A New Algorithm for Training SVMs Using Approximate Minimal Enclosing Balls

Emanuele Frandi[4], Maria Grazia Gasparo[3], Stefano Lodi[1], Ricardo Ñanculef[2], and Claudio Sartori[1]

[1] Dept. of Electronics, Computer Science and Systems, University of Bologna, Italy
{claudio.sartori,stefano.lodi}@unibo.it
[2] Dept. of Informatics, Federico Santa María University, Chile
jnancu@inf.utfsm.cl
[3] Dept. of Energetics *Sergio Stecco*, University of Florence, Italy
mariagrazia.gasparo@unifi.it
[4] Dept. of Mathematics *Ulisse Dini*, University of Florence, Italy
emanuele.frandi@gmail.com

Abstract. It has been shown that many kernel methods can be equivalently formulated as minimal enclosing ball (MEB) problems in a certain feature space. Exploiting this reduction, efficient algorithms to scale up Support Vector Machines (SVMs) and other kernel methods have been introduced under the name of Core Vector Machines (CVMs). In this paper, we study a new algorithm to train SVMs based on an instance of the Frank-Wolfe optimization method recently proposed to approximate the solution of the MEB problem. We show that, specialized to SVM training, this algorithm can scale better than CVMs at the price of a slightly lower accuracy.

1 Introduction

Support Vector Machines (SVMs) are currently a well known set of methods to address classification and other machine learning problems with successful results in several application fields. SVMs are usually formulated as the solution of a convex quadratic programming problem (QP), for which a naive implementation requires $O(m^2)$ space and $O(m^3)$ time in the number of examples m [15,18], complexities that are prohibitively expensive for large scale problems. Major research efforts have been hence directed towards scaling up SVM algorithms to large datasets.

Due to the typically dense structure of the matrices involved in the QP, large SVM problems are usually adressed using an *active set method* where at each iteration only a small number of variables are allowed to change [14,8,13]. The most prominent example is Sequential Minimal Optimization (SMO [4,13]), where only two variables are selected for optimization each time. The main disadvantage of these methods is that they tend to have slow convergence when getting closer to the solution and performance results in practice are very sensitive to the size of the active set, the way to select the active variables and other implementation details like the caching strategy used to avoid the repetitive computation of the kernel function [14]. Other attempts to scale up SVM methods consist in adapting *interior point methods* to some classes of the SVM QP [5]. For large-scale problems however the resulting rank of the kernel matrix can still be too high to be handled efficiently [18]. The reformulation of the SVM objective function [6], sampling methods to reduce the number of variables in the problem [11,10] and the combination of small SVMs using ensemble methods [12] have also been explored.

A key observation exploited in [18] is that the QP underlying many SVMs is equivalent to the QP defining a *minimal enclosing ball* (MEB) problem, that is, the problem of computing the ball of smallest radius containing a set of points. Recent advances in computational geometry have demonstrated that there are algorithms capable to approximate a MEB with any degree of accuracy ϵ in $O(1/\epsilon)$ iterations independently of the number of points and the dimensionality of the space [18]. Based on these ideas, [18] obtains an algorithm to train SVMs which exhibits linear time-complexity in the number of examples m approximating the solution with any desired accuracy. Experiments in [18] confirm that this approach can be faster than SMO in large-scale problems [4,13].

In this paper, we study a new algorithm to exploit the reduction of SVMs to MEBs, which can still approximate the solution with any degree of accuracy ϵ but is considerably simpler than [18]. The algorithm completely avoids the resolution of reduced QP problems at each iteration and the corresponding computation and storage of reduced gram matrices that [18] need to by-pass using SMO and a caching strategy. Experiments in small, medium and large-scale classification problems show that, specialized to SVMs, this algorithm to compute MEBs is slightly less accurate than [18] but can significatively improve the complexity and actual running times of this algorithm.

2 MEB Problems and Support Vector Machines

As pointed out first by [18] and then generalized by [19], several SVM methods can be equivalently formulated as MEB problems in a certain feature space, that is, as the computation of the ball of smallest radius containing the images of the dataset under a mapping into a dot-product space $\mathcal{Z}$.

