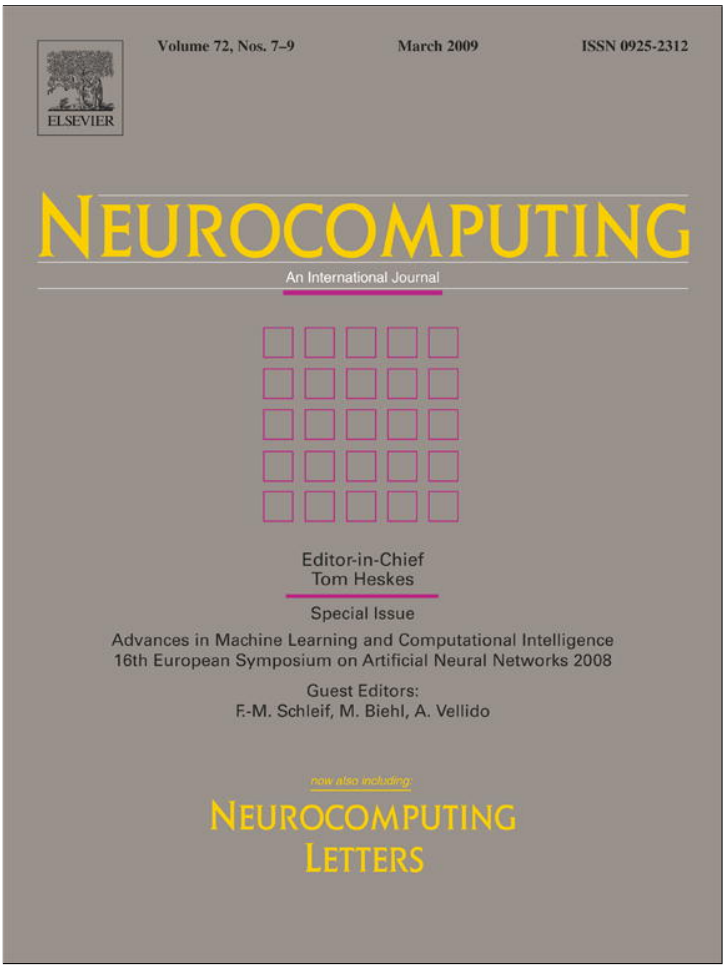


Neurocomputing 72 (2009) 1398–1406

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Cycle-breaking acceleration of SVM training ☆

Álvaro Barbero, Jorge López, José R. Dorronsoro *

Dpto. de Ingeniería Informática and Instituto de Ingeniería del Conocimiento, Universidad Autónoma de Madrid, 28049 Madrid, Spain

ARTICLE INFO

Available online 8 January 2009

Keywords:
Support vector machines
SMO
MDM algorithm
Cycles

ABSTRACT

Fast SVM training is an important goal for which many proposals have been given in the literature. In this work we will study from a geometrical point of view the presence, in both the Mitchell–Demyanov–Malozemov (MDM) algorithm and Platt's Sequential Minimal Optimization, of training cycles, that is, the repeated selection of some concrete updating patterns. We shall see how to take advantage of these cycles by partially collapsing them in a single updating vector that gives better minimizing directions. We shall numerically illustrate the resulting procedure, showing that it can lead to substantial savings in the number of iterations and kernel operations for both algorithms.

1. Introduction

Given a sample $\mathcal{S} = \{(X_i, y_i) : i = 1, \ldots, N\}$ with $y_i = \pm 1$, the standard formulation of support vector machines (SVMs) for linearly separable problems seeks [1,2] to maximize the margin of a separating hyperplane by solving the problem

$$\min \tfrac{1}{2}\|W\|^2 \quad \text{subject to } y_i(W \cdot X_i + b) \geqslant 1, \quad i = 1, \ldots, N. \tag{1}$$

In practice, however, the problem actually solved is the simpler dual problem of minimizing

$$\Theta(\mathbf{A}) = \frac{1}{2} \sum_{1 \leqslant i,j \leqslant N} \alpha_i \alpha_j y_i y_j X_i \cdot X_j - \sum_{i=1}^{N} \alpha_i$$
$$\text{subject to} \quad \alpha_i \geqslant 0, \quad 1 \leqslant i \leqslant N, \quad \sum_{i=1}^{N} \alpha_i y_i = 0, \tag{2}$$

where we write $\mathbf{A}$ for the multiplier vector $(\alpha_1, \ldots, \alpha_N)$. The optimal weight W^o can be then written in the so-called dual form as $W^o = \sum \alpha_i^o y_i X_i$ and patterns for which the optimal multipliers verify $\alpha_i^o > 0$ are called support vectors (SVs). The previous discussion assumes that the sample classes are linearly separable, something not likely in practice. This is usually overcome by working in a kernel setting and allowing margin slacks $\xi_i \geqslant 0$, $1 \leqslant i \leqslant N$, having now constraints of the form $y_i(W \cdot X_i + b) \geqslant 1 - \xi_i$, and adding to the weight norm a penalty term of the form $C \sum \xi_i^k$, with k typically being 1 or 2. When $k = 2$, the theory for linearly separable problems applies almost directly by simply considering extended vectors and kernels [3]. The main change when $k = 1$ is an extra constraint $\alpha_i \leqslant C$ that appears in the dual problem.

SVM training can be quite costly, as it scales at least quadratically with respect to the training sample size, and fast methods to build SVMs have received a wide attention in the literature. Many of them are derived (see for instance [4–6]) from Platt's SMO [7] or Joachims' SVMLight [8] algorithms. Both can be characterized as decomposition methods that at each step work only with a small set of updating vectors (two in the case of SMO and an even number $2q$ for SVMLight) and, in fact, if properly set, SMO coincides with SVMLight when $q = 1$. Moreover, operational improvements can be applied to them, such as Platt's types I and II updates or shrinking.

In this work, that extends previous one in [9], we will introduce cycle breaking, another potential improvement in SVM training, which is motivated by the study of algorithms that try to build SVMs from a geometric perspective. More precisely, their goal is to solve the so-called nearest point problem (NPP; see [10]) of finding the nearest points W_+^* and W_-^* of the convex hulls $C(\mathcal{S}_\pm)$ of the positive $\mathcal{S}_+ = \{X_i : y_i = 1\}$ and negative $\mathcal{S}_- = \{X_i : y_i = -1\}$ sample subsets. The maximum margin hyperplane is then $W^* = W_+^* - W_-^*$ and the optimal margin is given by $\|W^*\|/2$.

NPP can be simplified if we consider homogeneous SVMs, that is, classifiers of the form $\text{sign}(W \cdot X)$, where we drop the bias term b. The price for this is somewhat less powerful final classifiers, although good accuracies can still be obtained if we consider extended vectors $X' = (X, 1)$ and weights $W' = (W, w_0)$ and, therefore, classifiers of the form $\text{sign}(W' \cdot X') = \text{sign}(W \cdot X + w_0)$; while we will work with such extended patterns and weights, we will simply use the X and W notation from now on. In homogeneous SVM training we want to minimize $\|W\|^2$ subject to $y_i W \cdot X_i \geqslant 1$ and the new dual problem still seeks to minimize

$$\Theta(\mathbf{A}) = \frac{1}{2} \sum_{1 \leqslant i,j \leqslant N} \alpha_i \alpha_j y_i y_j X_i \cdot X_j - \sum_{i=1}^{N} \alpha_i,$$

☆ With partial support of Spain's TIN 2004-07676 and TIN 2007-66862. The first author is kindly supported by FPU-MEC Grant reference AP2006-02285 and the second one by FPU-MICINN Grant reference AP2007-00142.

* Corresponding author.
E-mail address: jose.dorronsoro@uam.es (J.R. Dorronsoro).

