Very Large SVM Training using Core Vector Machines

Ivor W. Tsang **James T. Kwok** **Pak-Ming Cheung**
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay
Hong Kong

Abstract

Standard SVM training has $O(m^3)$ time and $O(m^2)$ space complexities, where m is the training set size. In this paper, we scale up kernel methods by exploiting the "approximateness" in practical SVM implementations. We formulate many kernel methods as equivalent minimum enclosing ball problems in computational geometry, and then obtain provably approximately optimal solutions efficiently with the use of core-sets. Our proposed Core Vector Machine (CVM) algorithm 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 much faster and can handle much larger data sets than existing scale-up methods. In particular, on our PC with only 512M RAM, the CVM with Gaussian kernel can process the checkerboard data set with 1 million points in less than 13 seconds.

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 & Seeger, 2001), greedy approximation (Smola & Schölkopf, 2000) or matrix decompositions (Fine & 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 or more sophisticated decomposition methods. 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. (1997) suggested optimizing only a fixed-size subset of the training data (*working set*) 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 a large QP into a series of smallest possible QPs, each involving only two variables. In the context of classification, Mangasarian and Musicant (2001) proposed the Lagrangian SVM (LSVM) that avoids the QP (or LP) altogether. Instead, the solution is obtained by 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. Further speed-up is possible by employing the reduced SVM (RSVM) (Lee & Mangasarian, 2001), which uses a rectangular subset of the kernel matrix. However, this may lead to performance degradation (Lin & Lin, 2003).

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.

Moreover, practical SVM implementations, as in many numerical routines, only *approximate* the optimal solution by an iterative strategy. Typically, the stopping criterion utilizes either the precision of the Lagrange multipliers (e.g., (Joachims, 1999; Platt, 1999)) or the duality gap (e.g., (Smola & Schölkopf, 2004)). However, while approximation algorithms (with provable performance guarantees) have been extensively used in tackling computationally dif-

ficult problems like NP-complete problems (Garey & Johnson, 1979), such "approximateness" has never been exploited in the design of SVM implementations.

In this paper, we first transform the SVM optimization problem (with a possibly nonlinear kernel) to 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 do not scale well with the dimensionality d of the points. Consequently, recent attention has shifted to the development of approximation algorithms. Lately, a breakthrough was obtained by Bădoiu and Clarkson (2002), who showed that an $(1+\epsilon)$-approximation of the MEB can be efficiently obtained using *core-sets*. Generally speaking, in an optimization problem, a core-set is a subset of the input points, such that we can get a good approximation (with an approximation ratio[1] specified by a user-defined ϵ parameter) to the original input by solving the optimization problem directly on the core-set. Moreover, a surprising property of (Bădoiu & Clarkson, 2002) is that the size of its core-set is *independent* of both d and the size of the point set.

Inspired from this core-set-based approximate MEB algorithm, we will develop an approximation algorithm for SVM training that has an approximation ratio of $(1+\epsilon)^2$. Its time complexity is linear in m while its space complexity is independent of m. 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. Experimental results are presented in Section 5, and the last section gives some concluding remarks.

2 MEB in Computational Geometry

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 MEB($\mathcal{S}$)) is the smallest ball that contains all the points in $\mathcal{S}$. The MEB problem has found applications in diverse areas such as computer graphics (e.g., collision detection, visibility culling), machine learning (e.g., similarity search) and facility locations problems.

[1] Let C be the cost (or value of the objective function) of the solution returned by an approximate algorithm, and C^* be the cost of the optimal solution. Then, the approximate algorithm has an *approximation ratio* $\rho(n)$ for an input size n if $\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$. Intuitively, this 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*.

Here, we will focus on approximate MEB algorithms based on core-sets. Let $B(\mathbf{c}, R)$ be the ball with center $\mathbf{c}$ and radius R. Given $\epsilon > 0$, a ball $B(\mathbf{c}, (1+\epsilon)R)$ is an *$(1+\epsilon)$-approximation* of MEB($\mathcal{S}$) if $R \leq r_{\text{MEB}(\mathcal{S})}$ and $\mathcal{S} \subset B(\mathbf{c}, (1+\epsilon)R)$. A subset $X \subseteq \mathcal{S}$ is a *core-set* of $\mathcal{S}$ if an expansion by a factor $(1+\epsilon)$ of its MEB contains $\mathcal{S}$, i.e., $\mathcal{S} \subset B(\mathbf{c}, (1+\epsilon)r)$, where $B(\mathbf{c}, r) = \text{MEB}(X)$ (Figure 1).