Consider a dataset $S = \{\mathbf{x}_i : i \in I\} \subset \mathcal{X}$ indexed by $I = \{1, 2, \ldots, m\}$ and a mapping $\phi : \mathcal{X} \to \mathcal{Z}$, such that $\tilde{k}(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)\ \forall i, j \in I$ for a given kernel function $\tilde{k}$. The closed ball of center $\mathbf{c} \in \mathcal{Z}$ and radius $r \in \mathbb{R}^+$ is denoted by $\mathcal{B}(\mathbf{c}, r)$. The MEB $\mathcal{B}(\mathbf{c}^*, r^*)$ of $\phi(S)$ can hence be defined as the solution of the following optimization problem

$$\begin{aligned} &\mathbf{min}_{r^2,\mathbf{c}}\ r^2 \\ &\mathbf{st:}\ \|\phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq r^2\ \ \forall i \in I\,, \end{aligned} \tag{1}$$

whose Lagrange dual is given by [20]

$$\begin{gathered}\mathbf{max}_{\boldsymbol{\alpha}}\ \Phi(\boldsymbol{\alpha}) = \textstyle\sum_{i\in I} \alpha_i \tilde{k}(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,j\in I} \alpha_i \alpha_j \tilde{k}(\mathbf{x}_i, \mathbf{x}_j) \\ \mathbf{st:} \textstyle\sum_{i\in I} \alpha_i = 1,\ \ \alpha_i \geq 0\ \ \forall i \in I\,.\end{gathered} \tag{2}$$

If we denote by $\boldsymbol{\alpha}^*$ the solution of (2), the center $\mathbf{c}^*$ and the squared radius r^{*2} of the MEB of $\phi(S)$ follow from strong duality:

$$\mathbf{c}^* = \sum_{i\in I} \alpha_i^* \phi(\mathbf{x}_i) \ \ ; \ \ r^{*2} = \Phi(\boldsymbol{\alpha}^*)\,. \tag{3}$$

Problem (2) coincides with the formulation of a number of kernel methods by correspondingly setting the kernel function $\tilde{k}$. For L2-SVMs [18] for example, we are given a set of labels $\{y_i : i \in I\}$ for the set of training inputs $\{\mathbf{x}_i : i \in I\}$ and we are aimed to implement a decision function to predict the class of new inputs $\mathbf{x} \in \mathcal{X}$. If the problem is two-class we can suppose without loss of generality that $y_i = +1$ if $\mathbf{x}_i$ belongs to the first class and $y_i = -1$ if $\mathbf{x}_i$ belongs to the other class. SVMs implement a decision function of the following form [15]

$$h(\mathbf{x}) = \operatorname{sign}\left(\sum_{i\in I} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\right)\,, \tag{4}$$

where k is a kernel function used to implement non-linear classification boundaries in non-linearly separable cases and the weights α_i are determined by minimizing a risk functional. This functional takes the form of problem (2) if we set $\tilde{k}$ to

$$\tilde{k}(\mathbf{x}_i, \mathbf{x}_j) = y_i y_j \left(k(\mathbf{x}_i, \mathbf{x}_j) + 1\right) + \delta_{ij}/C\,, \tag{5}$$

where C is a regularization parameter used within the SVM to handle noisy data. Details can be found in [18]. The kernel k is required to satisfy the following normalization condition

$$k(\mathbf{x}_i, \mathbf{x}_i) = \Delta^2 = \text{constant}\,, \tag{6}$$

which is automatically satisfied for SVM kernels of the form $k(\mathbf{x}_i, \mathbf{x}_j) = g(\|\mathbf{x}_i - \mathbf{x}_j\|)$. Equivalent constructions for other kernel methods including regression and novelty detection are presented in [18] and [19]. Refer also to [19] for studies on kernels which do not satisfy the normalization condition.

3 Approximate MEBs and Core Vector Machines

The problem of computing the minimal enclosing ball of a set of points has a long history in computational geometry [20]. Traditional algorithms to find exact MEBs scale exponentially in the space dimension and hence could not be applied to SVM problems in which the feature space induced by the kernel is high-dimensional. Recent advances have been obtained by shifting the attention to approximation algorithms capable to output a ball near the MEB by a given degree of precision.