0925-2312/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2008.12.014

but the constraint $\sum \alpha_i y_i = 0$ no longer applies and we are left just with $\alpha_i \geqslant 0$. It can be then shown [11] that minimizing $\Theta(\mathbf{A})$ is equivalent to solving the Minimum Norm Problem (MNP) of finding the vector W^* in the convex hull $C(\tilde{\mathscr{S}})$ of the set $\tilde{\mathscr{S}} = \{y_i X_i : i = 1, \ldots, N\}$ with the smallest norm. That is, MNP wants to solve

$$\min \frac{1}{2} \sum_{1 \leqslant i,j \leqslant N} \alpha_i \alpha_j y_i y_j X_i \cdot X_j \quad \text{subject to } \alpha_i \geqslant 0, \ \sum_{i=1}^{N} \alpha_i = 1;$$

notice that now $\sum_{i,j} \alpha_i \alpha_j y_i y_j X_i \cdot X_j$ is just the norm of a $W = \sum_i \alpha_i y_i X_i \in C(\tilde{\mathscr{S}})$.

MNP is a much older problem than SVM training and two methods to solve it have been adapted to SVM construction, the Gilbert–Schlesinger–Kozinec (GSK) [11–13] and the Mitchell–Demyanov–Malozemov (MDM) [13,14] algorithms. The GSK algorithm uses a properly chosen, single pattern $y_L X_L$ to update the current weight vector W to a new one $W' = (1-\lambda)W + \lambda y_L X_L$. The new coefficients are thus

$$\alpha_i' = (1-\lambda)\alpha_i + \lambda \delta_{iL}, \tag{3}$$

where $\delta_{pq} = 1$ if $p = q$ and 0 otherwise. GSK is known to be a slow algorithm for MNP. This is, first, a consequence of the coefficient update (3), that implies that GSK may need many iterations to get rid of a wrong initial choice of a possible SV X_k. If this vector is not selected anymore as an updating pattern, its coefficient will decrease from its initial value α_k^0 as $\alpha_k^T = \alpha_k^0 \prod_1^T (1-\lambda_t)$. Since λ_t usually tends to 0 as $O(1/t)$ [12], we will have $\alpha_k^T = O(1/T)$. But another reason for the slow convergence of GSK is the presence of "cycles", that is, pairs of vectors X_i, X_j that are repeatedly selected as consecutive updating vectors in the GSK iterations. This is shown for instance in Fig. 1, (left), where GSK has been started from the uppermost vertex, while the minimum norm vector is clearly the middle point on the triangle's base. This causes the GSK updates to alternatively select the lower left and right vertices, zigzagging towards the minimum norm vector. On the other hand, it is also clear from the figure that the training evolution suggests better descent directions than the ones provided by GSK.

As we shall see, the MDM algorithm for MNP is not affected in the triangle problem by the just described situation. The algorithm, that we will briefly describe in Section 2, iteratively updates a weight vector W to a new $W' = W' + \lambda(y_L X_L - y_U X_U)$ with the L and U indices given by Eq. (4). Writing $W = \sum \alpha_j y_j X_j$, the corresponding α updates are

$$\alpha_L' = \alpha_L + \lambda, \quad \alpha_U' = \alpha_U - \lambda, \quad \alpha_i' = \alpha_i \quad \forall i \neq L, U.$$

In particular, if $\lambda = \alpha_U$, $\alpha_U' = 0$. Thus, the MDM algorithm can remove previous wrong SV choices; in particular, if applied to the problem in Fig. 1, it will get rid of the wrong initial choice of the uppermost vector in its second iteration and converge to the minimum norm vector right afterwards. In general, MDM is a much faster algorithm than GSK and allows for MNP solving (and SVM training) with a higher precision. However, MDM is still not free of cycle-related problems that can retard its convergence (see Fig. 2). Moreover, it can be shown that, when applied to general, non-homogeneous SVM training, MDM essentially coincides [15] with a simplification of the so-called Modification 2 of the SMO algorithm, a particularly efficient implementation of SMO proposed in [4] (see also [6]). In this simplification both updating vectors would be forced to be chosen within the same class. As a consequence, one may expect that cycles can also appear in SMO training and that avoiding the subsequent zigzags can also result in a faster convergence.

These questions are addressed in this paper. More precisely, in Section 2 we will briefly give a unified and very simple derivation of the MDM algorithm for MNP and of Keerthi et al.'s [4] Modification 2, as given in. These derivations will facilitate the cycle discussion that we will consider in Section 3, where we will analyze first the presence of cycles for MDM and propose a simple geometric weight update to avoid them and to achieve a faster training. We will discuss afterwards how to apply a similar procedure to linear penalty SMO. We will numerically illustrate the resulting methods in Section 4, showing that significant training savings can be achieved for both algorithms even if we take into account the higher computational cost of the cycle-breaking updates. The paper will end with a short discussion and conclusions section.

2. A unified approach to the MDM and SMO algorithms

We shall give here a presentation of the MDM and SMO algorithms from a common viewpoint suited to our discussion in the next section of their accelerated counterparts.

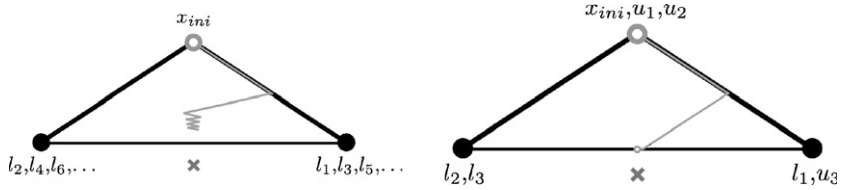


Fig. 1. Left: effect of a bad SV choice in the GSK algorithm. Right: the MDM algorithm is unaffected by the bad initial choice.

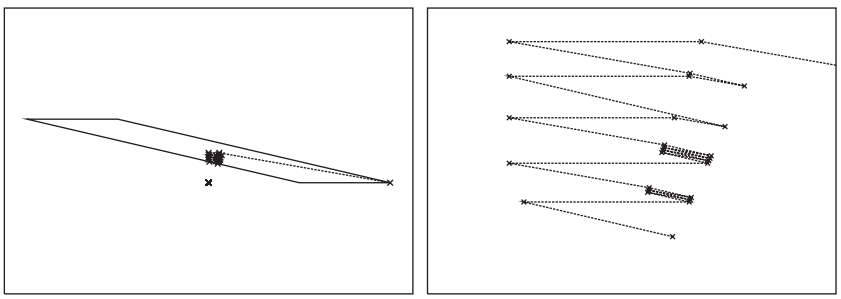


Fig. 2. The MDM algorithm may require many iterations even in simple problems. However (right) they may have a cyclical structure.

2.1. The MDM algorithm for MNP

The paper by Mitchell et al. [14], where the MDM algorithm for MNP was first proposed, suggests some heuristic reasons for its choice of updating vectors X_L, X_U. We will propose here an alternative, somewhat simpler motivation for this choice, that can also be applied to SMO. Recall that the MDM W updates have the form

$$W' = W + \lambda(y_L X_L - y_U X_U) = W + \lambda Z_{L,U},$$

where we write $Z_{L,U} = y_L X_L - y_U X_U$. Then $\|W'\|^2$ becomes a function Φ of λ, namely

$$\Phi(\lambda) = \|W'\|^2 = \|W\|^2 + 2\lambda W \cdot Z_{L,U} + \lambda^2 \|Z_{L,U}\|^2,$$

and it follows that $\Phi'(0) = 2W \cdot Z_{L,U}$; in particular, $Z_{L,U}$ defines a descent direction for Φ at W provided that $W \cdot Z_{L,U} < 0$. Moreover, since $\Phi'(\lambda) = 2W \cdot Z_{L,U} + 2\lambda\|Z_{L,U}\|^2$, Φ achieves a minimum at $\lambda^* = -W \cdot Z_{L,U}/\|Z_{L,U}\|^2$, for which we have

$$\Phi(\lambda^*) = \|W\|^2 - \frac{(W \cdot Z_{L,U})^2}{\|Z_{L,U}\|^2} = \|W\|^2 - \frac{(y_L W \cdot X_L - y_U W \cdot X_U)^2}{\|Z_{L,U}\|^2}.$$