To obtain such an $(1+\epsilon)$-approximation, Bădoiu and Clarkson (2002) proposed 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 $(1+\epsilon)$-ball $B(\mathbf{c}_t, (1+\epsilon)r_t)$. This is repeated until all the points in $\mathcal{S}$ are covered by $B(\mathbf{c}_t, (1+\epsilon)r_t)$. Despite its simplicity, Bădoiu and Clarkson (2002) showed that the number of iterations, and hence the size of the final core-set, depends only on ϵ but *not* on d or m. This 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 independence on m, it allows both the time and space complexities of our algorithm to grow slowly, as will be shown in Section 4.3.

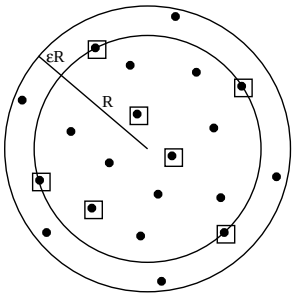


Figure 1: The inner circle is the MEB of the set of squares and its $(1+\epsilon)$ expansion (the outer circle) covers all the points. The set of squares is thus a core-set.

3 MEB Problems and Kernel Methods

Obviously, the MEB is equivalent to the hard-margin support vector data description (SVDD) (Tax & Duin, 1999), which will be briefly reviewed in Section 3.1. The MEB problem can also be used for finding the radius component of the radius-margin bound (Chapelle et al., 2002). Thus, as pointed out by Kumar et al. (2003), the MEB problem is useful in support vector clustering and SVM parameter tuning. However, we will show in Section 3.2 that other kernel-related problems, including the training of soft-margin one-class and two-class L2-SVMs, can also be viewed as MEB problems.

3.1 Hard-Margin SVDD

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

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

The corresponding dual is

$$\max \boldsymbol{\alpha}' \text{diag}(\mathbf{K}) - \boldsymbol{\alpha}' \mathbf{K} \boldsymbol{\alpha} \quad : \quad \mathbf{0} \leq \boldsymbol{\alpha}, \quad \boldsymbol{\alpha}' \mathbf{1} = 1, \qquad (2)$$

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

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

3.2 Viewing Kernel Methods as MEB Problems

In this paper, we consider the situation where

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

a constant[2]. This will be the case 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. Using the condition $\boldsymbol{\alpha}'\mathbf{1} = 1$ in (2), we have $\boldsymbol{\alpha}'\text{diag}(\mathbf{K}) = \kappa$. Dropping this constant term from the dual objective in (2), we obtain a simpler optimization problem:

$$\max -\boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha} \;:\; \mathbf{0} \le \boldsymbol{\alpha}, \;\; \boldsymbol{\alpha}'\mathbf{1} = 1. \quad (5)$$

Conversely, when the kernel k satisfies (4), QP's of the form (5) can always be regarded as a MEB problem (1). Note that (2) and (5) yield the same set of $\boldsymbol{\alpha}$'s, Moreover, let d_1^* and d_2^* denote the optimal dual objectives in (2) and (5) respectively, then, obviously,

$$d_1^* = d_2^* + \kappa. \quad (6)$$

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 (e.g., (Lee & Mangasarian, 2001; Mangasarian & Musicant, 2001). 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 the outliers from the normal data by solving the primal problem:

$$\min_{\mathbf{w},\rho,\xi_i} \|\mathbf{w}\|^2 - 2\rho + C\sum_{i=1}^{m} \xi_i^2 \;:\; \mathbf{w}'\varphi(\mathbf{x}_i) \ge \rho - \xi_i,$$

where $\mathbf{w}'\varphi(\mathbf{x}) = \rho$ is the desired hyperplane and C is a user-defined parameter. Note that constraints $\xi_i \ge 0$ are not needed for the L2-SVM. The corresponding dual is