Given $\epsilon > 0$, a ball $\mathcal{B}(\mathbf{c}, r) \subset \mathcal{Z}$ is said to be an $(1+\epsilon)$-*approximation* to the MEB $\mathcal{B}(\mathbf{c}^*, r^*)$ of $A \subset \mathcal{Z}$ (or more shortly an $(1+\epsilon)$-*MEB* of A) if $r \leq r^*$ and $A \subset \mathcal{B}(\mathbf{c}, (1+\epsilon)r)$.

In [1] and [20], algorithms to compute $(1+\epsilon)$-MEBs that scale independently of the dimension of $\mathcal{Z}$ and the cardinality of A have been provided. These algorithms are built on the concept of ϵ-*core set* for A, that is, a subset $C \subset A$ whose MEB is a $(1+\epsilon)$-MEB of A. In particular, the algorithm described in [1] is able to provide an ϵ-core set of a set A in no more than $O(1/\epsilon)$ iterations. We denote with C_k the core set approximation obtained at the k-th iteration and its MEB as $B_k = \mathcal{B}(\mathbf{c}_k, r_k)$. Starting from a given C_0, at each iteration C_{k+1} is defined as the union of C_k and the point of A furthest from $\mathbf{c}_k$. The algorithm then computes B_{k+1} and stops if $\mathcal{B}(\mathbf{c}_{k+1}, (1+\epsilon)r_{k+1})$ contains A.

Exploiting these ideas, Tsang and colleagues introduced in [18] the CVM (Core Vector Machine) for training SVMs supporting a reduction to a MEB problem. CVM is described in Algorithm 1, where each C_k is identified by the index set $I_k \subset I$. The expression for the radius r_k follows easily from (3). Moreover, it is easy to show [18] that step 10 exactly looks for the point $\mathbf{x}_{i^*}$ whose image $\phi(\mathbf{x}_{i^*})$ is the furthest from $\mathbf{c}_k$. The index i^* is then included in the index set and the reduced QP corresponding to the MEB of the new approximate core set is solved.

```
   Data: $S = \{(\mathbf{x}_i, y_i) : i \in I\}$, indexed by $I = \{1, 2, \ldots, m\}$, $\epsilon$, $I_0$, $\boldsymbol{\alpha}_0$.
1  For any $\boldsymbol{\alpha} \in \mathbb{R}^k$, define $R(\boldsymbol{\alpha}) = \sum_{i,j \in I_k} \alpha_i \alpha_j \tilde{k}(\mathbf{x}_i, \mathbf{x}_j)$;
2  $\Delta^2 \longleftarrow \tilde{k}(\mathbf{x}_1, \mathbf{x}_1)$;
3  $r_0^2 \longleftarrow \Delta^2 - R(\boldsymbol{\alpha}_0)$;
4  $i^* \longleftarrow \arg\max_{i \in I} \delta^2(i, \boldsymbol{\alpha}_0) = \Delta^2 + R(\boldsymbol{\alpha}_0) - 2\sum_{j \in I_0} \alpha_{0,j} \tilde{k}(\mathbf{x}_j, \mathbf{x}_i)$;
5  $k \longleftarrow 0$;
6  while $\delta^2(i^*, \boldsymbol{\alpha}_k) > (1+\epsilon)^2 r_k^2$ do
7      $k \longleftarrow k + 1$;
8      $I_k \longleftarrow I_{k-1} \cup \{i^*\}$;
9      Find $\boldsymbol{\alpha}_k$ by solving the reduced QP
       $\min_{\boldsymbol{\alpha} \in \mathbb{R}^m} R(\boldsymbol{\alpha})$ s.t. $\sum_{i \in I_k} \alpha_i = 1,\ \alpha_i \geq 0 \forall i \in I_k$;
10     $r_k^2 \longleftarrow \Delta^2 - R(\boldsymbol{\alpha}_k)$;
11     $i^* \longleftarrow \arg\max_{i \in I} \delta^2(i, \boldsymbol{\alpha}_k) = \Delta^2 + R(\boldsymbol{\alpha}_k) - 2\sum_{j \in I_k} \alpha_{k,j} \tilde{k}(\mathbf{x}_j, \mathbf{x}_i)$;
12 end
13 Output $I_S = I_k$, $\boldsymbol{\alpha} = \boldsymbol{\alpha}_k$.
```

