Core Vector Machines:
Fast SVM Training on Very Large Data Sets

Ivor W. Tsang IVOR@CS.UST.HK
James T. Kwok JAMESK@CS.UST.HK
Pak-Ming Cheung PAKMING@CS.UST.HK
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay
Hong Kong

Editor: Nello Cristianini

Abstract

Standard SVM training has $O(m^3)$ time and $O(m^2)$ space complexities, where m is the training set size. It is thus computationally infeasible on very large data sets. By observing that practical SVM implementations only *approximate* the optimal solution by an iterative strategy, we scale up kernel methods by exploiting such "approximateness" in this paper. We first show that many kernel methods can be equivalently formulated as minimum enclosing ball (MEB) problems in computational geometry. Then, by adopting an efficient approximate MEB algorithm, we obtain provably approximately optimal solutions with the idea of core sets. Our proposed Core Vector Machine (CVM) algorithm can be used with nonlinear kernels and has a time complexity that is *linear* in m and a space complexity that is *independent* of m. Experiments on large toy and real-world data sets demonstrate that the CVM is as accurate as existing SVM implementations, but is much faster and can handle much larger data sets than existing scale-up methods. For example, CVM with the Gaussian kernel produces superior results on the KDDCUP-99 intrusion detection data, which has about five million training patterns, in only 1.4 seconds on a 3.2GHz Pentium–4 PC.

Keywords: kernel methods, approximation algorithm, minimum enclosing ball, core set, scalability

1. Introduction

In recent years, there has been a lot of interest on using kernels in various machine learning problems, with the support vector machines (SVM) being the most prominent example. Many of these kernel methods are formulated as quadratic programming (QP) problems. Denote the number of training patterns by m. The training time complexity of QP is $O(m^3)$ and its space complexity is at least quadratic. Hence, a major stumbling block is in scaling up these QP's to large data sets, such as those commonly encountered in data mining applications.

To reduce the time and space complexities, a popular technique is to obtain low-rank approximations on the kernel matrix, by using the Nyström method (Williams and Seeger, 2001), greedy approximation (Smola and Schölkopf, 2000), sampling (Achlioptas et al., 2002) or matrix decompositions (Fine and Scheinberg, 2001). However, on very large data sets, the resulting rank of the kernel matrix may still be too high to be handled efficiently.

Another approach to scale up kernel methods is by chunking (Vapnik, 1998) or more sophisticated decomposition methods (Chang and Lin, 2004; Osuna et al., 1997b; Platt, 1999; Vishwanathan et al., 2003). However, chunking needs to optimize the entire set of non-zero Lagrange multipliers that have been identified, and the resultant kernel matrix may still be too large to fit into memory. Osuna et al. (1997b) suggested optimizing only a fixed-size subset (*working set*) of the training data each time, while the variables corresponding to the other patterns are frozen. Going to the extreme, the sequential minimal optimization (SMO) algorithm (Platt, 1999) breaks the original QP into a series of smallest possible QPs, each involving only two variables.

A more radical approach is to avoid the QP altogether. Mangasarian and his colleagues proposed several variations of the Lagrangian SVM (LSVM) (Fung and Mangasarian, 2003; Mangasarian and Musicant, 2001a,b) that obtain the solution with a fast iterative scheme. However, for nonlinear kernels (which is the focus in this paper), it still requires the inversion of an $m \times m$ matrix. Recently, Kao et al. (2004) and Yang et al. (2005) also proposed scale-up methods that are specially designed for the linear and Gaussian kernels, respectively.

Similar in spirit to decomposition algorithms are methods that scale down the training data before inputting to the SVM. For example, Pavlov et al. (2000b) used boosting to combine a large number of SVMs, each is trained on only a small data subsample. Alternatively, Collobert et al. (2002) used a neural-network-based gater to mix these small SVMs. Lee and Mangasarian (2001) proposed the reduced SVM (RSVM), which uses a random rectangular subset of the kernel matrix. Instead of random sampling, one can also use active learning (Schohn and Cohn, 2000; Tong and Koller, 2000), squashing (Pavlov et al., 2000a), editing (Bakir et al., 2005) or even clustering (Boley and Cao, 2004; Yu et al., 2003) to intelligently sample a small number of training data for SVM training. Other scale-up methods include the Kernel Adatron (Friess et al., 1998) and the SimpleSVM (Vishwanathan et al., 2003). For a more complete survey, interested readers may consult (Tresp, 2001) or Chapter 10 of (Schölkopf and Smola, 2002).

In practice, state-of-the-art SVM implementations typically have a training time complexity that scales between $O(m)$ and $O(m^{2.3})$ (Platt, 1999). This can be further driven down to $O(m)$ with the use of a parallel mixture (Collobert et al., 2002). However, these are only empirical observations and not theoretical guarantees. For reliable scaling behavior to very large data sets, our goal is to develop an algorithm that can be proved (using tools in analysis of algorithms) to be asymptotically efficient in both time and space.

A key observation is that practical SVM implementations, as in many numerical routines, only *approximate* the optimal solution by an iterative strategy. Typically, the stopping criterion uses either the precision of the Lagrange multipliers or the duality gap (Smola and Schölkopf, 2004). For example, in SMO, SVM^{light} (Joachims, 1999) and SimpleSVM, training stops when the Karush-Kuhn-Tucker (KKT) conditions are fulfilled within a tolerance parameter ε. Experience with these softwares indicate that near-optimal solutions are often good enough in practical applications. However, such "approximateness" has never been exploited in the design of SVM implementations.

On the other hand, in the field of theoretical computer science, approximation algorithms with provable performance guarantees have been extensively used in tackling computationally difficult problems (Garey and Johnson, 1979; Vazirani, 2001). Let C be the cost of the solution returned by an approximate algorithm, and C^* be the cost of the optimal solution. An approximate algorithm has *approximation ratio* $\rho(n)$ for an input size n if $\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$. Intuitively, this ratio measures how bad the approximate solution is compared with the optimal solution. A large (small) approximation ratio means the solution is much worse than (more or less the same as) the optimal solution. Observe that $\rho(n)$ is always ≥ 1. If the ratio does not depend on n, we may just write ρ and call the algorithm an ρ-*approximation algorithm*. Well-known NP-complete problems that can be efficiently addressed using approximation algorithms include the vertex-cover problem and the set-covering problem. This large body of experience suggests that one may also develop approximation algorithms for SVMs, with the hope that training of kernel methods will become more tractable, and thus more scalable, in practice.

In this paper, we will utilize an approximation algorithm for the *minimum enclosing ball* (MEB) problem in computational geometry. The MEB problem computes the ball of minimum radius enclosing a given set of points (or, more generally, balls). Traditional algorithms for finding exact MEBs (e.g., (Megiddo, 1983; Welzl, 1991)) do not scale well with the dimensionality d of the points. Consequently, recent attention has shifted to the development of approximation algorithms (Bădoiu and Clarkson, 2002; Kumar et al., 2003; Nielsen and Nock, 2004). In particular, a breakthrough was obtained by Bădoiu and Clarkson (2002), who showed that an $(1+\varepsilon)$-approximation of the MEB can be efficiently obtained using *core sets*. Generally speaking, in an optimization problem, a core set is a subset of input points such that we can get a good approximation to the original input by solving the optimization problem directly on the core set. A surprising property of (Bădoiu and Clarkson, 2002) is that the size of its core set can be shown to be *independent* of both d and the size of the point set.

In the sequel, we will show that there is a close relationship between SVM training and the MEB problem. Inspired from the core set-based approximate MEB algorithms, we will then develop an approximation algorithm for SVM training that has an approximation ratio of $(1+\varepsilon)^2$. Its time complexity is *linear* in m while its space complexity is *independent* of m. In actual implementation, the time complexity can be further improved with the use of probabilistic speedup methods (Smola and Schölkopf, 2000).

The rest of this paper is organized as follows. Section 2 gives a short introduction on the MEB problem and its approximation algorithm. The connection between kernel methods and the MEB problem is given in Section 3. Section 4 then describes our proposed Core Vector Machine (CVM) algorithm. The core set in CVM plays a similar role as the working set in decomposition algorithms, which will be reviewed briefly in Section 5. Finally, experimental results are presented in Section 6, and the last section gives some concluding remarks. Preliminary results on the CVM have been recently reported in (Tsang et al., 2005).

2. The (Approximate) Minimum Enclosing Ball Problem

Given a set of points $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, where each $\mathbf{x}_i \in \mathbb{R}^d$, the minimum enclosing ball of $\mathcal{S}$ (denoted $\text{MEB}(\mathcal{S})$) is the smallest ball that contains all the points in $\mathcal{S}$. The MEB problem can be dated back as early as in 1857, when Sylvester (1857) first investigated the smallest radius disk enclosing m points on the plane. It has found applications in diverse areas such as computer graphics (e.g., for collision detection, visibility culling), machine learning (e.g., similarity search) and facility locations problems (Preparata, 1985). The MEB problem also belongs to the larger family of shape fitting problems, which attempt to find the shape (such as a slab, cylinder, cylindrical shell or spherical shell) that best fits a given point set (Chan, 2000).

Traditional algorithms for finding exact MEBs (such as (Megiddo, 1983; Welzl, 1991)) are not efficient for problems with $d > 30$. Hence, as mentioned in Section 1, it is of practical interest to study faster approximation algorithms that return a solution within a multiplicative factor of $1+\varepsilon$ to

the optimal value, where ε is a small positive number. Let $B(\mathbf{c}, R)$ be the ball with center $\mathbf{c}$ and radius R. Given an $\varepsilon > 0$, a ball $B(\mathbf{c}, (1+\varepsilon)R)$ is an *$(1+\varepsilon)$-approximation* of $\text{MEB}(\mathcal{S})$ if $R \leq r_{\text{MEB}(\mathcal{S})}$ and $\mathcal{S} \subset B(\mathbf{c}, (1+\varepsilon)R)$. In many shape fitting problems, it is found that solving the problem on a subset, called the *core set*, $\mathcal{Q}$ of points from $\mathcal{S}$ can often give an accurate and efficient approximation. More formally, a subset $\mathcal{Q} \subseteq \mathcal{S}$ is a core set of $\mathcal{S}$ if an expansion by a factor $(1+\varepsilon)$ of its MEB contains $\mathcal{S}$, i.e., $\mathcal{S} \subset B(\mathbf{c}, (1+\varepsilon)r)$, where $B(\mathbf{c}, r) = \text{MEB}(\mathcal{Q})$ (Figure 1).

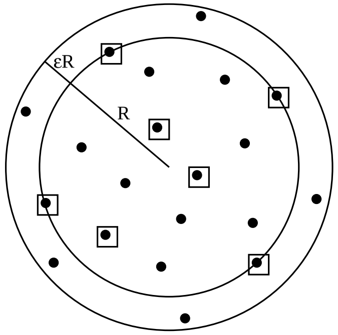


Figure 1: The inner circle is the MEB of the set of squares and its $(1+\varepsilon)$ expansion (the outer circle) covers all the points. The set of squares is thus a core set.