Clearly, the largest norm decrease will be given by the pair L, U that maximizes the rightmost term in the preceding formula. However, computing them would require two nested passes over the whole sample. A simpler way of obtaining a reasonable direction is to ignore the $\|Z_{L,U}\|^2$ denominator and computing L, U in a single pass as

$$L = \arg\min_j \{y_j W \cdot X_j\}, \quad U = \arg\max_j \{y_j W \cdot X_j : \alpha_j > 0\}. \tag{4}$$

Notice that we have $\lambda^* > 0$ and these choices imply that the new U coefficient will be $\alpha'_U = \alpha_U - \lambda^*$; this clearly requires $\alpha_U > 0$ to begin with. Moreover, to make sure that $\alpha'_L \leqslant 1$ and $\alpha'_U \geqslant 0$, we must further restrict the choice of λ^* as

$$\lambda^* = \min\left\{-\frac{W \cdot Z_{L,U}}{\|Z_{L,U}\|^2}, 1 - \alpha_L, \alpha_U\right\}. \tag{5}$$

Observe that $\lambda^* > 0$, since we must have $W \cdot Z_{L,U} < 0$ to have a descent direction. These choices coincide exactly [14] with the standard implementation of MDM for MNP.

Notice that MDM iteratively updates a weight vector $W^t = \sum \alpha_j^t y_j X_j$ as

$$W^{t+1} = W^t + \lambda_t^*(y_{L_t} X_{L_t} - y_{U_t} X_{U_t}) = W_t + \lambda_t^* Z_t.$$

The resulting algorithm can be easily extended to a kernel setting that only requires keeping track of the approximate margins $d_j^t = y_j W^t \cdot X_j$, the norms $\|W^t\|^2$ and the α_j^t coefficients. The first two values can be updated as

$$d_j^{t+1} = d_j^t + \lambda^* y_j (y_{L_t} X_{L_t} \cdot X_j - y_{U_t} X_{U_t} \cdot X_j), \tag{6}$$

$$\|W^{t+1}\|^2 = \|W^t\|^2 + 2\lambda^*(d_{L_t}^t - d_{U_t}^t) + (\lambda^*)^2 \|Z_{L_t,U_t}\|^2, \tag{7}$$

while the $\boldsymbol{\alpha}$ updates are given by

$$\alpha_{L_t}^{t+1} = \alpha_{L_t}^t + \lambda_t^*, \quad \alpha_{U_t}^{t+1} = \alpha_{U_t}^t - \lambda_t^*, \quad \alpha_i^{t+1} = \alpha_i^t \quad \forall i \neq L_t, U_t. \tag{8}$$

Algorithm 1 gives the general structure of the MDM algorithm and, as we shall see next, also of the SMO algorithm. It follows that it requires essentially $2N$ kernel operations (KOs) at each iteration, i.e., those needed to update the d_j^t. This is twice as many as those required by GSK, but this extra cost is amply compensated by the greater speed and accuracy of MDM.

Algorithm 1. General structure of the MDM and SMO algorithms

```
1:  while ( stopping condition = = false ) do
2:    select optimal L, U ; (Eq. (4))
3:    compute the optimal step λ*; (Eq. (5) )
4:    update the d_j dot products; (Eq. (6))
5:    update the squared norm ||W||^2; (Eq. (7))
6:    update the α_j coefficients; (Eq. (8) )
7:  end while
```

2.2. Keerthi's et al.'s Modification 2 for SMO

The previous point of view can also be applied to the SMO algorithm. Its updates are also of the form $W' = W + \delta_L y_L X_L + \delta_U y_U X_U$ (i.e., δ_L and δ_U are the increase/decrease of the multipliers α_L and α_U) but the constraint $0 = \sum \alpha'_i y_i$ implies that $\delta_L y_L + \delta_U y_U = 0$; that is, $\delta_U y_U = -\delta_L y_L$. Therefore, the SMO updates become

$$W' = W + \delta_L y_L (X_L - X_U) = W + \delta y_L Z_{L,U},$$

where now we just write δ instead of δ_L and $Z_{L,U} = X_L - X_U$. As a consequence, writing now $\mathbf{A}' = (\alpha'_1, \ldots, \alpha'_N)$, we have $\Theta(\mathbf{A}') = \frac{1}{2}\|W'\|^2 - \sum_i \alpha'_i$ is just a function Ψ of δ that we can try to minimize as just done for MDM. In fact, it is easy to see that

$$\Psi(\delta) = \Theta(\mathbf{A}) + \delta y_L (W \cdot Z_{L,U} - (y_L - y_U)) + \frac{\delta^2}{2}\|Z_{L,U}\|^2.$$

Writing now $\Delta = W \cdot Z_{L,U} - (y_L - y_U)$ and arguing as before, the optimal δ^* is given by

$$\delta^* = -\frac{y_L (W \cdot Z_{L,U} - (y_L - y_U))}{\|Z_{L,U}\|^2} = -y_L \frac{\Delta}{\|Z_{L,U}\|^2}. \tag{9}$$

It thus follows that

$$\Psi(\delta^*) = \Theta(\alpha) - \frac{1}{2}\frac{\Delta^2}{\|Z_{L,U}\|^2},$$

and ignoring as before the $\|Z_{L,U}\|^2$ denominator, the decrease in $\Theta(\mathbf{A})$ is close to maximal if we choose L, U so that $|\Delta|$ is maximized, something we can achieve taking for instance

$$L = \arg\max_j \{W \cdot X_j - y_j\}, \quad U = \arg\min_j \{W \cdot X_j - y_j\};$$

this implies $\Delta \geqslant 0$. Notice, however, that we must guarantee $0 \leqslant \alpha'_L = \alpha_L + \delta^* \leqslant C$. Now, it follows from (9) that $\delta^* < 0$ if $y_L = 1$, something that requires $\alpha_L > 0$ to begin with. Similarly, $y_L = -1$ implies $\delta^* > 0$, which obviously requires $\alpha_L < C$. On the other hand, we also have $\delta_U^* = -y_U y_L \delta^* = y_U \Delta / \|Z_{L,U}\|^2$, and, therefore, $\delta_U^* < 0$ if $y_U = -1$, which requires now $\alpha_U > 0$. Similarly, if $y_U = 1$, $\delta_U^* > 0$ and we must have $\alpha_U < C$. Thus, we must refine our previous L, U choices as

$$L = \arg\max_j \{W \cdot X_j - y_j : (y_j = -1, \alpha_j < C) \text{ or } (y_j = 1, \alpha_j > 0)\},$$

$$U = \arg\min_j \{W \cdot X_j - y_j : (y_j = 1, \alpha_j < C) \text{ or } (y_j = -1, \alpha_j > 0)\}. \tag{10}$$

These are essentially the index selections made in Keerthi's et al.'s Modification 2. Of course, we may also have to clip the previous choice of δ^* as necessary to ensure that $0 \leqslant \alpha'_L, \alpha'_U \leqslant C$.

To close this section we point out first that if we force $y_L = y_U$ in the above index selection, the resulting algorithm would coincide with the extension to NPP of the MDM algorithm (see [15] for more details). Moreover, and as done before, SMO can also be easily extended to a kernel setting that only requires keeping track of the approximate margins $d_j^t = W_t \cdot X_j$, the norms $\|W^t\|^2$ and the α_j^t coefficients. Since the weight updates are now

$W^{t+1} = W^t + \delta_t^* y_{L_t}(X_{L_t} - X_{U_t})$, we would have

$$d_j^{t+1} = d_j^t + \delta_t^* y_{L_t}(X_{L_t} \cdot X_j - X_{U_t} \cdot X_j),$$
$$\|W^{t+1}\|^2 = \|W^t\|^2 + 2\delta_t^* y_{L_t}(d_{L_t}^t - d_{U_t}^t) + (\delta_t^*)^2 \|Z_{L_t,U_t}\|^2,$$

while the α updates are given by

$$\alpha_{L_t}^{t+1} = \alpha_{L_t}^t + \delta_t^*, \quad \alpha_{U_t}^{t+1} = \alpha_{U_t}^t - y_{L_t} y_{U_t} \delta_t^*, \quad \alpha_i^{t+1} = \alpha_i^t \quad \forall i \neq L_t, U_t. \tag{11}$$

It follows that SMO also requires essentially $2N$ KOs at each iteration, those needed to update d_j^t.