Algorithm 1. Training SVMs using Approximate MEBs (MEB-SVMs)

Algorithm 1 has two main sources of computational overhead: the computation of the furthest point in step 10, which is linear in m, and the solution of the optimization problem in step 8. Complexity of step 10 can be made constant and independent of m by suitable sampling techniques [18]. As regards the optimization step 8, CVMs adopt a SMO method [4,13], where only two variables are selected for optimization at each iteration. It is known that the cost of each SMO iteration is not too high, but the method can require a large number of iterations in order to satisfy reasonable stopping criteria [13].

4 The New Algorithm for MEB-SVMs

In this section we provide a new algorithm to train SVMs based on approximate MEBs, which avoids the resolution of the reduced QP problem at each iteration of Algorithm 1. For this purpose we adopt a variant of the Frank-Wolfe algorithm, recently studied in [3] and [20]. The Frank-Wolfe method (see Algorithm 2, where $\mathbf{e}_i$ denotes the i-th vector of the canonical basis of $\mathbb{R}^m$) solves the general problem of maximizing a concave function $g(\boldsymbol{\alpha})$ on the unit simplex.

```
1 $\boldsymbol{\alpha} \longleftarrow \mathbf{e}_{i_0}$ with $i_0 = \arg\max_{i \in I} g(\mathbf{e}_i)$ ;
2 for $k = 0, 1, \ldots$ until a stopping criterion is satisfied, do
3     $i^* \longleftarrow \arg\max_{i \in I} \nabla g(\boldsymbol{\alpha}_k)_i$;
4     $\lambda^* \longleftarrow \arg\max_{\lambda \in [0,1]} g\left((1-\lambda)\boldsymbol{\alpha}_k + \lambda \mathbf{e}_{i^*}\right)$;
5     $\boldsymbol{\alpha}_{k+1} \longleftarrow (1-\lambda^*)\boldsymbol{\alpha}_k + \lambda^* \mathbf{e}_{i^*}$;
6 end
```

Algorithm 2. The Frank-Wolfe Algorithm

Consider now the problem defined in (2). The gradient of $\Phi(\boldsymbol{\alpha})$ is given by $\nabla\Phi(\boldsymbol{\alpha})_i = \tilde{k}(\mathbf{x}_i, \mathbf{x}_i) - 2\sum_{j \in I} \alpha_j \tilde{k}(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\phi(\mathbf{x}_i)^T \left(\sum_{j \in I} \alpha_j \phi(\mathbf{x}_j)\right)$. Furthermore, given any $\boldsymbol{\alpha}_k$, we can define the ball $B_k = \mathcal{B}(\mathbf{c}_k, r_k)$, where $\mathbf{c}_k = \sum_{j \in I} \alpha_j \phi(\mathbf{x}_j)$ and $r_k^2 = \Phi(\boldsymbol{\alpha}_k)$. In this way, step 3 of Algorithm 2 corresponds to

$$\arg\max_{i \in I} \left(\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\phi(\mathbf{x}_i)^T \mathbf{c}_k\right) = \arg\max_{i \in I} \|\mathbf{c}_k - \phi(\mathbf{x}_i)\|^2 , \tag{7}$$

that is, step 3 selects the point maximizing the distance from the center of B_k, as does step 10 of Algorithm 1. Moreover, the updating formula for $\boldsymbol{\alpha}$ shows that only the i^*-th entry of $\boldsymbol{\alpha}$ can become nonzero at each iteration. The optimization step 4 is analytic: indeed, it can be shown [3,20] that

$$\lambda^* = \frac{1}{2} - \frac{r_k^2}{2\|\mathbf{c}_k - \phi(\mathbf{x}_{i^*})\|^2} . \tag{8}$$