A breakthrough on achieving such an $(1+\varepsilon)$-approximation was recently obtained by Bădoiu and Clarkson (2002). They used a simple iterative scheme: At the tth iteration, the current estimate $B(\mathbf{c}_t, r_t)$ is expanded incrementally by including the furthest point outside the $(1+\varepsilon)$-ball $B(\mathbf{c}_t, (1+\varepsilon)r_t)$. This is repeated until all the points in $\mathcal{S}$ are covered by $B(\mathbf{c}_t, (1+\varepsilon)r_t)$. Despite its simplicity, a surprising property is that the number of iterations, and hence the size of the final core set, depends only on ε but *not* on d or m. The independence of d is important on applying this algorithm to kernel methods (Section 3) as the kernel-induced feature space can be infinite-dimensional. As for the remarkable independence on m, it allows both the time and space complexities of our algorithm to grow slowly (Section 4.3).

3. MEB Problems and Kernel Methods

The MEB can be easily seen to be equivalent to the hard-margin support vector data description (SVDD) (Tax and Duin, 1999), which will be briefly reviewed in Section 3.1. The MEB problem can also be used to find the radius component of the radius-margin bound (Chapelle et al., 2002; Vapnik, 1998). Thus, Kumar et al. (2003) has pointed out that the MEB problem can be used in support vector clustering and SVM parameter tuning. However, as will be shown in Section 3.2, other kernel-related problems, such as the soft-margin one-class and two-class SVMs, can also be viewed as MEB problems. Note that finding the soft-margin one-class SVM is essentially the same as fitting the MEB with outliers, which is also considered in (Har-Peled and Wang, 2004). However, a limitation of their technique is that the number of outliers has to be moderately small in order to be effective. Another heuristic approach for scaling up the soft-margin SVDD using core sets has also been proposed in (Chu et al., 2004).

3.1 Hard-Margin SVDD

Given a kernel k with the associated feature map φ, let the MEB (or hard-margin ball) in the kernel-induced feature space be $B(\mathbf{c},R)$. The primal problem in the hard-margin SVDD is

$$\min_{R,\mathbf{c}} R^2 \ : \ \|\mathbf{c}-\varphi(\mathbf{x}_i)\|^2 \leq R^2, \ \ i=1,\ldots,m. \tag{1}$$

The corresponding dual is

$$\begin{array}{ll} \max_{\alpha_i} & \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i,\mathbf{x}_i) - \sum_{i,j=1}^{m} \alpha_i \alpha_j k(\mathbf{x}_i,\mathbf{x}_j) \\ \text{s.t.} & \alpha_i \geq 0, \quad i=1,\ldots,m \\ & \sum_{i=1}^{m} \alpha_i = 1, \end{array}$$

or, in matrix form,

$$\max_{\alpha} \ \alpha' \mathrm{diag}(\mathbf{K}) - \alpha' \mathbf{K} \alpha \ : \ \alpha \geq \mathbf{0}, \ \alpha' \mathbf{1} = 1, \tag{2}$$

where $\alpha = [\alpha_i,\ldots,\alpha_m]'$ are the Lagrange multipliers, $\mathbf{0}=[0,\ldots,0]'$, $\mathbf{1}=[1,\ldots,1]'$ and $\mathbf{K}_{m\times m} = [k(\mathbf{x}_i,\mathbf{x}_j)]$ is the kernel matrix. As is well-known, this is a QP problem. The primal variables can be recovered from the optimal α as

$$\mathbf{c} = \sum_{i=1}^{m} \alpha_i \varphi(\mathbf{x}_i), \quad R = \sqrt{\alpha' \mathrm{diag}(\mathbf{K}) - \alpha' \mathbf{K} \alpha}. \tag{3}$$

3.2 Viewing Kernel Methods as MEB Problems

Consider the situation where

$$k(\mathbf{x},\mathbf{x}) = \kappa, \tag{4}$$

a constant. All the patterns are then mapped to a sphere in the feature space. (4) will be satisfied when either

1. the isotropic kernel $k(\mathbf{x},\mathbf{y}) = K(\|\mathbf{x}-\mathbf{y}\|)$ (e.g., Gaussian kernel); or
2. the dot product kernel $k(\mathbf{x},\mathbf{y}) = K(\mathbf{x}'\mathbf{y})$ (e.g., polynomial kernel) with normalized inputs; or
3. any normalized kernel $k(\mathbf{x},\mathbf{y}) = \frac{K(\mathbf{x},\mathbf{y})}{\sqrt{K(\mathbf{x},\mathbf{x})}\sqrt{K(\mathbf{y},\mathbf{y})}}$

is used. These three cases cover most kernel functions used in real-world applications. Schölkopf et al. (2001) showed that the hard (soft) margin SVDD then yields identical solution as the hard (soft) margin one-class SVM, and the weight $\mathbf{w}$ in the one-class SVM solution is equal to the center $\mathbf{c}$ in the SVDD solution.

Combining (4) with the condition $\alpha'\mathbf{1}=1$ in (2), we have $\alpha'\mathrm{diag}(\mathbf{K}) = \kappa$. Dropping this constant term from the dual objective in (2), we obtain a simpler optimization problem:

$$\max_{\alpha} -\alpha' \mathbf{K} \alpha \ : \ \alpha \geq \mathbf{0}, \ \alpha' \mathbf{1} = 1. \tag{5}$$

Conversely, whenever the kernel k satisfies (4), any QP of the form (5) can be regarded as a MEB problem in (1). Note that (2) and (5) yield the same set of optimal α's. Moreover, the optimal (dual) objectives in (2) and (5) (denoted d_1^* and d_2^* respectively) are related by

$$d_1^* = d_2^* + \kappa. \tag{6}$$

In the following, we will show that when (4) is satisfied, the duals in a number of kernel methods can be rewritten in the form of (5). While the 1-norm error has been commonly used for the SVM, our main focus will be on the 2-norm error. In theory, this could be less robust in the presence of outliers. However, experimentally, its generalization performance is often comparable to that of the L1-SVM (Lee and Mangasarian, 2001; Mangasarian and Musicant, 2001a,b). Besides, the 2-norm error is more advantageous here because a soft-margin L2-SVM can be transformed to a hard-margin one. While the 2-norm error has been used in classification (Section 3.2.2), we will also extend its use for novelty detection (Section 3.2.1).

3.2.1 One-Class L2-SVM

Given a set of unlabeled patterns $\{\mathbf{z}_i\}_{i=1}^m$ where $\mathbf{z}_i$ only has the input part $\mathbf{x}_i$, the one-class L2-SVM separates outliers from the normal data by solving the primal problem:

$$\begin{array}{ll} \min_{\mathbf{w},\rho,\xi_i} & \|\mathbf{w}\|^2 - 2\rho + C\sum_{i=1}^{m} \xi_i^2 \\ \text{s.t.} & \mathbf{w}'\varphi(\mathbf{x}_i) \geq \rho - \xi_i, \quad i=1,\ldots,m, \end{array} \tag{7}$$

where $\mathbf{w}'\varphi(\mathbf{x}) = \rho$ is the desired hyperplane and C is a user-defined parameter. Unlike the classification LSVM, the bias is not penalized here. Moreover, note that constraints $\xi_i \geq 0$ are not needed for the L2-SVM (Keerthi et al., 2000). The corresponding dual is

$$\max_{\alpha} \ -\alpha' \left(\mathbf{K} + \frac{1}{C}\mathbf{I}\right) \alpha \ : \ \alpha \geq \mathbf{0}, \ \alpha'\mathbf{1} = 1, \tag{8}$$

where $\mathbf{I}$ is the $m \times m$ identity matrix. From the Karush-Kuhn-Tucker (KKT) conditions, we can recover

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \varphi(\mathbf{x}_i) \tag{9}$$

and $\xi_i = \frac{\alpha_i}{C}$, and then $\rho = \mathbf{w}'\varphi(\mathbf{x}_i) + \frac{\alpha_i}{C}$ from any support vector $\mathbf{x}_i$.

Rewrite (8) in the form of (5) as:

$$\max_{\alpha} \ -\alpha' \tilde{\mathbf{K}} \alpha \ : \ \alpha \geq \mathbf{0}, \ \alpha'\mathbf{1} = 1, \tag{10}$$

where

$$\tilde{\mathbf{K}} = [\tilde{k}(\mathbf{z}_i,\mathbf{z}_j)] = \left[k(\mathbf{x}_i,\mathbf{x}_j) + \frac{\delta_{ij}}{C}\right]. \tag{11}$$

Since $k(\mathbf{x},\mathbf{x}) = \kappa$,

$$\tilde{k}(\mathbf{z},\mathbf{z}) = \kappa + \frac{1}{C} \equiv \tilde{\kappa}$$

is also a constant. This one-class L2-SVM thus corresponds to the MEB problem (1), in which φ is replaced by the nonlinear map $\tilde{\varphi}$ satisfying $\tilde{\varphi}(\mathbf{z}_i)'\tilde{\varphi}(\mathbf{z}_j) = \tilde{k}(\mathbf{z}_i, \mathbf{z}_j)$. It can be easily verified that this $\tilde{\varphi}$ maps the training point $\mathbf{z}_i = \mathbf{x}_i$ to a higher dimensional space, as

$$\tilde{\varphi}(\mathbf{z}_i) = \begin{bmatrix} \varphi(\mathbf{x}_i) \\ \frac{1}{\sqrt{C}}\mathbf{e}_i \end{bmatrix},$$

where $\mathbf{e}_i$ is the m-dimensional vector with all zeroes except that the ith position is equal to one.

3.2.2 Two-Class L2-SVM

In the two-class classification problem, we are given a training set $\{\mathbf{z}_i = (\mathbf{x}_i, y_i)\}_{i=1}^m$ with $y_i \in \{-1, 1\}$. The primal of the two-class L2-SVM is

$$\begin{aligned} \min_{\mathbf{w},b,\rho,\xi_i} \quad & \|\mathbf{w}\|^2 + b^2 - 2\rho + C\sum_{i=1}^m \xi_i^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}'\varphi(\mathbf{x}_i) + b) \geq \rho - \xi_i, \quad i = 1, \ldots, m. \end{aligned} \tag{12}$$

The corresponding dual is

$$\max_{\alpha} \; -\alpha'\left(\mathbf{K} \odot \mathbf{y}\mathbf{y}' + \mathbf{y}\mathbf{y}' + \frac{1}{C}\mathbf{I}\right)\alpha \;:\; \alpha \geq \mathbf{0}, \;\; \alpha'\mathbf{1} = 1, \tag{13}$$

where $\odot$ denotes the Hadamard product and $\mathbf{y} = [y_1, \ldots, y_m]'$. Again, we can recover

$$\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \varphi(\mathbf{x}_i), \quad b = \sum_{i=1}^m \alpha_i y_i, \quad \xi_i = \frac{\alpha_i}{C}, \tag{14}$$

from the optimal α and then $\rho = y_i(\mathbf{w}'\varphi(\mathbf{x}_i) + b) + \frac{\alpha_i}{C}$ from any support vector $\mathbf{z}_i$. Alternatively, ρ can also be obtained from the fact that QP's have zero duality gap. Equating the primal (12) and dual (13), we have

$$\|\mathbf{w}\|^2 + b^2 - 2\rho + C\sum_{i=1}^m \xi_i^2 = -\sum_{i,j=1}^m \alpha_i\alpha_j \left(y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + y_i y_j + \frac{\delta_{ij}}{C}\right).$$

Substituting in (14), we then have

$$\rho = \sum_{i,j=1}^m \alpha_i\alpha_j \left(y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + y_i y_j + \frac{\delta_{ij}}{C}\right). \tag{15}$$