3. Cycle breaking in SVM training

3.1. Cycle breaking in the MDM algorithm for MNP

As mentioned before, the MDM algorithm is not free of cycle-related problems. A simple illustration of this is shown in Fig. 2, where we want to find the minimum norm vector on the rhomboid; the origin is assumed to be in the $\times$-marked isolated point at the figure's center. It is clear that the optimal W^* must be placed on the rhomboid face closer to the origin but if we start the MDM iterations on the lower right vector, the algorithm requires quite a few iterations to reach that face. In other words, while the MDM update directions will eventually remove wrong initial SVs, some update directions may arise repeatedly and result in many zigzags being done and, therefore, a quite slow overall training.

To make more precise both this situation and the meaning of the "cycles" we will work with, let Z^T denote a certain update vector $Z^T = Z_{L_T,U_T}$ that we assume has appeared again after K steps from a former use as the update vector Z^{T-K}; in other words, we are assuming that $L_T = L_{T-K}, U_T = U_{T-K}$. As Fig. 2, right, illustrates, it may be very well the case that, somewhere later in training, the intermediate updates $Z^{T-j}, 1 \leqslant j \leqslant K-1$, will also be repeated and, in any case, better descent directions suggest themselves. For instance, if W^1 denotes the topmost point in the upper right corner of Fig. 2 (right), notice that the third and seventh updating vectors coincide and the vector V that connects the corresponding weight vectors W^3 and W^7 defines a better updating direction that leads to the right rhomboid face by minimizing $\|W^3 + \lambda V\|^2$. Once there, just another step brings us to the optimal W^*.

In general, such an update sequence $Z^{T-K}, Z^{T-K+1}, \ldots, Z^{T-1}, Z^T = Z^{T-K}$ will be called a "cycle". If we define then $V = \sum_{j=1}^K \lambda^{T-j} Z^{T-j}$, with λ^{T-j} the optimal updating coefficient at step $T-j$, it is clear that V would have been a better descent direction at W^{T-K} than Z^T. In fact the MDM operation guarantees that $\|W^T\|^2 = \|W^{T-K} + V\|^2 < \|W^{T-K}\|^2$. This suggests to consider again at W^T the update direction defined by V instead of the MDM standard one Z^T. Notice that V will be a descent direction at W^T if $0 > W^T \cdot V = W^{T-K} \cdot V + \|V\|^2$, i.e., if $W^{T-K} \cdot V < -\|V\|^2$ (observe that we know $W^{T-K} \cdot V < 0$). The optimal λ^T that minimizes the norm $\|W^T + \lambda V\|^2$ is given by $\lambda^T = -W^T \cdot V / \|V\|^2$ and we arrive at

$$\|W^{T+1}\|^2 = \|W^T\|^2 - \frac{(W^T \cdot V)^2}{\|V\|^2}.$$

To be able to apply these updates in a kernel setting, we need the efficient computation of the quantities $W^T \cdot V$ and $\|V\|^2$, as well as that of the new coefficients α_j^{T+1} and approximate margins $d_j^{T+1} = y_j W^{T+1} \cdot X_j$. Let $\mathscr{I} = \{i_1, \ldots, i_M\}$ be the index set of those patterns X_{i_h} that appear as either X_L or X_U in V; in other words, $i_h \in \mathscr{I}$ if either $i_h = L_p$ or $i_h = U_q$, $T-K \leqslant p, q \leqslant T-1$. We have then

$$V = \sum_{j=1}^K \lambda^{T-j} Z^{T-j} = \sum_{j=1}^K \lambda^{T-j}(y_{L_{T-j}} X_{L_{T-j}} - y_{U_{T-j}} X_{U_{T-j}}) = \sum_{h=1}^M \mu_h y_{i_h} X_{i_h}, \tag{12}$$

where $M \leqslant 2K$ and we use the notation $\mu_h = \sum_{\mathscr{A}_h} \lambda_p - \sum_{\mathscr{B}_h} \lambda_q$, with

$$\mathscr{A}_h = \{p : i_h = L_p, T-K \leqslant p \leqslant T-1\},$$
$$\mathscr{B}_h = \{q : i_h = U_q, T-K \leqslant q \leqslant T-1\}.$$

It is clear now that

$$\|V\|^2 = \sum_{1 \leqslant m,n \leqslant M} \mu_m \mu_n y_{i_m} y_{i_n} X_{i_m} \cdot X_{i_n}, \tag{13}$$

and for the new $d_j^{T+1} = y_j W^{T+1} \cdot X_j$ we have

$$d_j^{T+1} = y_j W^T \cdot X_j + y_j \lambda^T \sum_{h=1}^M \mu_h y_{i_h} X_{i_h} \cdot X_j = d_j^T + y_j \lambda^T \sum_{h=1}^M \mu_h y_{i_h} X_{i_h} \cdot X_j.$$

At first sight, computing $\|V\|^2$ and the d_j^{T+1} updates would require $M^2 + MN$ KOs, with $M \leqslant 2K$. We can save on this if we compute first the N auxiliary values

$$U_j = y_j V \cdot X_j = y_j \sum_{h=1}^M \mu_h y_{i_h} X_{i_h} \cdot X_j, \quad 1 \leqslant j \leqslant N, \tag{14}$$

at an overall cost of $M \times N$ KOs, for then we have

$$d_j^{T+1} = d_j^T + \lambda^T U_j \tag{15}$$

and $\|V\|^2 = \sum_{h=1}^M \mu_h U_{i_h}$. These last computations do not thus require new KOs. Finally, since

$$\sum_{j=1}^N \alpha_j^{T+1} y_j X_j = W^{T+1} = W^T + \lambda^T V = \sum_{j=1}^N \alpha_j^T y_j X_j + \lambda^T \sum_{h=1}^M \mu_h y_{i_h} X_{i_h}, \tag{16}$$

it follows that

$$\alpha_j^{T+1} = \alpha_j^T \quad \text{if } j \notin \mathscr{I}, \quad \alpha_{i_h}^{T+1} = \alpha_{i_h}^T + \lambda^T \mu_{i_h}, \ 1 \leqslant h \leqslant M \quad \text{if } i_h \in \mathscr{I}.$$

The last updates must also verify $0 \leqslant \alpha_{i_h}^{T+1} = \alpha_{i_h}^T + \lambda^T \mu_{i_h} \leqslant 1$. If $\mu_{i_h} > 0$, the relevant bound is the upper one, while the lower one has to be checked if $\mu_{i_h} < 0$. This implies that for these special cycle updates we must clip λ^T from above as $\lambda^T \leqslant \min\{(1 - \alpha_{i_h}^T)/\mu_{i_h} : \mu_{i_h} > 0\}$ and also as $\lambda^T \leqslant \min\{-\alpha_{i_h}^T/\mu_{i_h} : \mu_{i_h} < 0\}$.