$$\begin{aligned} & \max -\boldsymbol{\alpha}'\left(\mathbf{K} + \frac{1}{C}\mathbf{I}\right)\boldsymbol{\alpha} \;:\; \mathbf{0} \le \boldsymbol{\alpha}, \;\; \boldsymbol{\alpha}'\mathbf{1} = 1 \\ = \;& \max -\boldsymbol{\alpha}'\tilde{\mathbf{K}}\boldsymbol{\alpha} \;:\; \mathbf{0} \le \boldsymbol{\alpha}, \;\; \boldsymbol{\alpha}'\mathbf{1} = 1, \end{aligned} \quad (7)$$

where $\mathbf{I}$ is the $m\times m$ identity matrix and $\tilde{\mathbf{K}} = [\tilde{k}(\mathbf{z}_i, \mathbf{z}_j)] = [k(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{C}]$. It is thus of the form in (5). 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 SVM thus corresponds to the MEB problem (1), in which φ is replaced by the nonlinear map $\tilde{\varphi}$ satisfying $\tilde{\varphi}(\mathbf{z}_i)'\tilde{\varphi}(\mathbf{z}_j) = \tilde{k}(\mathbf{z}_i, \mathbf{z}_j)$. From the Karush-Kuhn-Tucker (KKT) conditions, we can recover $\mathbf{w} = \sum_{i=1}^m \alpha_i\varphi(\mathbf{x}_i)$ and $\xi_i = \frac{\alpha_i}{C}$, and $\rho = \mathbf{w}'\varphi(\mathbf{x}_i) + \frac{\alpha_i}{C}$ from any support vector $\mathbf{x}_i$.

3.2.2 Two-Class L2-SVM

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

$$\begin{aligned} \min_{\mathbf{w},b,\rho,\xi_i} \quad & \|\mathbf{w}\|^2 + b^2 - 2\rho + C\sum_{i=1}^{m} \xi_i^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}'\varphi(\mathbf{x}_i) + b) \ge \rho - \xi_i. \end{aligned} \quad (8)$$

The corresponding dual is

$$\begin{aligned} & \max_{\mathbf{0}\le\boldsymbol{\alpha}} -\boldsymbol{\alpha}'\left(\mathbf{K}\odot\mathbf{y}\mathbf{y}' + \mathbf{y}\mathbf{y}' + \frac{1}{C}\mathbf{I}\right)\boldsymbol{\alpha} \;:\; \boldsymbol{\alpha}'\mathbf{1} = 1 \\ = \;& \max -\boldsymbol{\alpha}'\tilde{\mathbf{K}}\boldsymbol{\alpha} \;:\; \mathbf{0} \le \boldsymbol{\alpha}, \;\; \boldsymbol{\alpha}'\mathbf{1} = 1, \end{aligned} \quad (9)$$

where $\odot$ denotes the Hadamard product, $\mathbf{y} = [y_1, \ldots, y_m]'$ and $\tilde{\mathbf{K}} = [\tilde{k}(\mathbf{z}_i, \mathbf{z}_j)]$ with

$$\tilde{k}(\mathbf{z}_i, \mathbf{z}_j) = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + y_i y_j + \frac{\delta_{ij}}{C}, \quad (10)$$

involving both input and label information. Again, this is of the form in (5), with $\tilde{k}(\mathbf{z}, \mathbf{z}) = \kappa + 1 + \frac{1}{C} \equiv \tilde{\kappa}$, a constant. Again, we can recover

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \varphi(\mathbf{x}_i), \quad b = \sum_{i=1}^{m} \alpha_i y_i, \quad \xi_i = \frac{\alpha_i}{C}, \quad (11)$$

from the optimal $\boldsymbol{\alpha}$ and $\rho = y_i(\mathbf{w}'\varphi(\mathbf{x}_i) + b) + \frac{\alpha_i}{C}$ from any support vector $\mathbf{z}_i$. Note that 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 is also described in (Schölkopf et al., 2001).

[2] In this case, it can be shown that the hard (soft) margin SVDD yields identical solution as the hard (soft) margin one-class SVM (Schölkopf et al., 2001). Moreover, the weight $\mathbf{w}$ in the one-class SVM solution is equal to the center $\mathbf{c}$ in the SVDD solution.