Rewriting (13) in the form of (5), we have

$$\max_{\alpha} \; -\alpha'\tilde{\mathbf{K}}\alpha \;:\; \alpha \geq \mathbf{0}, \;\; \alpha'\mathbf{1} = 1, \tag{16}$$

where $\tilde{\mathbf{K}} = [\tilde{k}(\mathbf{z}_i, \mathbf{z}_j)]$ with

$$\tilde{k}(\mathbf{z}_i, \mathbf{z}_j) = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + y_i y_j + \frac{\delta_{ij}}{C}, \tag{17}$$

Again, this $\tilde{k}$ satisfies (4), as

$$\tilde{k}(\mathbf{z}, \mathbf{z}) = \kappa + 1 + \frac{1}{C} \equiv \tilde{\kappa},$$

a constant. Thus, this two-class L2-SVM can also be viewed as a MEB problem (1) in which φ is replaced by $\tilde{\varphi}$, with

$$\tilde{\varphi}(\mathbf{z}_i) = \begin{bmatrix} y_i\varphi(\mathbf{x}_i) \\ y_i \\ \frac{1}{\sqrt{C}}\mathbf{e}_i \end{bmatrix}$$

for any training point $\mathbf{z}_i$. Note that as a classification (supervised learning) problem is now reformulated as a MEB (unsupervised) problem, the label information gets encoded in the feature map $\tilde{\varphi}$. Moreover, all the support vectors of this L2-SVM, including those defining the margin and those that are misclassified, now reside on the surface of the ball in the feature space induced by $\tilde{k}$. A similar relationship connecting one-class classification and binary classification for the case of Gaussian kernels is also discussed by Schölkopf et al. (2001). In the special case of a hard-margin SVM, $\tilde{k}$ reduces to $\tilde{k}(\mathbf{z}_i, \mathbf{z}_j) = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + y_i y_j$ and analogous results apply.

4. Core Vector Machine (CVM)

After formulating the kernel method as a MEB problem, we obtain a transformed kernel $\tilde{k}$, together with the associated feature space $\tilde{\mathcal{F}}$, mapping $\tilde{\varphi}$ and constant $\tilde{\kappa} = \tilde{k}(\mathbf{z}, \mathbf{z})$. To solve this kernel-induced MEB problem, we adopt the approximation algorithm described in the proof of Theorem 2.2 in (Bădoiu and Clarkson, 2002). A similar algorithm is also described in (Kumar et al., 2003). As mentioned in Section 2, the idea is to incrementally expand the ball by including the point furthest away from the current center. In the following, we denote the core set, the ball's center and radius at the tth iteration by $\mathcal{S}_t, \mathbf{c}_t$ and R_t respectively. Also, the center and radius of a ball B are denoted by $\mathbf{c}_B$ and r_B. Given an $\varepsilon > 0$, the CVM then works as follows:

1. Initialize $\mathcal{S}_0$, $\mathbf{c}_0$ and R_0.

2. Terminate if there is no training point $\mathbf{z}$ such that $\tilde{\varphi}(\mathbf{z})$ falls outside the $(1+\varepsilon)$-ball $B(\mathbf{c}_t, (1+\varepsilon)R_t)$.

3. Find $\mathbf{z}$ such that $\tilde{\varphi}(\mathbf{z})$ is furthest away from $\mathbf{c}_t$. Set $S_{t+1} = \mathcal{S}_t \cup \{\mathbf{z}\}$.

4. Find the new MEB$(\mathcal{S}_{t+1})$ from (5) and set $\mathbf{c}_{t+1} = \mathbf{c}_{\text{MEB}(\mathcal{S}_{t+1})}$ and $R_{t+1} = r_{\text{MEB}(\mathcal{S}_{t+1})}$ using (3).

5. Increment t by 1 and go back to Step 2.

In the sequel, points that are added to the core set will be called *core vectors*. Details of each of the above steps will be described in Section 4.1. Despite its simplicity, CVM has an approximation guarantee (Section 4.2) and small time and space complexities (Section 4.3).

4.1 Detailed Procedure

4.1.1 Initialization

Bădoiu and Clarkson (2002) simply used an arbitrary point $\mathbf{z} \in \mathcal{S}$ to initialize $S_0 = \{\mathbf{z}\}$. However, a good initialization may lead to fewer updates and so we follow the scheme in (Kumar et al., 2003).

We start with an arbitrary point $\mathbf{z} \in \mathcal{S}$ and find $\mathbf{z}_a \in \mathcal{S}$ that is furthest away from $\mathbf{z}$ in the feature space $\tilde{\mathcal{F}}$. Then, we find another point $\mathbf{z}_b \in \mathcal{S}$ that is furthest away from $\mathbf{z}_a$ in $\tilde{\mathcal{F}}$. The initial core set is then set to be $\mathcal{S}_0 = \{\mathbf{z}_a, \mathbf{z}_b\}$. Obviously, $\text{MEB}(\mathcal{S}_0)$ (in $\tilde{\mathcal{F}}$) has center $\mathbf{c}_0 = \frac{1}{2}(\tilde{\varphi}(\mathbf{z}_a) + \tilde{\varphi}(\mathbf{z}_b))$ On using (3), we thus have $\alpha_a = \alpha_b = \frac{1}{2}$ and all the other α_i's are zero. The initial radius is

$$\begin{aligned} R_0 &= \frac{1}{2}\|\tilde{\varphi}(\mathbf{z}_a) - \tilde{\varphi}(\mathbf{z}_b)\| \\ &= \frac{1}{2}\sqrt{\|\tilde{\varphi}(\mathbf{z}_a)\|^2 + \|\tilde{\varphi}(\mathbf{z}_b)\|^2 - 2\tilde{\varphi}(\mathbf{z}_a)'\tilde{\varphi}(\mathbf{z}_b)} \\ &= \frac{1}{2}\sqrt{2\tilde{\kappa} - 2\tilde{k}(\mathbf{z}_a, \mathbf{z}_b)}. \end{aligned}$$

In a classification problem, one may further require $\mathbf{z}_a$ and $\mathbf{z}_b$ to come from different classes. On using (17), R_0 then becomes $\frac{1}{2}\sqrt{2\left(\kappa + 2 + \frac{1}{C}\right) + 2k(\mathbf{x}_a, \mathbf{x}_b)}$. As κ and C are constants, choosing the pair $(\mathbf{x}_a, \mathbf{x}_b)$ that maximizes R_0 is then equivalent to choosing the closest pair belonging to opposing classes, which is also the heuristic used in initializing the DirectSVM (Roobaert, 2000) and SimpleSVM (Vishwanathan et al., 2003).

4.1.2 Distance Computations

Steps 2 and 3 involve computing $\|\mathbf{c}_t - \tilde{\varphi}(\mathbf{z}_\ell)\|$ for $\mathbf{z}_\ell \in \mathcal{S}$. On using $\mathbf{c} = \sum_{\mathbf{z}_i \in \mathcal{S}_t} \alpha_i \tilde{\varphi}(\mathbf{z}_i)$ in (3), we have

$$\|\mathbf{c}_t - \tilde{\varphi}(\mathbf{z}_\ell)\|^2 = \sum_{\mathbf{z}_i, \mathbf{z}_j \in \mathcal{S}_t} \alpha_i \alpha_j \tilde{k}(\mathbf{z}_i, \mathbf{z}_j) - 2 \sum_{\mathbf{z}_i \in \mathcal{S}_t} \alpha_i \tilde{k}(\mathbf{z}_i, \mathbf{z}_\ell) + \tilde{k}(\mathbf{z}_\ell, \mathbf{z}_\ell). \tag{18}$$

Hence, computations are based on kernel evaluations instead of the explicit $\tilde{\varphi}(\mathbf{z}_i)$'s, which may be infinite-dimensional. Note that, in contrast, existing MEB algorithms only consider finite-dimensional spaces.

However, in the feature space, $\mathbf{c}_t$ cannot be obtained as an explicit point but rather as a convex combination of (at most) $|\mathcal{S}_t|$ $\tilde{\varphi}(\mathbf{z}_i)$'s. Computing (18) for all m training points takes $O(|\mathcal{S}_t|^2 + m|\mathcal{S}_t|) = O(m|\mathcal{S}_t|)$ time at the tth iteration. This becomes very expensive when m is large. Here, we use the probabilistic speedup method in (Smola and Schölkopf, 2000). The idea is to randomly sample a sufficiently large subset $\mathcal{S}'$ from $\mathcal{S}$, and then take the point in $\mathcal{S}'$ that is furthest away from $\mathbf{c}_t$ as the approximate furthest point over $\mathcal{S}$. As shown in (Smola and Schölkopf, 2000), by using a small random sample of, say, size 59, the furthest point obtained from $\mathcal{S}'$ is with probability 0.95 among the furthest 5% of points from the whole $\mathcal{S}$. Instead of taking $O(m|\mathcal{S}_t|)$ time, this randomized method only takes $O(|\mathcal{S}_t|^2 + |\mathcal{S}_t|) = O(|\mathcal{S}_t|^2)$ time, which is much faster as $|\mathcal{S}_t| \ll m$. This trick can also be used in the initialization step.

4.1.3 Adding the Furthest Point

Points outside $\text{MEB}(\mathcal{S}_t)$ have zero α_i's (Section 4.1.1) and so violate the KKT conditions of the dual problem. As in (Osuna et al., 1997b), one can simply add any such violating point to $\mathcal{S}_t$. Our step 3, however, takes a greedy approach by including the point furthest away from the current center. In the one-class classification case (Section 3.2.1),

$$\begin{aligned} \arg \max_{\mathbf{z}_\ell \notin B(\mathbf{c}_t, (1+\varepsilon)R_t)} \|\mathbf{c}_t - \tilde{\varphi}(\mathbf{z}_\ell)\|^2 &= \arg \min_{\mathbf{z}_\ell \notin B(\mathbf{c}_t, (1+\varepsilon)R_t)} \sum_{\mathbf{z}_i \in \mathcal{S}_t} \alpha_i \tilde{k}(\mathbf{z}_i, \mathbf{z}_\ell) \\ &= \arg \min_{\mathbf{z}_\ell \notin B(\mathbf{c}_t, (1+\varepsilon)R_t)} \sum_{\mathbf{z}_i \in \mathcal{S}_t} \alpha_i k(\mathbf{x}_i, \mathbf{x}_\ell) \\ &= \arg \min_{\mathbf{z}_\ell \notin B(\mathbf{c}_t, (1+\varepsilon)R_t)} \mathbf{w}'\varphi(\mathbf{x}_\ell), \end{aligned} \tag{19}$$

on using (9), (11) and (18). Similarly, in the binary classification case (Section 3.2.2), we have

$$\begin{aligned} \arg \max_{\mathbf{z}_\ell \notin B(\mathbf{c}_t, (1+\varepsilon)R_t)} \|\mathbf{c}_t - \tilde{\varphi}(\mathbf{z}_\ell)\|^2 &= \arg \min_{\mathbf{z}_\ell \notin B(\mathbf{c}_t, (1+\varepsilon)R_t)} \sum_{\mathbf{z}_i \in \mathcal{S}_t} \alpha_i y_i y_\ell (k(\mathbf{x}_i, \mathbf{x}_\ell) + 1) \\ &= \arg \min_{\mathbf{z}_\ell \notin B(\mathbf{c}_t, (1+\varepsilon)R_t)} y_\ell(\mathbf{w}'\varphi(\mathbf{x}_\ell) + b), \end{aligned} \tag{20}$$

on using (14) and (17). Hence, in both cases, step 3 chooses the *worst* violating pattern corresponding to the constraint ((7) and (12) respectively).