3.2. Cycle breaking in the SMO algorithm

The previous discussion can be applied to SMO training with a few small changes due to its slightly different updates and the fact that SMO tries to minimize the function $\Theta(\mathbf{A}) = \frac{1}{2}\sum \alpha_i \alpha_j y_i y_j X_i \cdot X_j - \sum \alpha_i$ instead of $\|W\|^2$. Recall that the SMO updates are of the form $W' = W + \delta y_L (X_L - X_U) = W + \delta y_L Z_{L,U}$ and we will again consider that a cycle appears when we have $L_T = L_{T-K}$, $U_T = U_{T-K}$. If this is the case, we will also define

$$\begin{aligned} V &= \sum_{j=1}^K \delta_{T-j} Y_{L_{T-j}} Z^{T-j} = \sum_{j=1}^K \delta_{T-j} y_{L_{T-j}} (X_{L_{T-j}} - X_{U_{T-j}}) \\ &= \sum_{j=1}^K \delta_{T-j} (y_{L_{T-j}} X_{L_{T-j}} - y_{L_{T-j}} y_{U_{T-j}}^2 X_{U_{T-j}}) \\ &= \sum_{h=1}^M \mu_h y_{i_h} X_{i_h}, \end{aligned}$$

where the μ_h coefficients can be computed essentially as before. Again, as it was the case for MDM, V would have been at W^{T-K} a better descent direction than Z^{T-K} and we will try to use it at W^T to arrive at a better $W^{T+1} = W^T + \delta^T V$. To find δ^T, notice that

$\Theta(\mathbf{A}^{T+1})$ becomes a function Ψ of δ and we have

$$\begin{aligned}\Psi(\delta) = \Theta(\mathbf{A}^{T+1}) &= \frac{1}{2}\|W^T + \delta V\|^2 - \sum_i \alpha_i^T - \delta\sum_{h=1}^{M}\mu_h \\ &= \frac{1}{2}\|W^T\|^2 + \delta W^T\cdot V + \frac{1}{2}\delta^2\|V\|^2 - \sum_i \alpha_i^T - \delta\sum_{h=1}^{M}\mu_h \\ &= \Theta(\mathbf{A}^T) + \delta W^T\cdot V + \frac{1}{2}\delta^2\|V\|^2 - \delta R,\end{aligned}$$

where we have written $R = \sum_{h=1}^{M}\mu_h$. Now V defines a descent direction if $\Psi'(0) = W^T\cdot V - R < 0$ and the optimum δ^T is given by

$$\delta^T = -\frac{W^T\cdot V - R}{\|V\|^2}$$

that leads to

$$\Psi(\delta^T) = \Theta(\mathbf{A}^T) - \frac{1}{2}\frac{(W^T\cdot V - R)^2}{\|V\|^2}.$$

In terms of cost, the same reasoning used for MDM applies here and implies that an V update also requires at most $M\times N$ KOs. The α updates are now similar to the ones given for MDM and we may also have to clip the optimal δ^T conveniently so that we have $0\leqslant\alpha_j^{T+1}\leqslant C$.

3.3. Cycle detection and handling

The accelerated procedures for MDM and SMO introduce the overhead cost of keeping track of potential cycles, handling the data structures needed and performing the required computations. We have followed the simple expedient of keeping successive (L_t, U_t) updating index pairs in a standard queue Q, which is searched from its beginning each time a new pair (L_T, U_T) is selected. If it is not in Q, we insert it; on the other hand, if a previous copy (L_{T-K}, U_{T-K}) is found, we check whether the updating vector V actually defines a descent direction. This means checking whether $W^T\cdot V<0$ for MDM and $W^T\cdot V - R<0$ for SMO. Notice that once we compute the coefficients μ_h that allow writing V as $\sum_{h=1}^{M}\mu_h y_{i_h}X_{i_h}$, we have

$$W^T\cdot V = \sum_{h=1}^{M}\mu_h y_{i_h}W^T\cdot X_{i_h} = \sum_{h=1}^{M}\mu_h d_{i_h}^T.$$

Hence, neither computing $W^T\cdot V$ nor R's value do require any KO.

Algorithm 2. General cycle breaking algorithm

```
1:  while ( stopping condition = = false ) do
2:    select optimal L, U ; (Eq. (4))
3:    check whether there is a cycle ;
4:    if ( cycle = = true ) then
5:      compute μ coefficients ; (Eq. (12))
6:      if ( descent = = true ) then
7:        compute U_j ; (Eq. (14))
8:        compute ||V||^2 ; (Eq. (13))
9:        update the d_j ; (Eq. (15))
10:       update the α_j coefficients; (Eq. (16))
11:     else
12:       perform a standard MDM/SMO d_j update ; (Eq. (6))
13:     end if
14:   else
15:     perform a standard MDM/SMO d_j update ; (Eq. (6))
16:   end if
17: end while
```

If V gives indeed a descent direction, we will reset the queue emptying it after the update. On the other hand, if it does not, we will remove the previous appearance (L_{T-K}, U_{T-K}) from Q and, in order to keep the training temporal structure of the (L, U) index pairs in Q, we will also remove all pairs from Q's front up to the (L_{T-K}, U_{T-K}) position. Thus, although the maximum size of Q could conceivably be $N(N-1)$, with N being sample size, actual values will be below this. Moreover, a too large queue size may result in long searches and, also, in possibly very long and not too meaningful cycles. Thus, it makes sense to limit the maximum queue size and we will use a value of 100 in our experiments.

3.4. Complexity considerations

The complexity of SVM training algorithms is usually measured by the number of the Kernel operations they require. However, quite often KOs can be cached and, on the extreme case, for medium-size sample problems this can be done with the whole kernel matrix . The complexity analysis of the training algorithm must take then other costs into account. Algorithm 2 gives the overall structure and the costliest operations for both the MDM and SMO cycle breaking procedures; we shall use it here for our subsequent complexity analysis (notice that the equation references are given for MDM but the cost of the equivalent SMO steps is essentially the same).

Starting with KOs, their bulk for standard MDM and SMO comes from updating the d_j dot products, with a cost in both cases of two KOs per pattern and iteration. Thus, if N denotes sample size and niT_{std} the total number of iterations, standard MDM and SMO essentially require $2\times N\times niT_{std}$ KOs. On the other hand, considering the general accelerated Algorithm 2, it is seen that there are two kinds of iterations, the standard ones when no acceleration takes place and the accelerated ones where V does indeed give a descent direction. Let us denote by niS and niA the number of standard and accelerated iterations, by $niT_{acc} = niS + niA$ the total number of iterations, and by $\overline{M}$ the average number of patterns in the accelerating vector V. The cost in KOs of accelerated MDM and SMO becomes then

$$2\times N\times niS + \overline{M}\times N\times niA = 2\times N\times niT_{acc} + (\overline{M}-2)\times N\times niA.$$

Thus, the cycle breaking procedures will be faster if we clearly have $niT_{acc}\ll niT_{std}$ and $\overline{M}\times niA\ll niT_{acc}$.

When KOs are cached, however, their counting must be replaced with some kind of complexity analysis. For standard MDM and SMO the costlier parts of training will be those that require to iterate over the whole sample. There are two such loops in standard MDM and SMO. The first one is needed to obtain the L and U indices and requires one floating point comparison per pattern to get L and up to two comparisons to get U (since it is computed only over SVs). The second loop is needed to update the d_j and requires two floating point sums and three multiplications per pattern. It is not easy, though, how to assign homogeneous costs to these operations. We will have to do with a somewhat less precise analysis in which we will assume the cost of any single step in a loop to be essentially constant, that is, $\Theta(1)$. Doing so, we can estimate the complexity of standard MDM and SMO as

$$complex_{std} = 2\times N\times niT_{std}\times\Theta(1) = \Theta(N\times niT_{std}). \tag{17}$$

The complexity analysis is slightly more complicated for the accelerated procedures. Going over the steps of Algorithm 2 and taking into account their corresponding computations, we have:

- the cost of step 2 (select L, U) is $\Theta(N\times niT_{acc})$;
- the cost of step 3 (detect cycles) is $\Theta(\overline{Q}\times niT_{acc})$, with $\overline{Q}$ the average queue size;
- the cost of step 5 (compute the μ coefficients) is $\Theta(\overline{M}^2\times niA)$, with $\overline{M}$ the average number of sample patterns in the updating direction V;
- the cost of each one of steps 6 (check whether V defines a descent direction), 8 (compute $\|V\|^2$) and 10 (update the α coefficients) is $\Theta(\overline{M}\times niA)$;

- the cost of step 7 (compute the U_j values) is $\Theta(N \times \overline{M} \times niA)$;
- the cost of step 9 (update the d_j values) is $\Theta(N \times niA)$;
- the combined cost of steps 12 and 15 (performing a standard update) is $\Theta(N \times niS)$.