In [20], it has been proved that $\{r_k\}$ is a monotonically increasing sequence. On the other hand, r_k is bounded by the radius $r*$ of the optimal MEB. Therefore, if we stop the Frank-Wolfe procedure with the same criterion as in Algorithm 1, then we have $r_k \leq r*$, that is, I_k identifies a core set for $\phi(S)$ and B_k a

$(1+\epsilon)$-MEB. In conclusion, we can preserve the main structure of Algorithm 1, substituting step 8 with a step 8′ defined as follows:

$$\boldsymbol{\alpha}_{k+1} \longleftarrow (1-\lambda^*)\boldsymbol{\alpha}_k + \lambda^* \mathbf{e}_{i^*}, \quad \text{with } \lambda^* \longleftarrow \left(\frac{1}{2} - \frac{r_k^2}{2\delta^2(i^*, \boldsymbol{\alpha}_k)}\right) .$$

This way, we are able to avoid the solution of reduced QP problems. The cost of each iteration of the algorithm is now dominated only by the computation of the furthest point from $\mathbf{c}_k$. The new algorithm hence offers considerably lighter iterations, whose complexity does not depend on additional numerical routines. Moreover, a theoretical bound in terms of ϵ on the total number of iterations, exactly identical to that of CVMs, holds. Indeed, Yildirim [20] proved that the method computes an ϵ-core set in $O(1/\epsilon)$ iterations.

Remark. From a numerical optimization point of view, the following remark might be of interest. From Theorem 2.2 of [3], we have that $\Phi(\boldsymbol{\alpha}^*) - \Phi(\boldsymbol{\alpha}_k) \leq 4C_f/(k+3)$, where C_f is a constant, bounded by the squared diameter of the MEB of S, that is, $4\Phi(\boldsymbol{\alpha}^*)$. Therefore, $\Phi(\boldsymbol{\alpha}^*) - \Phi(\boldsymbol{\alpha}_k) \leq 16\Phi(\boldsymbol{\alpha}^*)/(k+3)$. It follows that the relative error $(\Phi(\boldsymbol{\alpha}^*) - \Phi(\boldsymbol{\alpha}_k))/\Phi(\boldsymbol{\alpha}^*)$ is smaller than any fixed tolerance τ after $(16/\tau - 3)$ iterations.

5 Experiments and Conclusions

We provide experiments on 13 datasets listed in Table 1. The size of each problem is quantified by considering the number of training examples m and the number of classes K. Since we employ a one-versus-one method for multi-category classification [9], the number of binary submodels to compute is given by $s_K = K(K-1)/2$ [15]. The average number of examples for submodel is denoted by $\bar{m}_K$. The datasets *Kdd-full*, *Ijcnn* and *extended Usps* (abbreviated as *Usps-ext*) were used as in previous research to test the large-scale capabilities of CVMs [18] and are available at [17]. The other problems are available at [7] or [2]. For problems without a predefined test-set (*Iris*, *Wine*, *Glass* and *Kdd-10pc*) a 20% of the data was randomly selected and reserved to assess prediction accuracy.

SVMs were trained using a gaussian kernel $k(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\|\mathbf{x}_1 - \mathbf{x}_2\|^2/\sigma^2)$. For datasets *Kdd-full*, *Usps-ext* and *Ijcnn* we used the hyper-parameter values reported at [18,16]. For the small datasets ($\leq 10^4$ examples) hyper-parameters were determined using 10-fold cross-validation. For the remaining datasets σ^2 was set to the average squared distance among training patterns and C was determined on a logarithmic grid $[2^0, 2^{12}]$ using a validation-set (30% of the training-set). In order to compute I_0 and $\boldsymbol{\alpha}_0$ for Algorithm 1 we adopted the initialization procedure of [17]. We also adopted the LRR caching strategy designed in [17] for CVMs to avoid the computation of recently used kernel values.