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[3] described in the proof of Theorem 2.2 in (Bădoiu & Clarkson, 2002). 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 $\epsilon > 0$, the CVM then works as follows:

1. Initialize $\mathcal{S}_0$, $\mathbf{c}_0$ and R_0.
2. Terminate if there is no $\tilde{\varphi}(\mathbf{z})$ (where $\mathbf{z}$ is a training point) falling outside the $(1+\epsilon)$-ball $B(\mathbf{c}_t, (1+\epsilon)R_t)$.
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 also provably 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, 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 $R_0 = \frac{1}{2}\|\tilde{\varphi}(\mathbf{z}_a) - \tilde{\varphi}(\mathbf{z}_b)\| = \frac{1}{2}\sqrt{\|\tilde{\varphi}(\mathbf{z}_a)\|^2 + \|\tilde{\varphi}(\mathbf{z}_b)\|^2 - 2\tilde{\varphi}(\mathbf{z}_a)'\tilde{\varphi}(\mathbf{z}_b)} = \frac{1}{2}\sqrt{2\tilde{\kappa} - 2\tilde{k}(\mathbf{z}_a, \mathbf{z}_b)}$.

In a classification problem, one may further require $\mathbf{z}_a$ and $\mathbf{z}_b$ to come from different classes. On using (10), 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 SimpleSVM (Vishwanathan et al., 2003).

[3] A similar algorithm is also described in (Kumar et al., 2003).

4.1.2 Distance Computations

Steps 2 and 3 involve computing $\|\mathbf{c}_t - \tilde{\varphi}(\mathbf{z}_\ell)\|$ for $\mathbf{z}_\ell \in \mathcal{S}$. Now,

$$\|\mathbf{c}_t - \tilde{\varphi}(\mathbf{z}_\ell)\|^2 = \sum_{\mathbf{z}_i, \mathbf{z}_j \in \mathcal{S}_t} \alpha_i \alpha_j \tilde{k}(\mathbf{z}_i, \mathbf{z}_j) - 2\sum_{\mathbf{z}_i \in \mathcal{S}_t} \alpha_i \tilde{k}(\mathbf{z}_i, \mathbf{z}_\ell) + \tilde{k}(\mathbf{z}_\ell, \mathbf{z}_\ell), \quad (12)$$

on using (3). 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 (12) 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 & 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 & 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 initialization.

4.1.3 Adding the Furthest Point

Points outside 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., 1997), 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 classification case[4] (Section 3.2.2), we have

$$\begin{aligned} &\arg \max_{\mathbf{z}_\ell \notin B(\mathbf{c}_t, (1+\epsilon)R_t)} \|\mathbf{c}_t - \tilde{\varphi}(\mathbf{z}_\ell)\|^2 \\ &= \arg \min_{\mathbf{z}_\ell \notin B(\mathbf{c}_t, (1+\epsilon)R_t)} \sum_{\mathbf{z}_i \in \mathcal{S}_t} \alpha_i y_i y_\ell (k(\mathbf{x}_i, \mathbf{x}_\ell) + 1) \\ &= \arg \min_{\mathbf{z}_\ell \notin B(\mathbf{c}_t, (1+\epsilon)R_t)} y_\ell(\mathbf{w}'\varphi(\mathbf{x}_\ell) + b), \end{aligned} \quad (13)$$

on using (10), (11) and (12). Hence, (13) chooses the *worst* violating pattern corresponding to the constraint (8). Also, as the dual objective in (9) has gradient $-2\tilde{\mathbf{K}}\boldsymbol{\alpha}$, so for a pattern ℓ currently outside the ball

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

[4] The case for one-class classification (Section 3.2.1) is similar.

on using (10), (11) and $\alpha_\ell = 0$. Thus, the pattern chosen in (13) also makes the most progress towards maximizing the dual objective. This subset selection heuristic has been commonly used by various decomposition algorithms (e.g., (Chang & 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 5), 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 & Poggio, 2001; Vishwanathan et al., 2003) can also be used. The cost then becomes quadratic rather than cubic. In the current implementation (Section 5), we use SMO. 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 $\epsilon = 0$. The proof in (Bădoiu & Clarkson, 2002) does not apply as it requires $\epsilon > 0$. Nevertheless, as the number of core vectors increases by one at each iteration and the training set size is finite, so CVM must terminate in a finite number (say, τ) of iterations, With $\epsilon = 0$, MEB($\mathcal{S}_\tau$) is an enclosing ball for all the 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, MEB($\mathcal{S}_\tau$) must also be the exact MEB of the whole ($\tilde{\varphi}$-transformed) training set. In other words, when $\epsilon = 0$, CVM outputs the *exact* solution of the kernel problem.