All this leads to an overall complexity estimate of the form

$$\begin{aligned} complex_{acc} &= \Theta(N \times niT_{acc}) + \Theta(\overline{Q} \times niT_{acc}) + \Theta(\overline{M}^2 \times niA) \\ &\quad + \Theta(\overline{M} \times niA) + \Theta(N \times \overline{M} \times niA) + \Theta(N \times niA) \\ &\quad + \Theta(N \times niS). \end{aligned}$$

However, as we shall see in our experiments, $\overline{Q}$ and $\overline{M}$ usually are $\ll N$. Thus, we may expect the previous $complex_{acc}$ estimate to simplify to

$$\begin{aligned} complex_{acc} &= \Theta(N \times niT_{acc}) + \Theta(N \times \overline{M} \times niA) \\ &\quad + \Theta(N \times niA) + \Theta(N \times niS) \\ &= \Theta(N \times niT_{acc}) + \Theta(N \times \overline{M} \times niA) \end{aligned} \tag{18}$$

for we have $niA + niS = niT_{acc}$. Now, comparing estimates (17) and (18), it is clear that the situation is quite similar to counting KOs and we can expect again the accelerated procedures to be faster whenever we have $niT_{acc} \ll niT_{std}$ and $\overline{M} \times niA \ll niT_{acc}$. In other words, whenever the accelerated procedures have an advantage in KOs, they should also be faster even when KOs are cached.

4. Numerical experiments

We shall illustrate the previous procedure over the datasets Titanic (Ti), heart disease (HD), Pima Indians' diabetes (PID), breast cancer-Wisconsin (BCW), thyroid (Thy), flare (Fla), splice (Spl), image (Imag), German credit (GCr) and banana (Ban). While originally in the UCI, Statlog or other databases, we shall work here with their versions in Rätsch's Benchmark Repository [16]. Moreover, in order to perform a cross validation analysis, we will use in our experiments the 100 train-test splits provided in [16].

These problems are not linearly separable and, as usual, we will consider margin slacks ξ_i and use a linear penalty term $C\sum_i \xi_i$ for SMO and a square penalty term $C\sum_i \xi_i^2$ for MDM. An advantage of square penalties is that the linearly separable sample theory can be carried over straightforwardly by considering extended vectors and kernels [3]. On the other hand, linear penalties are usually employed for SMO as they tend to give SVMs with less SVs. An adequate version of the MDM algorithm could also be applied in a linear penalty setting, but this requires replacing standard NPP by its version over the so-called μ-Reduced Convex Hulls, where we would look for the nearest points between $\mathscr{C}_\mu(\mathscr{S}_+) = \{\sum \alpha_i X_i : y_i = 1, 0 \leqslant \alpha_i \leqslant \mu\}$ and $\mathscr{C}_\mu(\mathscr{S}_-) = \{\sum \alpha_j X_j : y_j = -1, 0 \leqslant \alpha_j \leqslant \mu\}$. However, even in this setting, a homogeneous comparison between linear penalty MDM and SMO would still be somewhat difficult as there is no clear-cut a priori translation between the C parameter used in SMO and the μ bound required by MDM, since they are related through the concrete values of the optimal SMO solution. This is the reason why we shall use different slack penalties for each method.

We will use the Gaussian kernel $k(x,x') = \exp(-\|x - x'\|^2/2\sigma^2)$ in both algorithms, although with different C and $2\sigma^2$ parameters. For SMO we will use the values suggested in [16]; Table 1 summarizes the $2\sigma^2$ and C values used for MDM. These last values have been obtained by an optimized grid search so that good classifier accuracy is achieved. There are several criteria to stop SVM training. Here we will stop MDM when $y_U W \cdot X_U - y_L W \cdot X_L < 10^{-6}$, and SMO when $W \cdot X_L - y_L - (W \cdot X_U - y_U) < 10^{-6}$. Notice that our choices for L, U ensure that the left-hand sides are positive. Although apparently similar, the effect of these conditions on the number of iterations can be quite different for MDM and SMO, because of the possibly much different scale of the weight vectors W obtained by each algorithm (this will be the case in our experiments).

Table 1
$2\sigma^2$ and C parameters used in square penalty MDM training.

	Ti	HD	PID	BCW	Thy	Fla	Spl	Imag	GCr	Ban
$2\sigma^2$	30	$10^{3.5}$	10^2	10^4	1	$10^{2.5}$	$10^{1.5}$	30	$10^{2.5}$	1
C	0.5	10	10	10	$10^{1.5}$	10	10	750	$10^{1.5}$	10

Table 2
Test errors of standard and accelerated MDM, proportion ρ_{SV} of different SVs relative to their number as % of sample size and average differences μ between multiplier values; a zero value means μ is less than 10^{-6}.

Dataset	Err. std. MDM	Err. acc. MDM	ρ_{SV}	μ
Ti	22.78 ±1.20	22.78 ±1.20	0.0000	0.000000
HD	15.65 ±3.22	15.66 ±3.23	0.0235	0.000004
PID	23.19 ±1.67	23.22 ±1.64	0.0940	0.000004
BCW	27.70 ±4.83	27.70 ±4.83	0.0000	0.000005
Thy	5.07 ±2.17	5.07 ±2.17	0.0214	0.000001
Fla	33.43 ±1.52	33.42 ±1.52	0.0706	0.000003
Spl	11.58 ±0.65	11.57 ±0.65	0.0050	0.000000
Imag	2.84 ±0.54	2.85 ±0.51	0.8038	0.000025
GCr	23.49 ±2.04	23.46 ±2.04	0.3371	0.000011
Ban	10.44 ±0.43	10.44 ±0.43	0.1300	0.000002

Table 3
Test errors of standard and accelerated SMO, proportion ρ_{SV} of different SVs relative to their number as % of sample size and average differences μ between multiplier values; a zero value means μ is less than 10^{-6}.

Dataset	Err. std. MDM	Err. acc. MDM	ρ_{SV}	μ
Ti	22.41 ±1.02	22.41 ±1.02	0.0800	0.004956
HD	15.88 ±3.24	15.88 ±3.24	0.0059	0.000000
PID	23.49 ±1.68	23.50 ±1.69	0.0085	0.000000
BCW	26.30 ±4.57	26.30 ±4.57	0.1650	0.000079
Thy	4.44 ±2.19	4.44 ±2.19	0.0214	0.000000
Fla	32.80 ±1.69	32.80 ±1.69	0.0931	0.000413
Spl	10.79 ±0.64	10.79 ±0.64	0.0000	0.000000
Imag	3.07 ±0.48	3.07 ±0.48	0.1423	0.000039
GCr	23.63 ±2.17	23.63 ±2.17	0.0129	0.000000
Ban	11.57 ±0.68	11.57 ±0.68	0.4725	0.000000

A first question to be addressed is whether accelerating MDM and SMO could result in final models with different accuracies. Tables 2 and 3 report in their second and third columns the misclassification error of standard and accelerated MDM and SMO. As it can be seen, the accuracies of the final models are essentially the same: average and standard deviation values are almost equal in both cases and a Wilcoxon rank test at the 10% level fails to detect any significant difference on the averages. Moreover, it can also be said that the final standard and accelerated models are quite similar. Tables 2 and 3 also give the percentage ρ_{SV} of different SVs among both models relative to training sample size and the average μ of the absolute differences between normalized multiplier values of both standard and accelerated models at individual training patterns. More precisely, we have

$$\rho_{SV} = \frac{1}{N}\frac{1}{100}\sum_{i=1}^{100} v_i, \quad \mu = \frac{1}{N}\frac{1}{100}\sum_{i=1}^{100}\sum_{j=1}^{N}\left|\frac{\alpha_{ij}^S}{A} - \frac{\alpha_{ij}^A}{A}\right|,$$

where N denotes training sample size, we write v_i for the number of SVs that appear in just one of the final standard or accelerated

models for the i-th training-test split, $\alpha_{ij}^S, \alpha_{ij}^A$ are the multipliers of pattern j in the final standard or accelerated models obtained for the i-th split and A is a normalizing factor with values $A=1$ for MDM and $A=C$ for SMO, so that the above fractions are bounded by 1. As it can be seen in the tables, the final models have less than 0.9% different SVs for all datasets in MDM, and less than 0.5% for SMO. On the other hand the normalized multiplier difference is generally well below than 5×10^{-4} except for the Titanic dataset and SMO.