Also, as the dual objective in (10) has gradient $-2\tilde{\mathbf{K}}\alpha$, so for a pattern ℓ currently outside the ball

$$(\tilde{\mathbf{K}}\alpha)_\ell = \sum_{i=1}^{m} \alpha_i \left(k(\mathbf{x}_i, \mathbf{x}_\ell) + \frac{\delta_{i\ell}}{C} \right) = \mathbf{w}'\varphi(\mathbf{x}_\ell),$$

on using (9), (11) and $\alpha_\ell = 0$. Thus, the pattern chosen in (19) also makes the most progress towards maximizing the dual objective. This is also true for the two-class L2-SVM, as

$$(\tilde{\mathbf{K}}\alpha)_\ell = \sum_{i=1}^{m} \alpha_i \left(y_i y_\ell k(\mathbf{x}_i, \mathbf{x}_\ell) + y_i y_\ell + \frac{\delta_{i\ell}}{C} \right) = y_\ell(\mathbf{w}'\varphi(\mathbf{x}_\ell) + b),$$

on using (14), (17) and $\alpha_\ell = 0$. This subset selection heuristic is also commonly used by decomposition algorithms (Chang and Lin, 2004; Joachims, 1999; Platt, 1999).

4.1.4 Finding the MEB

At each iteration of Step 4, we find the MEB by using the QP formulation in Section 3.2. As the size $|\mathcal{S}_t|$ of the core set is much smaller than m in practice (Section 6), the computational complexity of each QP sub-problem is much lower than solving the whole QP. Besides, as only one core vector is added at each iteration, efficient rank-one update procedures (Cauwenberghs and Poggio, 2001; Vishwanathan et al., 2003) can also be used. The cost then becomes quadratic rather than cubic. As will be demonstrated in Section 6, the size of the core set is usually small to medium even for very large data sets. Hence, SMO is chosen in our implementation as it is often very efficient (in terms of both time and space) on data sets of such sizes. Moreover, as only one point is added each time, the new QP is just a slight perturbation of the original. Hence, by using the MEB solution obtained from the previous iteration as starting point (*warm start*), SMO can often converge in a small number of iterations.

4.2 Convergence to (Approximate) Optimality

First, consider $\varepsilon = 0$. The convergence proof in Bădoiu and Clarkson (2002) does not apply as it requires $\varepsilon > 0$. But as the number of core vectors increases in each iteration and the training set size is finite, so CVM must terminate in a finite number (say, τ) of iterations, With $\varepsilon = 0$, $\text{MEB}(\mathcal{S}_\tau)$ is an enclosing ball for all the ($\tilde{\varphi}$-transformed) points on termination. Because $\mathcal{S}_\tau$ is a subset of the whole training set and the MEB of a subset cannot be larger than the MEB of the whole set. Hence, $\text{MEB}(\mathcal{S}_\tau)$ must also be the exact MEB of the whole ($\tilde{\varphi}$-transformed) training set. In other words, when $\varepsilon = 0$, CVM outputs the *exact* solution of the kernel problem.

When $\varepsilon > 0$, we can still obtain an approximately optimal dual objective as follows. Assume that the algorithm terminates at the τth iteration, then

$$R_\tau \leq r_{\text{MEB}(\mathcal{S})} \leq (1+\varepsilon)R_\tau \tag{21}$$

by definition. Recall that the optimal primal objective p^* of the kernel problem in Section 3.2.1 (or 3.2.2) is equal to the optimal dual objective d_2^* in (10) (or (16)), which in turn is related to the optimal dual objective $d_1^* = r^2_{\text{MEB}(\mathcal{S})}$ in (2) by (6). Together with (21), we can then bound p^* as

$$R_\tau^2 \leq p^* + \tilde{\kappa} \leq (1+\varepsilon)^2 R_\tau^2. \tag{22}$$

Hence, $\max\left(\frac{R_\tau^2}{p^*+\tilde{\kappa}}, \frac{p^*+\tilde{\kappa}}{R_\tau^2}\right) \leq (1+\varepsilon)^2$ and thus CVM is an $(1+\varepsilon)^2$-approximation algorithm. This also holds with high probability[1] when probabilistic speedup is used.

As mentioned in Section 1, practical SVM implementations also output approximated solutions only. Typically, a parameter similar to our ε is required at termination. For example, in SMO, SVM^{light} and SimpleSVM, training stops when the KKT conditions are fulfilled within ε. Experience with these softwares indicate that near-optimal solutions are often good enough in practical applications. It can be shown that when CVM terminates, all the training patterns also satisfy similar loose KKT conditions. Here, we focus on the binary classification case. Now, at any iteration t, each training point falls into one of the following three categories:

1. Core vectors: Obviously, they satisfy the loose KKT conditions as they are involved in the QP.

2. Non-core vectors inside/on the ball $B(\mathbf{c}_t, R_t)$: Their α_i's are zero[2] and so the KKT conditions are satisfied.

3. Points lying outside $B(\mathbf{c}_t, R_t)$: Consider one such point ℓ. Its α_ℓ is zero (by initialization) and

$$\begin{aligned}\|\mathbf{c}_t - \tilde{\varphi}(\mathbf{z}_\ell)\|^2 &= \sum_{\mathbf{z}_i,\mathbf{z}_j \in \mathcal{S}_t} \alpha_i\alpha_j \left(y_i y_j k(\mathbf{x}_i,\mathbf{x}_j) + y_i y_j + \frac{\delta_{ij}}{C}\right) \\ &\quad -2\sum_{\mathbf{z}_i \in \mathcal{S}_t} \alpha_i \left(y_i y_\ell k(\mathbf{x}_i,\mathbf{x}_\ell) + y_i y_\ell + \frac{\delta_{i\ell}}{C}\right) + \tilde{k}(\mathbf{z}_\ell,\mathbf{z}_\ell) \\ &= \rho_t + \tilde{\kappa} - 2y_\ell(\mathbf{w}_t'\varphi(\mathbf{x}_\ell) + b_t),\end{aligned} \tag{23}$$

on using (14), (15), (17) and (18). This leads to

$$R_t^2 = \tilde{\kappa} - \rho_t. \tag{24}$$

[1] Obviously, the probability increases with the number of points subsampled and is equal to one when all the points are used. Obtaining a precise probability statement will be studied in future research.

[2] Recall that all the α_i's (except those of the two initial core vectors) are initialized to zero.

on using (3), (15) and (16). As $\mathbf{z}_\ell$ is inside/on the $(1+\varepsilon)$-ball at the τth iteration, $\|\mathbf{c}_\tau - \tilde{\varphi}(\mathbf{z}_\ell)\|^2 \leq (1+\varepsilon)^2 R_\tau^2$. Hence, from (23) and (24),

$$\begin{aligned} (1+\varepsilon)^2(\tilde{\kappa}-\rho_\tau) &\geq \rho_\tau + \tilde{\kappa} - 2y_\ell(\mathbf{w}_\tau'\varphi(\mathbf{x}_\ell) + b_\tau) \\ \Rightarrow \quad 2y_\ell(\mathbf{w}_\tau'\varphi(\mathbf{x}_\ell) + b_\tau) &\geq \rho_\tau + \tilde{\kappa} - (1+\varepsilon)^2(\tilde{\kappa}-\rho_\tau) \\ &\geq 2\rho_\tau - (2\varepsilon+\varepsilon^2)(\tilde{\kappa}-\rho_\tau) \\ \Rightarrow \quad y_\ell(\mathbf{w}_\tau'\varphi(\mathbf{x}_\ell) + b_\tau) - \rho_\tau &\geq -\left(\varepsilon + \frac{\varepsilon^2}{2}\right) R_\tau^2. \end{aligned} \tag{25}$$

Obviously, $R_\tau^2 \leq \tilde{k}(\mathbf{z},\mathbf{z}) = \tilde{\kappa}$. Hence, (25) reduces to

$$y_\ell(\mathbf{w}_\tau'\varphi(\mathbf{x}_\ell) + b_\tau) - \rho_\tau \geq -\left(\varepsilon + \frac{\varepsilon^2}{2}\right)\tilde{\kappa} \equiv -\varepsilon_2,$$

which is a loose KKT condition on pattern ℓ (which has $\alpha_\ell = 0$ and consequently $\xi_\ell = 0$ by (14)).

4.3 Time and Space Complexities

Existing decomposition algorithms cannot guarantee the number of iterations and consequently the overall time complexity (Chang and Lin, 2004). In this Section, we show how this can be obtained for CVM. In the following, we assume that a plain QP implementation, which takes $O(m^3)$ time and $O(m^2)$ space for m patterns, is used for the QP sub-problem in step 4. The time and space complexities obtained below can be further improved if more efficient QP solvers were used. Moreover, each kernel evaluation is assumed to take constant time.

Consider first the case where probabilistic speedup is not used in Section 4.1.2. As proved in (Bădoiu and Clarkson, 2002), CVM converges in at most $2/\varepsilon$ iterations. In other words, the total number of iterations, and consequently the size of the final core set, are of $\tau = O(1/\varepsilon)$. In practice, it has often been observed that the size of the core set is much smaller than this worst-case theoretical upper bound[3] (Kumar et al., 2003). As only one core vector is added at each iteration, $|S_t| = t+2$. Initialization takes $O(m)$ time while distance computations in steps 2 and 3 take $O((t+2)^2 + tm) = O(t^2 + tm)$ time. Finding the MEB in step 4 takes $O((t+2)^3) = O(t^3)$ time, and the other operations take constant time. Hence, the tth iteration takes a total of $O(tm + t^3)$ time. The overall time for $\tau = O(1/\varepsilon)$ iterations is

$$T = \sum_{t=1}^{\tau} O(tm + t^3) = O(\tau^2 m + \tau^4) = O\left(\frac{m}{\varepsilon^2} + \frac{1}{\varepsilon^4}\right),$$

which is *linear* in m for a fixed ε.

Next, we consider its space complexity. As the m training patterns may be stored outside the core memory, the $O(m)$ space required will be ignored in the following. Since only the core vectors are involved in the QP, the space complexity for the tth iteration is $O(|\mathcal{S}_t|^2)$. As $\tau = O(1/\varepsilon)$, the space complexity for the whole procedure is $O(1/\varepsilon^2)$, which is *independent* of m for a fixed ε.

On the other hand, when probabilistic speedup is used, initialization only takes $O(1)$ time while distance computations in steps 2 and 3 take $O((t+2)^2) = O(t^2)$ time. Time for the other operations

[3] This will also be corroborated by our experiments in Section 6.

remains the same. Hence, the tth iteration takes $O(t^3)$ time. As probabilistic speeedup may not find the furthest point in each iteration, τ may be larger than $2/\varepsilon$ though it can still be bounded by $O(1/\varepsilon^2)$ (Bădoiu et al., 2002). Hence, the whole procedure takes

$$T = \sum_{t=1}^{\tau} O(t^3) = O(\tau^4) = O\left(\frac{1}{\varepsilon^8}\right).$$

For a fixed ε, it is thus *independent* of m. The space complexity, which depends only on the number of iterations τ, becomes $O(1/\varepsilon^4)$.