Now, consider $\epsilon > 0$. Assume that the algorithm terminates at the τth iteration, then

$$R_\tau \leq r_{\text{MEB}(\mathcal{S})} \leq (1+\epsilon) R_\tau \tag{14}$$

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 (7) (or (9)), which in turn is related to the optimal dual objective $d_1^* = r^2_{\text{MEB}(\mathcal{S})}$ in (2) by (6). Together with (14), we can then bound p^* as

$$R_\tau^2 \leq p^* + \tilde{\kappa} \leq (1+\epsilon)^2 R_\tau^2. \tag{15}$$

Hence, $\max\left(\frac{R_\tau^2}{p^*+\tilde{\kappa}}, \frac{p^*+\tilde{\kappa}}{R_\tau^2}\right) \leq (1+\epsilon)^2$ and thus CVM is an $(1+\epsilon)^2$-approximation algorithm. This also holds with high probability 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 and SVM^{light} (Joachims, 1999), 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. Moreover, it can also be shown that when the CVM terminates, all the points satisfy loose KKT conditions as in SMO and SVM^{light}.

4.3 Time and Space Complexities

Existing decomposition algorithms cannot guarantee the number of iterations and consequently the overall time complexity (Chang & 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 MEB sub-problem in Section 4.1.4. Moreover, we assume that each kernel evaluation takes constant time.

As proved in (Bădoiu & Clarkson, 2002), CVM converges in at most $2/\epsilon$ iterations. In other words, the total number of iterations, and consequently the size of the final core-set, are of $\tau = O(1/\epsilon)$. In practice, it has often been observed that the size of the core-set is much smaller than this worst-case theoretical upper bound (Kumar et al., 2003). This will also be corroborated by our experiments in Section 5.

Consider first the case where probabilistic speedup is not used in Section 4.1.2. 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 $O(tm + t^3)$ time, and the overall time for $\tau = O(1/\epsilon)$ iterations is

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

which is *linear* in m for a fixed ϵ.

As for space[5], since only the core vectors are involved in the QP, the space complexity for the tth iteration is $O(|\mathcal{S}_t|^2)$. As $\tau = O(1/\epsilon)$, the space complexity for the whole procedure is $O(1/\epsilon^2)$, 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 remains the same. Hence, tth iteration takes $O(t^3)$ time and the whole procedure takes

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

[5] As the patterns may be stored out of core, we ignore the $O(m)$ space required for storing the m patterns.

For a fixed ϵ, it is thus *constant*, independent of m. The space complexity, which depends only on the number of iterations τ, is still $O(1/\epsilon^2)$.

If more efficient QP solvers were used in the MEB subproblem of Section 4.1.4, both the time and space complexities can be further improved. For example, with SMO, the space complexity for the tth iteration is reduced to $O(|\mathcal{S}_t|)$ and that for the whole procedure driven down to $O(1/\epsilon)$.

Note that 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. Morever, be cautioned that the O-notation is used for studying the asymptotic efficiency of algorithms. As we are interested on 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 5.

5 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 both toy and real-world data sets. For comparison, we also run the following SVM implementations[6]:

1. L2-SVM: LIBSVM implementation (in C++);
2. L2-SVM: LSVM implementation (in MATLAB), with low-rank approximation (Fine & Scheinberg, 2001) of the kernel matrix added;
3. L2-SVM: RSVM (Lee & 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. All experiments are performed on a 3.2GHz Pentium–4 machine with 512M RAM, running Windows XP. 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 $\beta = \frac{1}{m^2}\sum_{i,j=1}^{m} \|\mathbf{x}_i - \mathbf{x}_j\|^2$.

Our CVM implementation is adapted from LIBSVM, and uses SMO for each QP sub-problem in Section 4.1.4. As in LIBSVM, our CVM also uses caching (with the same cache size as in the other LIBSVM implementations above) and stores all training patterns in main memory. For simplicity, shrinking is not used in our current CVM implementation. Moreover, we employ probabilistic speedup (Section 4.1.2) and set $\epsilon = 10^{-6}$ in all the experiments. As in other decomposition methods, the use of a very stringent stopping criterion is not necessary in practice. Preliminary studies show that $\epsilon = 10^{-6}$ is acceptable for most tasks. Using an even smaller ϵ does not show improved generalization performance, but may increase the training time unnecessarily.