Turning our attention to training costs, we will consider for both algorithms two model building scenarios, a first one where we assume a KO has to be computed each time it is required, and a second one where the required kernel matrices are assumed to be cached before training starts. Tables 4 and 5 give for MDM and SMO the averages of the number of iterations and of KOs for both the standard and accelerated procedures, together with their standard deviations. For the accelerated MDM a Wilcoxon rank test at the 10% level shows that the improvement over the standard MDM both in the number of iterations and KOs is statistically relevant. The only exception appears in the Titanic dataset, where the accelerated version involves computing more KOs while there is an improvement in the number of iterations. This is easily explained by the additional costs of computing the accelerated updates, which in this case do not make up for a global improvement. On the other hand there are clear gains for the other datasets, mainly in the Image, German Credit and Heart Disease problems. Accelerated SMO, in turn, shows clear savings in KOs for all datasets, but savings are now more modest than for the best MDM results, as they mostly fall in the 10–30% range (although with bigger gains for the Image and Banana datasets). However, a straight comparison between MDM and SMO cannot strictly be made: recall that stopping criteria are actually different for both methods. This results in quite different number of standard (and, therefore, accelerated) training iterations but it can be safely said that the accelerated procedures tend to result in less costly training.

We finally compare the standard and accelerated procedures when the whole kernel matrix has been cached. As mentioned in Section 3.4, besides the sample size N and the total number of iterations niT_{std} and niT_{acc} of the standard and accelerated methods, the most relevant parameters when complexities are compared in this situation are the average number $\overline{M}$ of sample patterns in the accelerating direction V, the average queue size $\overline{Q}$ and the number $\overline{niA}$ of accelerated iterations. As it can be seen in Tables 6 and 7, the values of the latter are quite small when compared to sample size. It is also clear from these tables that the products $\overline{M}\times\overline{niA}$ are clearly much smaller than the corresponding values of niT_{acc} given in the third column of Tables 4 and 5. This suggests that the number of KOs could also be a good general complexity indicator when the kernel matrix is actually cached. To test this, we have measured actual execution times for the training loops of two "bare", stripped down versions of the standard and accelerated SMO algorithms. The resulting average times are given in the second and third columns of Table 8, together with their standard deviations; the fourth column gives the average time ratio. Times have been measured in seconds in an Intel Dual Core machine with a 2.4 GHz clock and 2 GB RAM. As it can be seen, time averages are smaller for standard SMO in the Titanic and Splice datasets, but only in the second one the difference is significant under a Wilcoxon rank test at the

Table 4
Average number of iterations and kernel operations (in thousands) for the standard and accelerated MDM algorithm.

Dataset	Iterations		Kernel operations	
	Standard MDM	Accelerated MDM	Standard MDM	Accelerated MDM
Ti	558 ±36	551 ±30*	169 ±11*	185 ±16
HD	3269 ±733	674 ±37*	1122 ±252	320 ±19*
PID	9915 ±1206	3403 ±200*	9311 ±1133	4329 ±262*
BCW	1942 ±213	733 ±42*	783 ±86	390 ±26*
Thy	813 ±138	454 ±55*	230 ±39	169 ±20*
Fla	17,671 ±2149	13,008 ±6296*	23,591 ±2868	18,358 ±7895*
Spl	4581 ±443	4465 ±499*	9176 ±888	8975 ±994*
Imag	157,734 ±13,623	24,596 ±1876*	410,584 ±35,461	79,350 ±6098*
GCr	34,202 ±2776	6699 ±244*	47,986 ±3895	12,031 ±431*
Ban	5676 ±625	1905 ±177*	4558 ±502	1975 ±181*

An asterisk is shown whenever there is statistical improvement of one model over its counterpart.

Table 6
Sample size N, average queue length $\overline{Q}$, average number $\overline{M}$ of patterns in the updating vector V and of accelerated updates $\overline{niA}$ and the product of the latter for the accelerated MDM algorithm.

Dataset	N	$\overline{Q}$	$\overline{M}$	$\overline{niA}$	$\overline{M}\times\overline{niA}$
Ti	150	66.34 ±10.28	47.87 ±16.90	2.96 ±1.32	142
HD	170	10.2 ±4.04	5.52 ±0.48	78.37 ±6.74	433
PID	468	14.65 ±1.22	9.09 ±0.50	189.03 ±10.94	1718
BCW	200	18.12 ±1.87	6.76 ±0.72	55.58 ±4.88	376
Thy	140	8.08 ±1.26	7.44 ±1.29	31.35 ±6.73	233
Fla	666	75.19 ±25.22	19.88 ±6.18	79.82 ±38.00	1587
Spl	1000	88.30 ±5.96	3.70 ±4.18	12.00 ±6.30	44
Imag	1300	4.00 ±0.28	3.14 ±0.13	3526.40 ±186.42	11,073
GCr	700	6.39 ±0.36	4.20 ±0.11	715.73 ±26.49	3006
Ban	400	13.26 ±1.77	9.07 ±0.93	90.91 ±7.72	825

Table 5
Average number of iterations and kernel operations (in thousands) for the standard and accelerated SMO algorithm.

Dataset	Iterations		Kernel operations	
	Standard SMO	Accelerated SMO	Standard SMO	Accelerated SMO
Ti	213 ±40	139 ±20*	65 ±12	57 ±8*
HD	332 ±121	217 ±54*	114 ±41	96 ±27*
PID	730 ±194	521 ±88*	686 ±182	600 ±110*
BCW	4693 ±2721	2316 ±1185*	1891 ±1096	1243 ±650*
Thy	519 ±224	305 ±107*	147 ±63	117 ±42*
Fla	1867 ±1771	1044 ±584*	2494 ±2364	1752 ±1087*
Spl	4574 ±434	4234 ±195*	9164 ±870	8832 ±529*
Imag	184,481 ±39,426	78,142 ±14,311*	480,206 ±102,626	265,702 ±49,157*
GCr	2985 ±369	1931 ±179*	4190 ±517	3343 ±311*
Ban	131,599 ±110,892	36,799 ±32,244*	105,675 ±89,047	40,198 ±35,517*

An asterisk is shown whenever there is statistical improvement of one model over its counterpart.

Table 7
Sample size N, average queue length $\overline{Q}$, average number $\overline{M}$ of patterns in the updating vector V and of accelerated updates $\overline{niA}$ and the product of the latter for the accelerated SMO algorithm.

Dataset	N	$\overline{Q}$	$\overline{M}$	$\overline{niA}$	$\overline{M} \times \overline{niA}$
Ti	150	5.55 ±1.74	4.97 ±0.40	19.15 ±3.76	95
HD	170	13.81 ±3.36	8.31 ±1.00	14.57 ±5.18	121
PID	468	28.94 ±4.49	10.85 ±2.03	22.45 ±7.77	244
BCW	200	11.09 ±1.62	9.82 ±0.78	158.86 ±92.22	1560
Thy	140	6.51 ±1.26	7.43 ±1.16	29.61 ±10.57	220
Fla	666	31.53 ±9.43	7.27 ±0.97	74.95 ±68.53	545
Spl	1000	89.28 ±9.26	40.13 ±42.73	8.75 ±8.98	351
Imag	1300	11.46 ±1.23	10.65 ±1.23	4533.35 ±866.87	48,280
GCr	700	41.94 ±3.05	29.62 ±3.68	31.04 ±6.18	919
Ban	400	5.81 ±0.53	6.09 ±0.52	4392.18 ±4092.21	26,748

Table 8
Average training times in seconds and standard deviations for standard and accelerated SMO and average ratios.