When ε decreases, the CVM solution becomes closer to the exact optimal solution, but at the expense of higher time and space complexities. Such a tradeoff between efficiency and approximation quality is typical of all approximation schemes. Moreover, be cautioned that the O-notation is used for studying the asymptotic efficiency of algorithms. As we are interested in handling very large data sets, an algorithm that is asymptotically more efficient (in time and space) will be the best choice. However, on smaller problems, this may be outperformed by algorithms that are not as efficient asymptotically. These will be demonstrated experimentally in Section 6.

5. Related Work

The core set in CVM plays a similar role as the working set in other decomposition algorithms, and so these algorithms will be reviewed briefly in this Section. Following the convention in (Chang and Lin, 2004; Osuna et al., 1997b), the working set will be denoted B while the remaining subset of training patterns denoted N.

Chunking (Vapnik, 1998) is the first decomposition method used in SVM training. It starts with a random subset (*chunk*) of data as B and train an initial SVM. Support vectors in the chunk are retained while non-support vectors are replaced by patterns in N violating the KKT conditions. Then, the SVM is re-trained and the whole procedure repeated. Chunking suffers from the problem that the entire set of support vectors that have been identified will still need to be trained together at the end of the training process.

Osuna et al. (1997a) proposed another decomposition algorithm that fixes the size of the working set B. At each iteration, variables corresponding to patterns in N are frozen, while those in B are optimized in a QP sub-problem. After that, a new point in N violating the KKT conditions will replace some point in B. The SVM^{light} software (Joachims, 1999) follows the same scheme, though with a slightly different subset selection heuristic.

Going to the extreme, the sequential minimal optimization (SMO) algorithm (Platt, 1999) breaks the original, large QP into a series of smallest possible QPs, each involving only two variables. The first variable is chosen among points that violate the KKT conditions, while the second variable is chosen so as to have a large increase in the dual objective. This two-variable joint optimization process is repeated until the loose KKT conditions are fulfilled for all training patterns. By involving only two variables, SMO is advantageous in that each QP sub-problem can be solved analytically in an efficient way, without the use of a numerical QP solver. Moreover, as no matrix operations are involved, extra matrix storage is not required for keeping the kernel matrix. However, as each iteration only involves two variables in the optimization, SMO has slow convergence (Kao et al., 2004). Nevertheless, as each iteration is computationally simple, an overall speedup is often observed in practice.

Recently, Vishwanathan et al. (2003) proposed a related scale-up method called the SimpleSVM. At each iteration, a point violating the KKT conditions is added to the working set by using rank-one update on the kernel matrix. However, as pointed out in (Vishwanathan et al., 2003), storage is still a problem when the SimpleSVM is applied to large dense kernel matrices.

As discussed in Section 4.1, CVM is similar to these decomposition algorithms in many aspects, including initialization, subset selection and termination. However, subset selection in CVM is much simpler in comparison. Moreover, while decomposition algorithms allow training patterns to join and leave the working set multiple times, patterns once recruited as core vectors by the CVM will remain there for the whole training process. These allow the number of iterations, and consequently the time and space complexities, to be easily obtained for the CVM but not for the decomposition algorithms.

6. Experiments

In this Section, we implement the two-class L2-SVM in Section 3.2.2 and illustrate the scaling behavior of CVM (in C++) on several toy and real-world data sets. Table 1 summarizes the characteristics of the data sets used. For comparison, we also run the following SVM implementations:[4]

1. L2-SVM: LIBSVM implementation (in C++);
2. L2-SVM: LSVM implementation (in MATLAB), with low-rank approximation (Fine and Scheinberg, 2001) of the kernel matrix added;
3. L2-SVM: RSVM (Lee and Mangasarian, 2001) implementation (in MATLAB). The RSVM addresses the scale-up issue by solving a smaller optimization problem that involves a random $\bar{m} \times m$ rectangular subset of the kernel matrix. Here, $\bar{m}$ is set to 10% of m;
4. L1-SVM: LIBSVM implementation (in C++);
5. L1-SVM: SimpleSVM (Vishwanathan et al., 2003) implementation (in MATLAB).

Parameters are used in their default settings unless otherwise specified. Since our focus is on nonlinear kernels, we use the Gaussian kernel $k(\mathbf{x},\mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|^2/\beta)$ with $\beta = \frac{1}{m^2}\sum_{i,j=1}^{m}\|\mathbf{x}_i - \mathbf{x}_j\|^2$ unless otherwise specified. Experiments are performed on Pentium–4 machines running Windows XP. Detailed machine configurations will be reported in each section.

Our CVM implementation is adapted from the LIBSVM, and uses SMO for solving each QP sub-problem in step 4. As discussed in Section 4.1.4, warm start is used to initialize each QP sub-problem. Besides, as in LIBSVM, our CVM uses caching (with the same cache size as in the other LIBSVM implementations above) and stores all the training patterns in main memory. For simplicity, shrinking (Joachims, 1999) is not used in our current CVM implementation. Besides, we employ the probabilistic speedup method[5] as discussed in Section 4.1.2. On termination, we perform the (probabilistic) test in step 2 a few times so as to ensure that almost all the points have been covered by the $(1+\varepsilon)$-ball. The value of ε is fixed at 10^{-6} in all the experiments. As in other

[4] Our CVM implementation can be downloaded from http://www.cs.ust.hk/∼jamesk/cvm.zip. LIBSVM can be downloaded from http://www.csie.ntu.edu.tw/∼cjlin/libsvm/; LSVM from http://www.cs.wisc.edu/dmi/lsvm; and SimpleSVM from http://asi.insa-rouen.fr/∼gloosli/. Moreover, we followed http://www.csie.ntu.edu.tw/∼cjlin/libsvm/faq.html in adapting the LIBSVM package for L2-SVM.

[5] Following (Smola and Schölkopf, 2000), a random sample of size 59 is used.

date set	max training set size	# attributes
checkerboard	1,000,000	2
forest cover type	522,911	54
extended USPS digits	266,079	676
extended MIT face	889,986	361
KDDCUP-99 intrusion detection	4,898,431	127
UCI adult	32,561	123

Table 1: Data sets used in the experiments.

decomposition methods, the use of a very stringent stopping criterion is not necessary in practice. Preliminary studies show that $\varepsilon = 10^{-6}$ is acceptable for most tasks. Using an even smaller ε does not show improved generalization performance, but may increase the training time unnecessarily.

6.1 Checkerboard Data

We first experiment on the 4×4 checkerboard data (Figure 2) commonly used for evaluating large-scale SVM implementations (Lee and Mangasarian, 2001; Mangasarian and Musicant, 2001b; Schwaighofer and Tresp, 2001). We use training sets with a maximum of 1 million points and 2,000 independent points for testing. Of course, this problem does not need so many points for training, but it is convenient for illustrating the scaling properties. Preliminary study suggests a value of $C = 1000$. A 3.2GHz Pentium–4 machine with 512MB RAM is used.

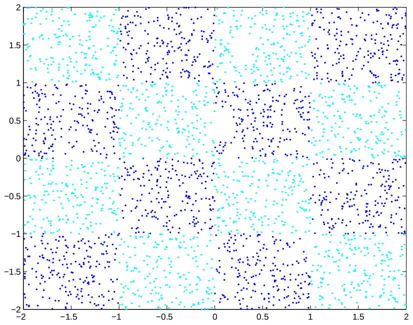


Figure 2: The 4×4 checkerboard data set.

Experimentally, L2-SVM with low rank approximation does not yield satisfactory performance on this data set, and so its result is not reported here. RSVM, on the other hand, has to keep a rectangular kernel matrix of size $\bar{m} \times m$ ($\bar{m}$ being the number of random samples used), and cannot be run on our machine when m exceeds 10K. Similarly, the SimpleSVM has to store the kernel matrix of the active set, and runs into storage problem when m exceeds 30K.

Results are shown in Figure 3. As can be seen, CVM is as accurate as the other implementations. Besides, it is much faster[6] and produces far fewer support vectors (which implies faster testing) on large data sets. In particular, one million patterns can be processed in under 13 seconds. On the other hand, for relatively small training sets, with less than 10K patterns, LIBSVM is faster. This, however, is to be expected as LIBSVM uses more sophisticated heuristics and so will be more efficient on small-to-medium sized data sets. Figure 3(b) also shows the core set size, which can be seen to be small and its curve basically overlaps with that of the CVM. Thus, almost all the core vectors are useful support vectors. Moreover, it also confirms our theoretical findings that both time and space required are constant w.r.t. the training set size, when it becomes large enough.

6.2 Forest Cover Type Data

The forest cover type data set[7] has been used for large scale SVM training (e.g., (Bakir et al., 2005; Collobert et al., 2002)). Following (Collobert et al., 2002), we aim at separating class 2 from the other classes. 1% – 90% of the whole data set (with a maximum of 522,911 patterns) are used for training while the remaining are used for testing. We use the Gaussian kernel with $\beta = 10000$ and $C = 10000$. Experiments are performed on a 3.2GHz Pentium–4 machine with 512MB RAM.

Preliminary studies show that the number of support vectors is over ten thousands. Consequently, RSVM and SimpleSVM cannot be run on our machine. Similarly, for low rank approximation, preliminary studies show that over thousands of basis vectors are required for a good approximation. Therefore, only the two LIBSVM implementations will be compared with the CVM here.

As can be seen from Figure 4, CVM is, again, as accurate as the others. Note that when the training set is small, more training patterns bring in additional information useful for classification and so the number of core vectors increases with training set size. However, after processing around 100K patterns, both the time and space requirements of CVM begin to exhibit a constant scaling with the training set size. With hindsight, one might simply sample 100K training patterns and hope to obtain comparable results.[8] However, for satisfactory classification performance, different problems require samples of different sizes and CVM has the important advantage that the required sample size does not have to be pre-specified. Without such prior knowledge, random sampling gives poor testing results, as demonstrated in (Lee and Mangasarian, 2001).