5.1 Checkerboard Data

We first experiment on the 4×4 checkerboard data used by Lee and Mangasarian (2001) for evaluating large-scale SVM implementations. We use training sets with a maximum of 1 million points and 2000 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. 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 $\bar{m} \times m$ 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.

As can be seen from Figure 2, CVM is as accurate as the others. Besides, it is much faster[7] and produces far fewer support vectors (which implies faster testing) on large data sets. In particular, one million patterns can be processed in under 13s. 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 2(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 are constant w.r.t. the training set size, when it is large enough.

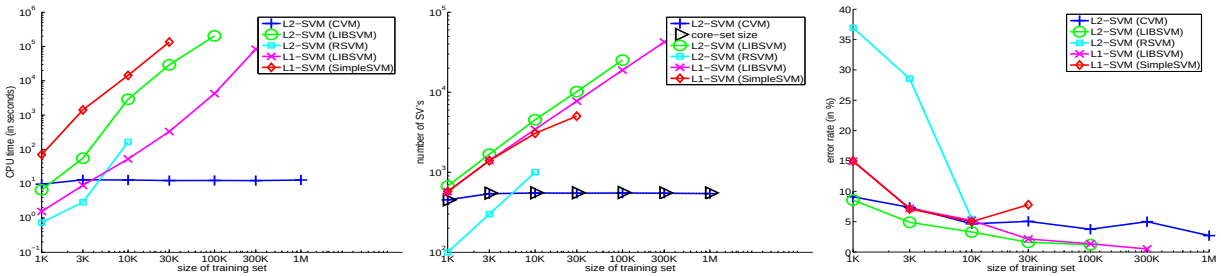
5.2 Forest Cover Type Data[8]

This data set has been used for large scale SVM training by Collobert et al. (2002). Following (Collobert et al.,

[6] 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.

[7] 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.

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



(a) CPU time. (b) number of SV's. (c) testing error.

Figure 2: Results on the checkerboard data set (Except for the CVM, all 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.

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 set $\beta = 10000$ for the Gaussian kernel. 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.

Figure 3 shows that 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[9]. 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 has been demonstrated in (Lee & Mangasarian, 2001).

5.3 Relatively Small Data Sets: UCI Adult Data[10]

Following (Platt, 1999), we use training sets with up to 32,562 patterns. As can be seen in Figure 4, 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 5.2, 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 $2/\epsilon$, independent of m (Section 4.3). For the value of $\epsilon = 10^{-6}$ used here, $2/\epsilon = 2 \times 10^6$. Although we have seen that the actual size of the core-set is often much smaller than this worst case value, however, when $m \ll 2/\epsilon$, the number of core vectors can still be dependent on m. Moreover, as has been observed in Section 5.1, the CVM is slower than the more sophisticated LIBSVM on processing these smaller data sets.

[9] 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.