Dataset	Standard SMO training time	Accelerated SMO training time	Average ratio
Ti	2.60 ±0.55	2.68 ±0.56	0.97
HD	4.85 ±1.70	4.39 ±1.20*	1.10
PID	29.32 ±7.76	27.32 ±5.14*	1.07
BCW	78.54 ±45.39	53.79 ±27.97*	1.46
Thy	5.87 ±2.50	5.18 ±1.80*	1.13
Fla	106.84 ±99.79	83.07 ±51.39*	1.29
Spl	364.78 ±33.48*	384.94 ±28.95	0.95
Imag	18,342.11 ±3933.55	15,749.31 ±2922.93*	1.16
GCr	180.33 ±23.13	172.95 ±16.85*	1.04
Ban	4157.00 ±3532.89	1697.22 ±1507.01*	2.45

An asterisk is shown whenever there is statistical improvement of one model over its counterpart.

Table 9
Theoretical complexities of standard and accelerated SMO and their ratios.

Dataset	Standard SMO complexity	Accelerated SMO complexity	Ratio
Ti	48.02	45.58	1.05
HD	84.54	76.24	1.11
PID	512.46	475.71	1.08
BCW	1488.90	1003.85	1.48
Thy	108.99	94.71	1.15
Fla	1865.13	1399.19	1.33
Spl	6861.00	6699.68	1.02
Imag	359,737.95	214,658.99	1.68
GCr	3134.25	2659.59	1.18
Ban	78,959.40	32,714.20	2.41

10% level. The average times of the accelerated procedure are significantly smaller for the other eight datasets. Moreover, actual times seem to agree quite well with more abstract estimates where standard and accelerated SMO complexities would be given by the formulae

$$complex_{std} \simeq 1.5 \times N \times niT_{std},$$
$$complex_{acc} \simeq 1.5 \times N \times niT_{acc} + N \times \overline{M} \times niA.$$

Table 9 gives in columns 2 and 3 the concrete values given by these formulae for $complex_{std}$ and $complex_{acc}$ and their ratio in column 4. As it can be seen, for all datasets but Image, there is a quite good agreement between the ratios in the fourth columns of Tables 8 and 9. Of course, this should not be taken entirely at face value, but seems to indicate that the advantages of the accelerated procedures when KOs must be computed also extend to the situation where kernel matrices are cached.

5. Discussion and conclusions

Fast SVM training is an important issue for which many proposals have been given. In this work we have related Platt's well known Sequential Minimal Optimization with the Mitchell–Demyanov–Malozemov (MDM) algorithm, an approach to SVM construction that follows a geometric point of view. The analysis of application of MDM to simple problems shows that, while quite fast and precise, its convergence may be slowed by the presence of cycles, that is, the repeated selection of some concrete MDM updating pattern pairs. Moreover, its relationship with SMO suggests that this algorithm may also be affected by updating cycles.

In this work, that extends previous one in [9], we have addressed these questions and show for both algorithms how to take advantage of the presence of cycles in the updating sequences by partially collapsing them in a single updating vector that gives a better minimizing direction. As we have seen over 10 publicly available datasets, substantial savings in the number of iterations and KOs can be obtained for both algorithms over most of these problems and this is also indeed the case when the kernel matrix is cached. Moreover, our approach is clearly compatible with other procedural improvements on SMO, such as Platt's type I and II updates or shrinking, and this may also be the case with the so-called second order methods proposed to speed up SMO [5]. Finally, and from a broader point of view, it may also imply that other insights derived from geometrical methods for SVM training could also be applied to the more standard decomposition methods, possibly further improving them.

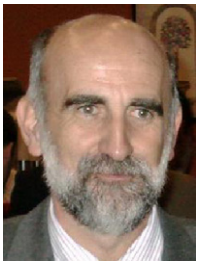
References

[1] V. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.
[2] B. Schölkopf, A.J. Smola, Learning With Kernels: Support Vector Machines, Regularization, Optimization and Beyond, Machine Learning, MIT Press, Cambridge, MA, 2002.
[3] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge University Press, Cambridge, 2000.
[4] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy, Improvements to Platt's smo algorithm for SVM classifier design, Neural Computation 13 (3) (2001) 637–649.
[5] T. Glasmachers, Ch. Igel, Maximum gain working set selection using second order information for support vector machines, Journal of Machine Learning Research 7 (2006) 1437–1466.
[6] R.E. Fan, P.H. Chen, Ch.J. Lin, Working set selection using second order information for training support vector machines, Journal of Machine Learning Research 6 (2005) 1889–1918.
[7] J.C. Platt, Fast training of support vector machines using sequential minimal optimization, Advances in Kernel Methods—Support Vector Machines, 1999, pp. 185–208.
[8] T. Joachims, Making large-scale support vector machine learning practical, Advances in Kernel Methods—Support Vector Machines, 1999, pp. 169–184.
[9] A. Barbero, J. López, J. Dorronsoro, An accelerated MDM algorithm for SVM training, in: Advances in Computational Intelligence and Learning, Proceedings of ESANN 2008 Conference, 2008, pp. 421–426.
[10] K.P. Bennett, E.J. Bredensteiner, Duality and geometry in SVM classifiers, in: Proceedings of the 17th International Conference on Machine Learning, 2000, pp. 57–64.
[11] V. Franc, V. Hlaváč, An iterative algorithm learning the maximal margin classifier, Pattern Recognition 36 (2003) 1985–1996.
[12] E.G. Gilbert, Minimizing the quadratic form on a convex set, SIAM Journal on Control 4 (1966) 61–79.
[13] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy, A fast iterative nearest point algorithm for support vector machine classifier design, IEEE Transactions on Neural Networks 11 (1) (2000) 124–136.
[14] B.F. Mitchell, V.F. Dem'yanov, V.N. Malozemov, Finding the point of a polyhedron closest to the origin, SIAM Journal on Control 12 (1974) 19–26.
[15] J. López, A. Barbero, J. Dorronsoro, On the equivalence of the smo and mdm algorithms for svm training, in: Proceedings of the 2008 European Conference

in Machine Learning, ECML PKDD 2008, Lecture Notes in Computer Science, vol. 5211, Springer, Berlin, 2008, pp. 288–300.
[16] G. Rätsch, Benchmark repository, ⟨http://ida.first.fraunhofer.de/projects/bench/benchmarks.htm⟩.

Álvaro Barbero received the Computer Scientist degree from the Universidad Autónoma de Madrid (UAM) in 2006 is currently a student in a Master/Ph.D. degree in Computer Science at the same university. Nowadays he is working at the UAM under a national predoctoral grant, in collaboration with the Instituto de Ingeniería del Conocimiento. His research interests are in pattern recognition, kernel methods and wind power forecasting.

Jorge López received his Computer Engineering degree from the Universidad Autónoma de Madrid in 2006, where he got an honorific mention as the best student. Currently he is attending the postgraduate programme organized by the Computer Engineering Department of the same university. His research interests concentrate on support vector machines, but also cover additional machine learning and pattern recognition paradigms.



José R. Dorronsoro received the Licenciado en Matemáticas degree from the Universidad Complutense de Madrid in 1977 and the Ph.D. degree in Mathematics from Washington University in St. Louis in 1981. Currently he is a Full Professor in the Computer Engineering Department of the Universidad Autónoma de Madrid, of which he was head from 1993 to 1996. His research interests are in neural networks, image processing and pattern recognition.