6.3 Extended USPS Digits Data

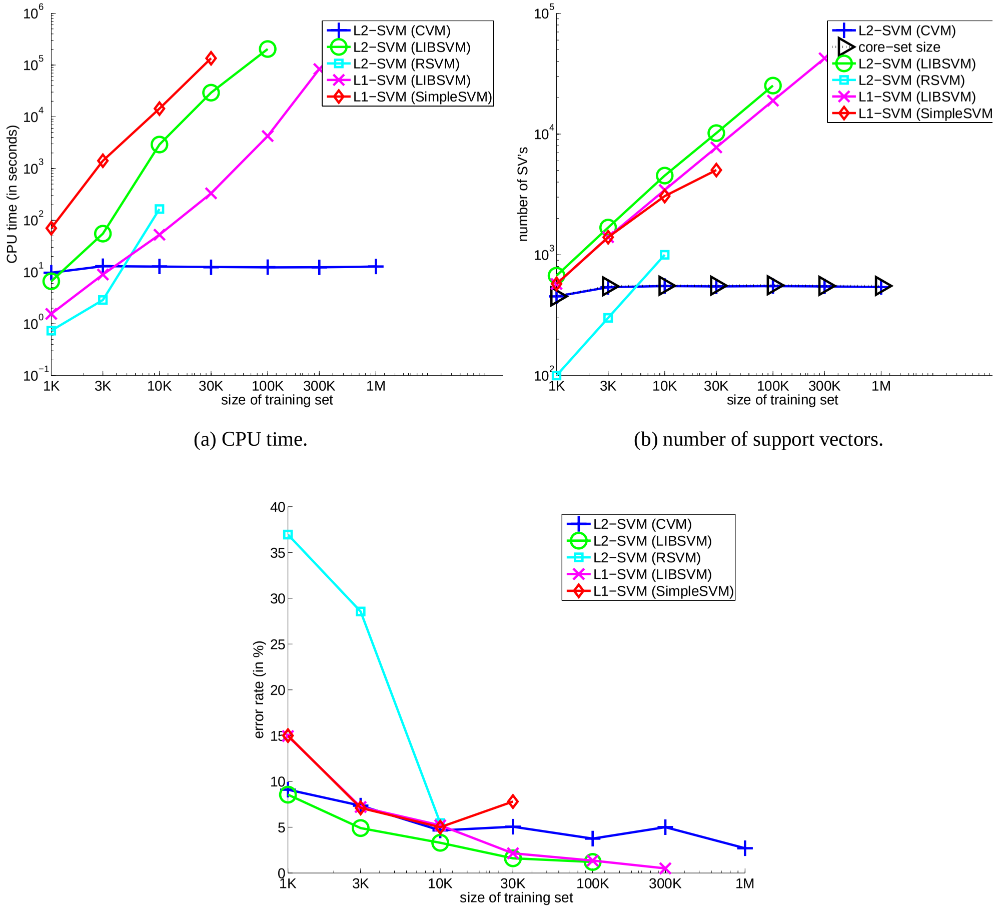
In this experiment, our task is to classify digits zero from one in an extended version of the USPS data set.[9] The original training set has 1,005 zeros and 1,194 ones, while the test set has 359 zeros and 264 ones. To better study the scaling behavior, we extend this data set by first converting the resolution from 16×16 to 26×26, and then generate new images by shifting the original ones in all directions for up to five pixels. Thus, the resultant training set has a total of $(1005 + 1194) \times 11^2 =$

[6] The CPU time only measures the time for training the SVM. Time for reading the training patterns into main memory is not included. Moreover, as some implementations are in MATLAB, so not all the CPU time measurements can be directly compared. However, it is still useful to note the constant scaling exhibited by the CVM and its speed advantage over other C++ implementations, when the data set is large.

[7] http://kdd.ics.uci.edu/databases/covertype/covertype.html

[8] In fact, we tried both LIBSVM implementations on a random sample of 100K training patterns, but their testing accuracies are inferior to that of CVM.

[9] http://www.kernel-machines.org/data/usps.mat.gz

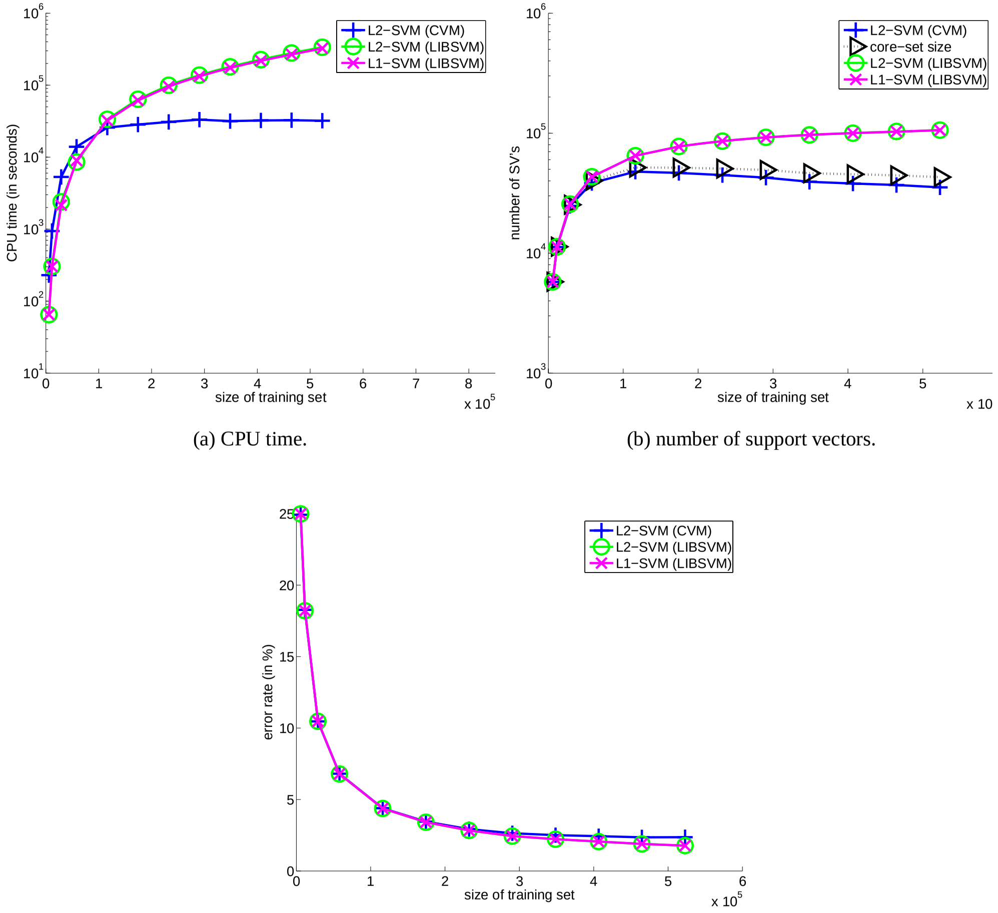


(a) CPU time.

(b) number of support vectors.

(c) testing error.

Figure 3: Results on the checkerboard data set (Except for the CVM, the other implementations have to terminate early because of not enough memory and/or the training time is too long). Note that the CPU time, number of support vectors, and size of the training set are in log scale.



(a) CPU time.

(b) number of support vectors.

(c) testing error.

Figure 4: Results on the forest cover type data set. Note that the CPU time and number of support vectors are in log scale.

266,079 patterns while the extended test set has $(359+264)\times 11^2 = 753,83$ patterns. In this experiment, $C = 100$ and a 3.2GHz Pentium–4 machine with 512MB RAM is used.

As can be seen from Figure 5, the behavior of CVM is again similar to those in the previous sections. In particular, both the time and space requirements of CVM increase when the training set is small. They then stabilize at around 30K patterns and CVM becomes faster than the other decomposition algorithms.

6.4 Extended MIT Face Data

In this Section, we perform face detection using an extended version of the MIT face database[10] (Heisele et al., 2000; Sung, 1996). The original data set has 6,977 training images (with 2,429 faces and 4,548 nonfaces) and 24,045 test images (472 faces and 23,573 nonfaces). The original 19×19 grayscale images are first enlarged to 21×21. To better study the scaling behavior of various SVM implementations, we again enlarge the training set by generating artificial samples. As in (Heisele et al., 2000; Osuna et al., 1997b), additional nonfaces are extracting from images that do not contain faces (e.g., images of landscapes, trees, buildings, etc.). As for the set of faces, we enlarge it by applying various image transformations (including blurring, flipping and rotating) to the original faces. The following three training sets are thus created (Table 6.4):

1. Set A: This is obtained by adding 477,366 nonfaces to the original training set, with the nonface images extracted from 100 photos randomly collected from the web.

2. Set B: Each training face is blurred by the arithmetic mean filter (with window sizes 2×2, 3×3 and 4×4, respectively) and added to set A. They are then flipped laterally, leading to a total of $2429\times 4\times 2 = 19,432$ faces.

3. Set C: Each face in set B is rotated between -20^o and 20^o, in increments of 2^o. This results in a total of $19432\times 21 = 408,072$ faces.

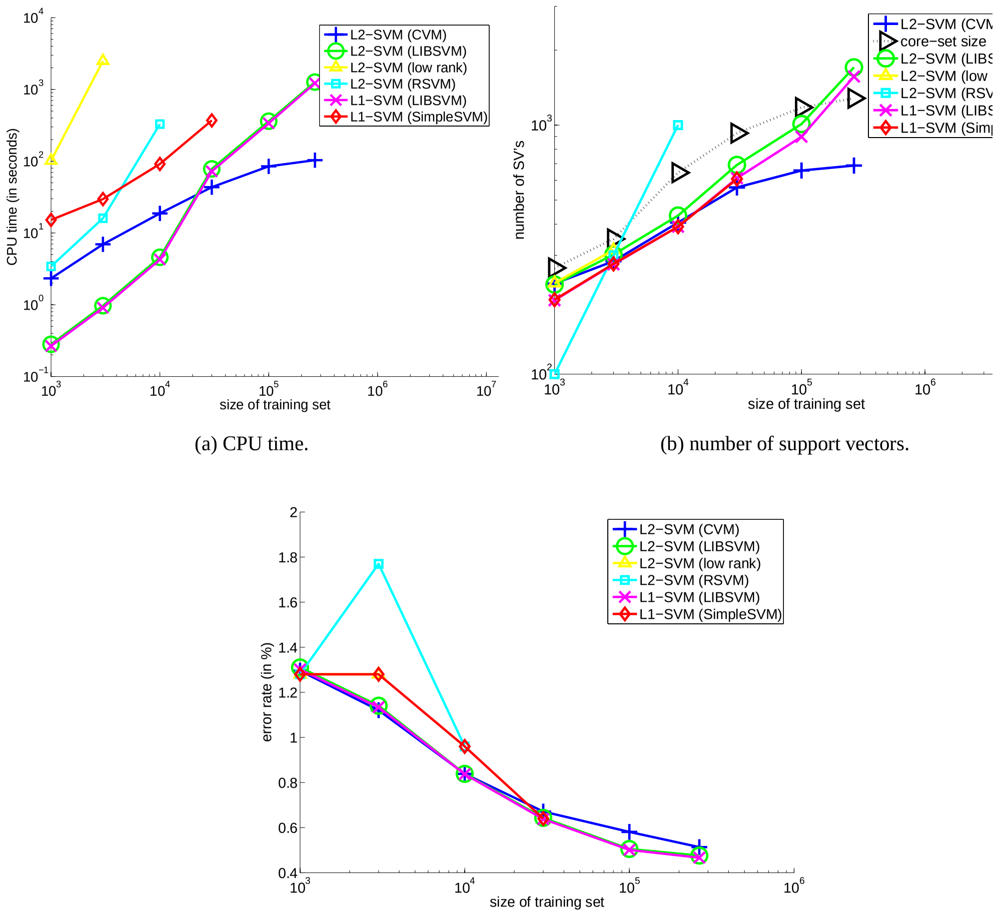
In this experiment, $C = 20$ and a 2.5GHz Pentium–4 machine with 1GB RAM is used. Moreover, a dense data format, which is more appropriate for this data set, is used in all the implementations. Recall that the intent of this experiment is on studying the scaling behavior rather than on obtaining state-of-the-art face detection performance. Nevertheless, the ability of CVM in handling very large data sets could make it a better base classifier in powerful face detection systems such as the boosted cascade (Viola and Jones, 2001).

training set	# faces	# nonfaces	total
original	2,429	4,548	6,977
set A	2,429	481,914	484,343
set B	19,432	481,914	501,346
set C	408,072	481,914	889,986

Table 2: Number of faces and nonfaces in the face detection data sets.