[10] http://research.microsoft.com/users/jplatt/smo.html

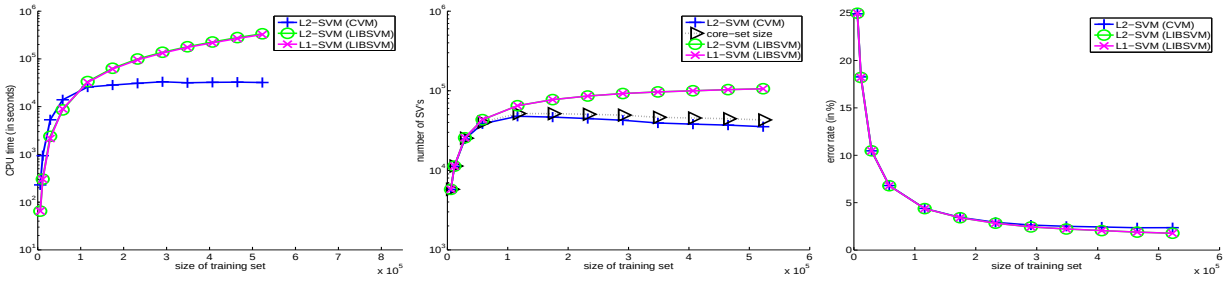
6 Conclusion

In this paper, we exploit the "approximateness" in SVM implementations. We formulate kernel methods as equivalent MEB problems, and then obtain provably 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. When probabilistic speedup is used, it even has *constant* asymptotic time and space complexities for a fixed ϵ, independent of the training set size m. Experimentally, on large data sets, it is much faster and produce far fewer support vectors (and thus faster testing) than existing methods. On the other hand, on relatively small data sets where $m \ll 2/\epsilon$, SMO can be faster. CVM can also be used for other kernel methods such as support vector regression, and details will be reported elsewhere.

References

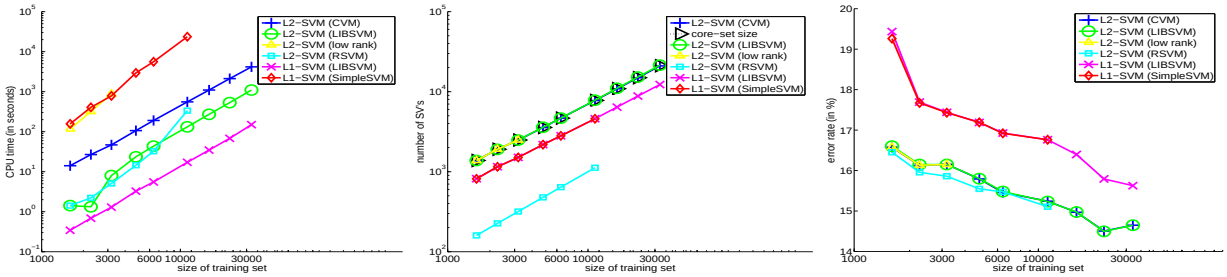
Bădoiu, M., & Clarkson, K. (2002). Optimal core-sets for balls. *DIMACS Workshop on Computational Geometry*.

Cauwenberghs, G., & Poggio, T. (2001). Incremental and decremental support vector machine learning. *Advances in Neural Information Processing Systems 13*. Cambridge, MA: MIT Press.



(a) CPU time. (b) number of SV's. (c) testing error.

Figure 3: Results on the forest cover type data set. Note that the y-axes in Figures 3(a) and 3(b) are in log scale.



(a) CPU time. (b) number of SV's. (c) testing error.

Figure 4: Results on the UCI adult data set (The other implementations have to terminate early because of not enough memory and/or training time is too long). Note that the CPU time, number of SV's and size of training set are in log scale.

Chang, C.-C., & Lin, C.-J. (2004). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

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

Fine, S., & Scheinberg, K. (2001). Efficient SVM training using low-rank kernel representation. *Journal of Machine Learning Research*, *2*, 243–264.

Garey, M., & Johnson, D. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman.

Joachims, T. (1999). Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges and A. Smola (Eds.), *Advances in kernel methods – Support vector learning*, 169–184. Cambridge, MA: MIT Press.

Kumar, P., Mitchell, J., & Yildirim, A. (2003). Approximate minimum enclosing balls in high dimensions using core-sets. *ACM Journal of Experimental Algorithmics*, *8*.

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

Lin, K.-M., & Lin, C.-J. (2003). A study on reduced support vector machines. *IEEE Transactions on Neural Networks*, *14*, 1449–1459.

Mangasarian, O., & Musicant, D. (2001). Lagrangian support vector machines. *Journal of Machine Learning Research*, *1*, 161–177.

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

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges and A. Smola (Eds.), *Advances in kernel methods – support vector learning*, 185–208. Cambridge, MA: MIT Press.

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

Smola, A., & Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 911–918). Standord, CA, USA.

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

Tax, D., & Duin, R. (1999). Support vector domain description. *Pattern Recognition Letters*, *20*, 1191–1199.

Vishwanathan, S., Smola, A., & Murty, M. (2003). SimpleSVM. *Proceedings of the Twentieth International Conference on Machine Learning* (pp. 760–767). Washington, D.C., USA.

Williams, C., & Seeger, M. (2001). Using the Nyström method to speed up kernel machines. *Advances in Neural Information Processing Systems 13*. Cambridge, MA: MIT Press.