Tables 1 and 2 summarize the experimental results obtained after training with the parameters determined by the model-selection procedure. The proposed algorithm is denoted here as FVM (acronym of Frank-Wolfe Vector Machine). In Table 1 we present accuracy and running times obtained on a 2.40GHz Intel

Table 1. Dataset features (first 4 columns) and Performance of the two algorithms (last 4 columns)

Dataset	m	K	$\bar{m}_K$	**Accuracy** (%)		**Time** (secs)	
				CVM	**FVM**	**CVM**	**FVM**
Glass	1.4E+02	6	2.3E+01	67.93	55.81	1.5E-01	8.0E-03
Wine	1.1E+02	3	3.8E+01	97.49	97.49	5.0E-02	3.5E-03
Iris	1.2E+02	3	4.0E+01	96.24	94.62	5.5E-03	3.0E-03
Letter	1.5E+04	26	5.8E+02	97.44	95.98	3.2E+01	1.2E+01
Usps	7.3E+03	10	7.3E+02	95.76	95.37	6.3E+00	6.0E+00
Pendigits	7.5E+03	10	7.5E+02	98.40	97.91	5.3E-01	1.1E+00
Protein	1.8E+04	3	5.9E+03	69.84	60.91	8.7E+03	3.0E+02
Mnist	6.0E+04	10	6.0E+03	98.53	97.82	4.0E+02	3.0E+02
Shuttle	4.4E+04	7	6.2E+03	99.77	98.41	2.1E+00	3.3E-01
Ijcnn	5.0E+04	2	2.5E+04	98.55	95.17	1.3E+03	1.0E+02
Kdd-10pc	4.0E+05	5	7.9E+04	99.92	98.84	1.3E+03	2.5E+00
Usps-ext	2.7E+05	2	1.3E+05	99.50	99.25	1.8E+01	6.7E+00
Kdd-full	4.9E+06	2	2.4E+06	90.88	91.66	1.4E+00	4.7E+00

Table 2. Detailed measures of complexity

Dataset	**Number of Support Vectors**		**Kernel Evals (with Cache)**		**Kernel Evals (without Cache)**		**SMO-It**
	CVM	**FVM**	**CVM**	**FVM**	**CVM**	**FVM**	**CVM**
Glass	1.2E+02	1.4E+02	2.8E+04	2.8E+04	1.3E+07	5.6E+05	2.8E+05
Wine	5.6E+01	6.4E+01	1.0E+04	1.0E+04	4.7E+06	4.0E+05	1.5E+05
Iris	1.4E+01	7.0E+01	5.3E+03	9.3E+03	4.7E+05	3.8E+05	1.7E+04
Letter	6.8E+03	9.4E+03	3.6E+07	6.6E+07	2.5E+09	2.1E+09	2.6E+07
Usps	1.6E+03	2.1E+03	6.6E+06	8.4E+06	1.8E+08	1.5E+08	1.7E+06
Pendigits	7.6E+02	1.7E+03	2.3E+06	8.2E+06	2.7E+07	8.7E+07	4.5E+05
Protein	1.5E+04	1.4E+04	3.2E+08	2.7E+08	7.9E+11	7.2E+09	1.5E+08
Mnist	1.0E+04	1.2E+04	1.9E+08	2.3E+08	1.5E+10	1.1E+09	3.1E+07
Shuttle	3.1E+02	7.7E+02	8.8E+05	3.0E+06	1.7E+08	5.5E+06	2.0E+06
Ijcnn	8.0E+03	7.1E+03	6.5E+08	7.2E+08	1.2E+11	2.4E+09	3.1E+07
Kdd-10pc	1.1E+04	1.4E+03	1.9E+09	1.4E+07	6.9E+10	2.7E+07	2.2E+07
Usps-ext	5.4E+02	5.6E+02	1.6E+07	9.3E+06	5.4E+08	9.7E+06	1.1E+06
Kdd-full	1.6E+01	7.0E+02	6.6E+04	1.5E+07	7.0E+06	1.5E+07	1.7E+05

Core 2 Duo with 2GB RAM running openSUSE 11.1. In Table 2 we present platform-independent measures of complexity: the number of Support Vectors in the resulting model, which determines model complexity, and the number of kernel evaluations, that is, the number of times that the kernel function is evaluated on a pair of examples, which is frequently used as the measure of algorithmic complexity for kernel methods. Table 2 presents both the total number of kernel evaluations required by the algorithm and the values effectively computed after checking the cache: *Kernel Evals (without Cache)* and *Kernel Evals (with Cache)* respectively. Finally, we report the total number of SMO-iterations (*SMO-It*) carried out by the CVM algorithm.