Because of the imbalanced nature of this data set, the testing error is inappropriate for performance evaluation here. Instead, we will use the AUC (area under the ROC curve), which has been

[10] http://cbcl.mit.edu/cbcl/software-datasets/FaceData2.html



(a) CPU time.

(b) number of support vectors.

(c) testing error.

Figure 5: Results on the extended USPS digits data set (Except for the CVM, the other implementations have to terminate early because of not enough memory and/or the training time is too long). Note that the CPU time, number of support vectors, and size of the training set are in log scale.

commonly used for face detectors. The ROC (receiver operating characteristic) curve (Bradley, 1997) plots TP on the Y-axis and the false positive rate

$$\text{FP} = \frac{\text{negatives incorrectly classified}}{\text{total negatives}}$$

on the X-axis. Here, faces are treated as positives while non-faces as negatives. The AUC is always between 0 and 1. A perfect face detector will have unit AUC, while random guessing will have an AUC of 0.5. Another performance measure that will be reported is the balanced loss (Weston et al., 2002)

$$\ell_{bal} = 1 - \frac{\text{TP} + \text{TN}}{2},$$

which is also suitable for imbalanced data sets. Here,

$$\text{TP} = \frac{\text{positives correctly classified}}{\text{total positives}}, \quad \text{TN} = \frac{\text{negatives correctly classified}}{\text{total negatives}},$$

are the true positive and true negative rates respectively.

The ROC on using CVM is shown in Figure 6, which demonstrates the usefulness of using extra faces and nonfaces in training. This is also reflected in Figure 7, which shows that the time and space requirements of CVM are increasing with larger training sets. Even in this non-asymptotic case, the CVM still significantly outperforms both LIBSVM implementations in terms of training time and number of support vectors, while the values of AUC and ℓ_{bal} are again very competitive. Note also that the LIBSVM implementations of both L1- and L2-SVMs do not perform well (in terms of ℓ_{bal}) on the highly imbalanced set A. On the other hand, CVM shows a steady improvement and is less affected by the skewed distribution. In general, the performance of SVMs could be improved by assigning different penalty parameters (C's) to the classes. A more detailed study on the use of CVM in an imbalanced setting will be conducted in the future.

6.5 KDDCUP-99 Intrusion Detection Data

This intrusion detection data set[11] has been used for the Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99. The training set contains 4,898,431 connection records, which are processed from about four gigabytes of compressed binary TCP dump data from seven weeks of network traffic. Another two weeks of data produced the test data with 311,029 patterns. The data set includes a wide variety of intrusions simulated in a military network environment. There are a total of 24 training attack types, and an additional 14 types that appear in the test data only.

We follow the same setup in (Yu et al., 2003). The task is to separate normal connections from attacks. Each original pattern has 34 continuous features and 7 symbolic features. We normalize each continuous feature to the range zero and one, and transform each symbolic feature to multiple binary features. Yu et al. (2003) used the clustering-based SVM (CB-SVM), which incorporates the hierarchical micro-clustering algorithm BIRCH (Zhang et al., 1996) to reduce the number of patterns in SVM training. However, CB-SVM is restricted to the use of linear kernels. In our experiments with the CVM, we will continue the use of the Gaussian kernel (with $\beta = 1000$ and $C = 10^6$) as in the previous sections. Moreover, as the whole data set is stored in the core in our current implementation, we use a 3.2GHz Pentium–4 machine with 2GB RAM.

[11] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

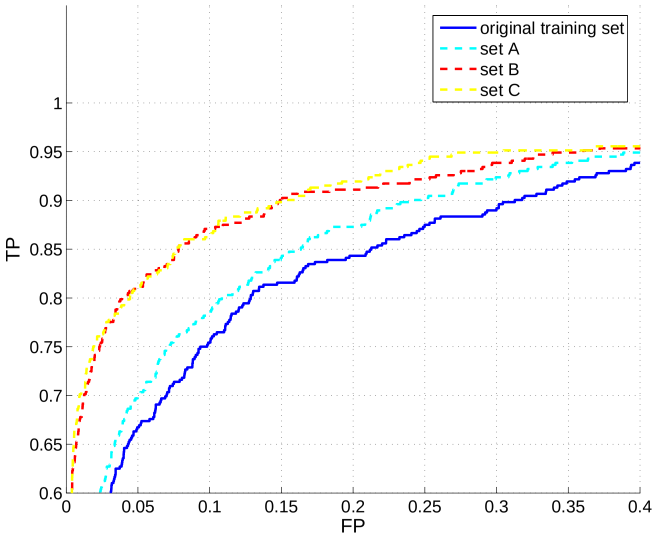


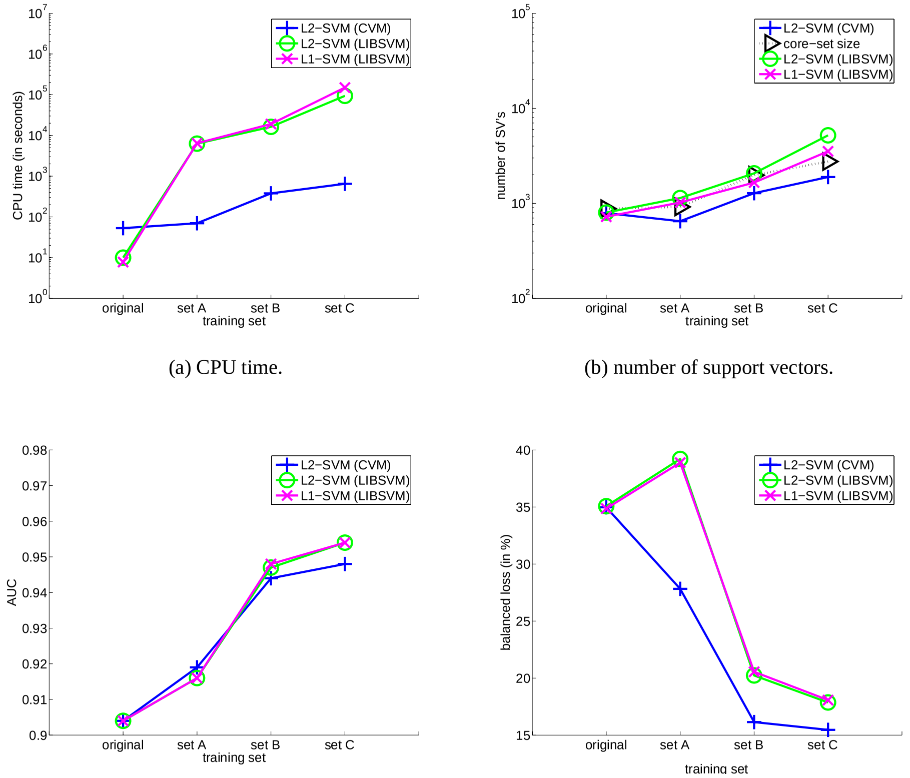
Figure 6: ROC of the extended MIT face data set on using CVM.

Table 6.5 compares the results of CVM with those reported in (Yu et al., 2003), which include SVMs using random sampling, active learning (Schohn and Cohn, 2000) and CB-SVM. Surprisingly, our CVM on the whole training set (which has around five million patterns) takes only 1.4 seconds, and yields a lower testing error than all other methods. The performance of CVM on this data set, as evaluated by some more measures and the ROC, are reported in Table 6.5 and Figure 8 respectively.

6.6 Relatively Small Data Sets: UCI Adult Data

Following (Platt, 1999), we use training sets[12] with up to 32,562 patterns. Experiments are performed with $C = 0.1$ and on a 3.2GHz Pentium–4 machine with 512MB RAM. As can be seen in Figure 9, CVM is still among the most accurate methods. However, as this data set is relatively small, more training patterns do carry more classification information. Hence, as discussed in Section 6.2, the number of iterations, the core set size and consequently the CPU time all increase with the number of training patterns. From another perspective, recall that the worst case core set size is $2/\varepsilon$, independent of m (Section 4.3). For the value of $\varepsilon = 10^{-6}$ used here, $2/\varepsilon = 2 \times 10^6$. Besides, although we have seen that the actual size of the core set is often much smaller than this worst case value, however, when $m \ll 2/\varepsilon$, the number of core vectors can still be dependent on m. More-

[12] ftp://ftp.ics.uci.edu/pub/machine-learning-databases/adult



(a) CPU time.

(b) number of support vectors.

(c) AUC.

(d) ℓ_{bal}.

Figure 7: Results on the extended MIT face data set. Note that the CPU time and number of support vectors are in log scale.

method		# training patterns input to SVM	# test errors	SVM training time (in sec)	other processing time (in sec)
random sampling	0.001%	47	25,713	0.000991	500.02
	0.01%	515	25,030	0.120689	502.59
	0.1%	4,917	25,531	6.944	504.54
	1%	49,204	25,700	604.54	509.19
	5%	245,364	25,587	15827.3	524.31
active learning		747	21,634	94192.213	
CB-SVM		4,090	20,938	7.639	4745.483
CVM		4,898,431	19,513	1.4	

Table 3: Results on the KDDCUP-99 intrusion detection data set by CVM and methods reported in (Yu et al., 2003). Here, "other processing time" refers to the (1) sampling time for SVM with random sampling; and (2) clustering time for CB-SVM. For SVM with active learning and CVM, the total training time required is reported. Note that Yu et al. (2003) used a 800MHz Pentium-3 machine with 906MB RAM while we use a 3.2GHz Pentium–4 machine with 2GB RAM. Hence, the time measurements are for reference only and cannot be directly compared.

AUC	ℓ_{bal}	# core vectors	# support vectors
0.977	0.042	55	20

Table 4: More performance measures of CVM on the KDDCUP-99 intrusion detection data.

