developed a new implementation of the decomposition strategy proposed by Joachims in [9]. Our implementation is appropriately designed to exploit the inner solver effectiveness, by working with sufficiently large subproblems and few expensive decomposition iterations. Conversely, the Joachims' package gets its best results with very small subproblems, leading to many nonexpensive decomposition iterations. The two implementations are compared on large-scale test problems arising from the MNIST and the UCI Adult data sets. Our implementation allows a remarkable improvement of the decomposition technique performances on both the data sets. Moreover, the proposed approach is well suited for an implementation on distributed memory multiprocessor systems, since its few expensive decomposition iterations can be efficiently faced in parallel. Some numerical experiments on large-scale test problems are presented, to assess the effectiveness of this parallel approach.

Riferimenti bibliografici

[1] Bertsekas D.P.; *Nonlinear Programming*, Athena Scientific, Belmont Mass., 1999.

[2] Boser B.E., Guyon I.M., Vapnik V.; *A Training Algorithm for Optimal Margin Classifiers*, Proceedings of the Fifth Annual Workshop on Computational Learning Theory (Haussler D. editor), ACM Press, Pittsburg, PA, 1992, 144–152.

[3] Brucker P.; *An $O(n)$ Algorithm for Quadratic Knapsack Problems*, Oper. Res. Let. 3, 1984, 163–166.

[4] Burges C.J.C.; *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery 2(2), 1998, 121–167.

[5] Cortes C., Vapnik V.N.; *Support Vector Network*, Machine Learning 20, 1995, 1–25.

[6] Ferris M.C., Munson T.S.; *Interior Point Methods for Massive Support Vector Machines*, Data Mining Institute Technical Report 00-05, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.

[7] Fung G., Mangasarian O.L.; *Proximal Support Vector Machines Classifiers*, Technical Report 01-02, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2001.

[8] Galligani E., Ruggiero V., Zanni L.; *Projection-Type Methods for Large Convex Quadratic Programs: Theory and Computational Experience*, Monograph 5, Scientific Research Program "Numerical Analysis: Methods and Mathematical Software, University of Ferrara, Ferrara, Italy, 2000.

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

[10] LeCun Y.; *MNIST Handwritten Digit Database*. Available at http://yann.lecun.com/exdb/mnist.

[11] Lee Y.J., Mangasarian O.L.; *SSVM: A Smooth Support Vector Machines*, Technical Report 99-03, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1999.

[12] Lee Y.J., Mangasarian O.L.; *RSVM: Reduced Support Vector Machines*, Technical Report 00-07, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.

[13] Lin C.J.; *On the Convergence of the Decomposition Method for Support Vector Machines*, IEEE Transactions on Neural Networks 12(6), 2001, 1288–1298.

[14] Lin C.J.; *Linear Convergence of a Decomposition Method for Support Vector Machines*, Technical Report, Department of Computer Science and Information Engineering, National Taiwan University, 2002.

[15] Mangasarian O.L., Musicant D.R.; *Successive Overrelaxation for Support Vector Machines*, IEEE Transaction on Neural Networks 10, 1999, 1032–1037.

[16] Mangasarian O.L., Musicant D.R.; *Active Support Vector Machine Classification*, Technical Report 00-04, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.

[17] Mangasarian O.L., Musicant D.R.; *Lagrangian Support Vector Machines*, Technical Report 00-06, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.

[18] Murphy P.M., Aha D.W.; *UCI Repository of Machine Learning Databases*, 1992. Availabe at www.ics.uci.edu/∼mlearn/MLRepository.html.

[19] Murtagh B., Saunders M.; *MINOS 5.4 User's Guide*, System Optimization Laboratory, Stanford University, 1995.

[20] Nielsen S.N., Zenios S.A.; *Massively Parallel Algorithms for Single Constrained Convex Programs*, ORSA Journal on Computing 4, 1992, 166–181.

[21] Osuna E., Freund R., Girosi F.; *Training Support Vector Machines: an Application to Face Detection*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Puerto Rico, 1997, 130–136.

[22] Osuna E., Freund R., Girosi F.; *An Improved Training Algorithm for Support Vector Machines*, Proceedings of the IEEE Workshop on Neural Networks for Signal Processing (Principe J., Giles L., Morgan N. and Wilson E., eds.), Amelia Island, FL, 1997, 276–285.

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

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

[25] Platt J.C.; *Using Analytic QP and Sparseness to Speed Training of Support Vector Machines*, Advances in Neural Information Processing Systems 11 (Kearns M.S., Solla S.A. and Cohn D.A., eds.), MIT Press, Cambridge, Mass., 1999.

[26] Ruggiero V., Zanni L.; *A Modified Projection Algorithm for Large Strictly Convex Quadratic Programs*, J. Optim. Theory Appl. 104(2), 2000, 281–299.

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

[28] Ruggiero V., Zanni L.; *An Overview on Projection-Type Methods for Convex Large-Scale Quadratic Programs*, Equilibrium Problems: Nonsmooth Optimization and Variational Inequality Models (Giannessi F. et al., eds.), Nonconvex Optimization and Its Applications 58, Kluwer Academic Publ. 2001, 269–300.

[29] Smola A.J.; *pr_LOQO optimizer*. Available at www.kernel-machines.org/code/prloqo.tar.gz.

[30] Vanderbei R.; *LOQO: An Interior Point Code for Quadratic Programming*, Technical Report SOR 94-15, Princeton University, 1994.

[31] Vapnik V.N.; *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.

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