Experiments show that FVM exhibits a slightly lower accuracy than CVM but a quite better algorithmic complexity, measured as the number of kernel evaluations. Even if the caching strategy benefits significatively more to CVM than FVM, by reducing by some orders of magnitude the number kernel evaluations that CVM effectively computes, our method exhibits much better actual running times in all the cases but two (on one of them CVM is faster at expenses of the classification accuracy).

Note that the number of support vectors (training points taking part in the final model) is of the same order for both algorithms. However, CVM computes on average between 10^3 and 10^4 SMO iterations for each support vector in the model while for FVM the inclusion of a new active point in the model involves the evaluation of a single analytical rule (see section 4). The overall runtime disadvantage of the CVM algorithm can hence be explained by the additional cost of the SMO iterations, which are completely avoided in the proposed algorithm.

References

1. Bădoiu, M., Clarkson, K.: Smaller core-sets for balls. In: Proceedings of the SODA 2003, pp. 801–802. SIAM, Philadelphia (2003)
2. Chang, C.-C., Lin, C.-J.: LIBSVM: a library for support vector machines (2010)
3. Clarkson, K.: Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. In: Proceedings of SODA 2008, pp. 922–931. SIAM, Philadelphia (2008)
4. Fan, R.-E., Chen, P.-H., Lin, C.-J.: Working set selection using second order information for training support vector machines. Journal of Machine Learning Research 6, 1889–1918 (2005)
5. Fine, S., Scheinberg, K.: Efficient SVM training using low-rank kernel representations. Journal of Machine Learning Research 2, 243–264 (2002)
6. Fung, G., Mangasarian, O.: Finite newton method for lagrangian support vector machine classification. Neurocomputing 55(1-2), 39–55 (2003)
7. Hettich, S., Bay, S.: The UCI KDD Archive (2010), http://kdd.ics.uci.edu
8. Joachims, T.: Making large-scale support vector machine learning practical, pp. 169–184. MIT Press, Cambridge (1999)
9. Kressel, U.: Pairwise classification and support vector machines. In: Advances in Kernel Methods: Support Vector Learning, pp. 255–268. MIT Press, Cambridge (1999)
10. Kumar, K., Bhattacharya, C., Hariharan, R.: A randomized algorithm for large scale support vector learning. In: Advances in Neural Information Processing Systems, vol. 20, pp. 793–800. MIT Press, Cambridge (2008)
11. Lee, Y.-J., Huang, S.: Reduced support vector machines: A statistical theory. IEEE Transactions on Neural Networks 18(1), 1–13 (2007)
12. Pavlov, D., Mao, J., Dom, B.: An improved training algorithm for support vector machines. In: Proceedings of the 15th International Conference on Pattern Recognition, vol. 2, pp. 2219–2222. IEEE, Los Alamitos (2000)
13. Platt, J.: Fast training of support vector machines using sequential minimal optimization, pp. 185–208 (1999)
14. Scheinberg, K.: An efficient implementation of an active set method for SVMs. Journal of Machine Learning Research 7, 2237–2257 (2006)

15. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2001)
16. Tsang, I., Kocsor, A., Kwok, J.: Simpler core vector machines with enclosing balls. In: ICML 2007, pp. 911–918. ACM, New York (2007)
17. Tsang, I., Kocsor, A., Kwok, J.: LibCVM Toolkit (2009)
18. Tsang, I., Kwok, J., Cheung, P.-M.: Core vector machines: Fast SVM training on very large data sets. J. of Machine Learning Research 6, 363–392 (2005)
19. Tsang, I., Kwok, J., Zurada, J.: Generalized core vector machines. IEEE Transactions on Neural Networks 17(5), 1126–1140 (2006)
20. Yildirim, E.A.: Two algorithms for the minimum enclosing ball problem. SIAM Journal on Optimization 19(3), 1368–1391 (2008)