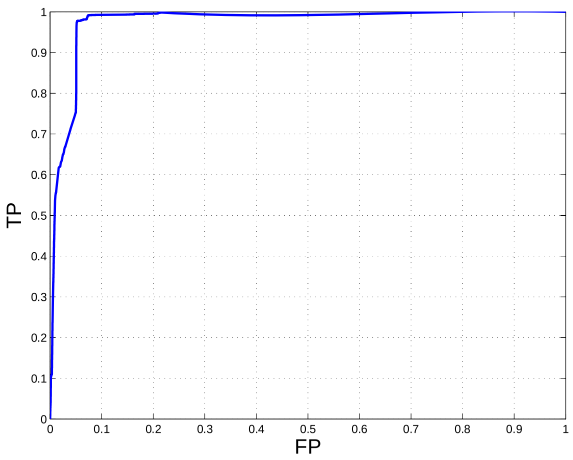
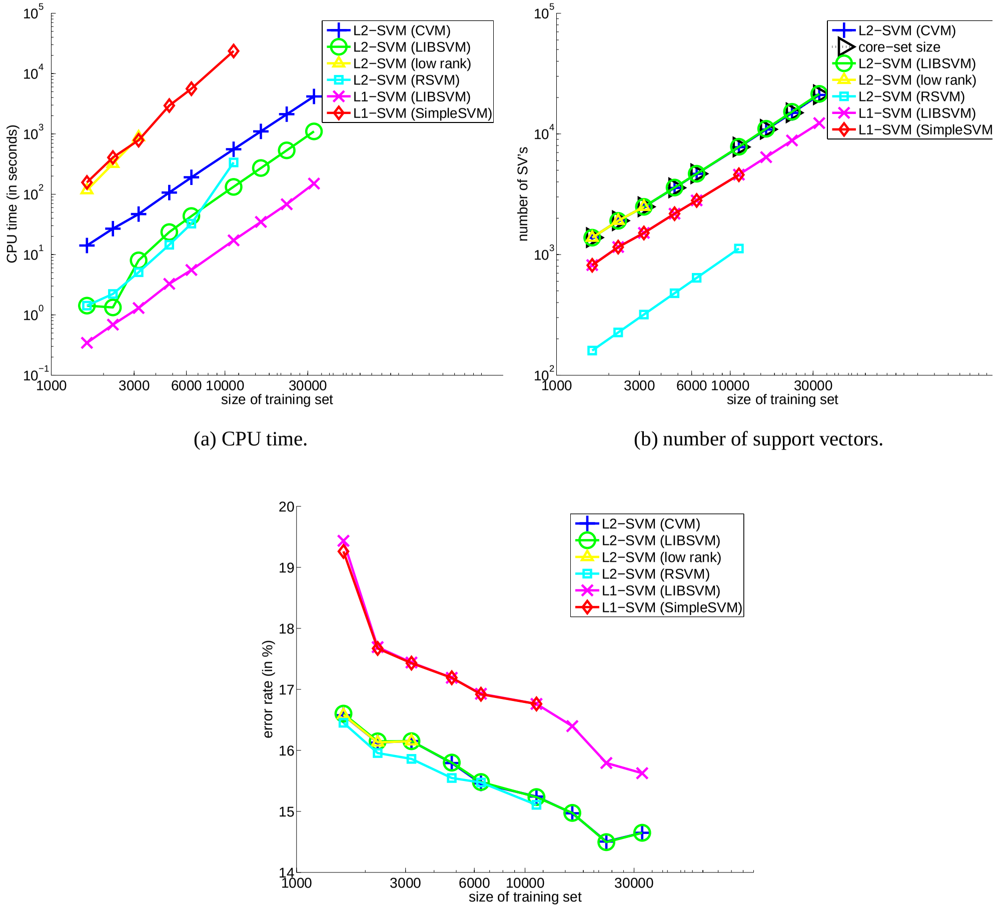


Figure 8: ROC of the the KDDCUP-99 intrusion detection data using CVM.

over, as has been observed in the previous sections, the CVM is slower than the more sophisticated LIBSVM on processing these smaller data sets.



(a) CPU time.

(b) number of support vectors.

(c) testing error.

Figure 9: Results on the UCI adult data set (The other implementations have to terminate early because of not enough memory and/or the training time is too long). Note that the CPU time, number of SV's and size of training set are in log scale.

7. Conclusion

In this paper, we exploit the "approximateness" in practical SVM implementations to scale-up SVM training. We formulate kernel methods (including the soft-margin one-class and two-class SVMs) as equivalent MEB problems, and then obtain approximately optimal solutions efficiently with the use of core sets. The proposed CVM procedure is simple, and does not require sophisticated heuristics as in other decomposition methods. Moreover, despite its simplicity, CVM has small asymptotic time and space complexities. In particular, for a fixed ε, its asymptotic time complexity is *linear* in the training set size m while its space complexity is *independent* of m. This can be further improved when probabilistic speedup is used. Experimentally, it is as accurate as existing SVM implementations, but is much faster and produces far fewer support vectors (and thus faster testing) on large data sets. On the other hand, on relatively small data sets where $m \ll 2/\varepsilon$, SMO can be faster. Besides, although we have fixed the value of ε in the experiments, one could also vary the value of ε to adjust the tradeoff between efficiency and approximation quality. In general, with a smaller ε, the CVM solution becomes closer to the exact optimal solution, but at the expense of higher time and space complexities. Our experience suggests that a fixed value of $\varepsilon = 10^{-6}$ is acceptable for most tasks.

The introduction of CVM opens new doors for applying kernel methods to data-intensive applications involving very large data sets. The use of approximation algorithms also brings immense opportunities to scaling up other kernel methods. For example, we have obtained preliminary success in extending support vector regression using the CVM technique. In the future, we will also apply CVM-like approximation algorithms to other kernel-related learning problems such as imbalanced learning, ranking and clustering. The iterative recruitment of core vectors is also similar to incremental procedures (Cauwenberghs and Poggio, 2001; Fung and Mangasarian, 2002), and this connection will be further explored. Besides, although the CVM can obtain much fewer support vectors than standard SVM implementations on large data sets, the number of support vectors may still be too large for real-time testing. As the core vectors in CVM are added incrementally and never removed, it is thus possible that some of them might be redundant. We will consider post-processing methods to further reduce the number of support vectors. Finally, all the training patterns are currently stored in the main memory. We anticipate that even larger data sets can be handled, possibly with reduced speed, when traditional scale-up techniques such as out-of-core storage and low-rank approximation are also incorporated.

Acknowledgements

This research has been partially supported by the Research Grants Council of the Hong Kong Special Administrative Region. The author would also like to thank the anonymous reviewers for their constructive comments on an earlier version of this paper.

References

D. Achlioptas, F. McSherry, and B. Schölkopf. Sampling techniques for kernel methods. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

G. H. Bakir, J. Weston, and L. Bottou. Breaking SVM complexity with cross-training. In *Advances in Neural Information Processing Systems 17*, Cambridge, MA, 2005. MIT Press.

D. Boley and D. Cao. Training support vector machine using adaptive clustering. In *Proceedings of the SIAM International Conference on Data Mining*, pages 126–137, Lake Buena Vista, FL, USA, April 2004.

A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

M. Bădoiu and K. L. Clarkson. Optimal core sets for balls. In *DIMACS Workshop on Computational Geometry*, 2002.

M. Bădoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core sets. In *Proceedings of 34th Annual ACM Symposium on Theory of Computing*, pages 250–257, Montréal, Québec, Canada, 2002.

G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, Cambridge, MA, 2001. MIT Press.

T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, pages 300–309, Clear Water Bay, Hong Kong, 2000.

C.-C. Chang and C.-J. Lin. *LIBSVM: a Library for Support Vector Machines*, 2004. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159, 2002.

C. S. Chu, I. W. Tsang, and J. T. Kwok. Scaling up support vector data description by using core-sets. In *Proceedings of the International Joint Conference on Neural Networks*, pages 425–430, Budapest, Hungary, July 2004.

R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 14(5):1105–1114, May 2002.

S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, December 2001.

T. Friess, N. Cristianini, and C. Campbell. The Kernel-Adatron algorithm: a fast and simple learning procedure for support vector machines. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 188–196, Madison, Wisconsin, USA, July 1998.

G. Fung and O. L. Mangasarian. Incremental support vector machine classification. In R. Grossman, H. Mannila, and R. Motwani, editors, *Proceedings of the Second SIAM International Conference on Data Mining*, pages 247–260, Arlington, Virginia, USA, 2002.

G. Fung and O. L. Mangasarian. Finite Newton method for Lagrangian support vector machine classification. *Neurocomputing*, 55:39–55, 2003.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

S. Har-Peled and Y. Wang. Shape fitting with outliers. *SIAM Journal on Computing*, 33(2):269–285, 2004.

B. Heisele, T. Poggio, and M. Pontil. Face detection in still gray images. A.I. memo 1687, Center for Biological and Computational Learning, MIT, Cambridge, MA, 2000.

T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 169–184. MIT Press, Cambridge, MA, 1999.

W.-C. Kao, K.-M. Chung, C.-L. Sun, and C.-J. Lin. Decomposition methods for linear support vector machines. *Neural Computation*, 16:1689–1704, 2004.

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks*, 11(1):124–136, January 2000.

P. Kumar, J. S. B. Mitchell, and A. Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *ACM Journal of Experimental Algorithmics*, 8, January 2003.

Y.-J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. In *Proceeding of the First SIAM International Conference on Data Mining*, 2001.

O. L. Mangasarian and D. R. Musicant. Active set support vector machine classification. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 577–583, Cambridge, MA, 2001a. MIT Press.

O. L. Mangasarian and D. R. Musicant. Lagrangian support vector machines. *Journal of Machine Learning Research*, 1:161–177, 2001b.

N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12:759–776, 1983.

F. Nielsen and R. Nock. Approximating smallest enclosing balls. In *Proceedings of International Conference on Computational Science and Its Applications*, volume 3045, pages 147–157, 2004.

E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*, pages 276–285, Amelia Island, FL, USA, 1997a.

E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *Proceedings of Computer Vision and Pattern Recognition*, pages 130–136, San Juan, Puerto Rico, June 1997b.

D. Pavlov, D. Chudova, and P. Smyth. Towards scalable support vector machines using squashing. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 295–299, Boston, Massachusetts, USA, 2000a.

D. Pavlov, J. Mao, and B. Dom. Scaling-up support vector machines using boosting algorithm. In *Proceedings of the International Conference on Pattern Recognition*, volume 2, pages 2219–2222, Barcelona, Spain, September 2000b.

J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 185–208. MIT Press, Cambridge, MA, 1999.

F. P. Preparata. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

D. Roobaert. DirectSVM: a simple support vector machine perceptron. In *Proceedings of IEEE International Workshop on Neural Networks for Signal Processing*, pages 356–365, Sydney, Australia, December 2000.

G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 839–846, San Francisco, CA, USA, 2000. Morgan Kaufmann.

B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, July 2001.

B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

A. Schwaighofer and V. Tresp. The Bayesian committee support vector machine. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Proceedings of the International Conference on Artificial Neural Networks*, pages 411–417. Springer Verlag, 2001.

A. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 911–918, Stanford, CA, USA, June 2000.

A. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14 (3):199–222, August 2004.

K.-K. Sung. *Learning and Example Selection for Object and Pattern Recognition*. PhD thesis, Artificial Intelligence Laboratory and Center for Biological and Computational Learning, MIT, Cambridge, MA, 1996.

J. J. Sylvester. A question in the geometry of situation. *Quarterly Journal on Mathematics*, 1:79, 1857.

D. M. J. Tax and R. P. W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20 (14):1191–1199, 1999.

S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In *Proceedings of the 17th International Conference on Machine Learning*, pages 999–1006, San Francisco, CA, USA, 2000. Morgan Kaufmann.

V. Tresp. Scaling kernel-based systems to large data sets. *Data Mining and Knowledge Discovery*, 5(3):197–211, 2001.

I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Very large SVM training using core vector machines. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, Barbados, January 2005.

V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, volume 1, pages 1063–6919, 2001.

S. V. N. Vishwanathan, A. J. Smola, and M. N. Murty. SimpleSVM. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 760–767, Washington, D.C., USA, August 2003.

E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, pages 359–370. Springer-Verlag, 1991.

J. Weston, B. Schölkopf, E. Eskin, C. Leslie, and S. W. Noble. Dealing with large diagonals in kernel matrices. *Principles of Data Mining and Knowledge Discovery, Springer Lecture Notes in Computer Science 243*, 2002.

C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, Cambridge, MA, 2001. MIT Press.

C. Yang, R. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast Gauss transform. In *Advances in Neural Information Processing Systems 17*, Cambridge, MA, 2005. MIT Press.

H. Yu, J. Yang, and J. Han. Classifying large data sets using SVM with hierarchical clusters. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 306–315, Washington DC, USA, 2003.

T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In H. V. Jagadish and I. S. Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Quebec, Canada, June 1996. ACM Press.