Selective sampling-based training schemes for core vector machines

S. Asharaf

Indian Institute of Information Technology and Management – Kerala,
Technopark Campus, Trivandrum,
Kerala 695581, India
Fax: +91-471-2527568
E-mail: asharaf.s@iiitmk.ac.in
E-mail: asharaf@gmail.com

Abstract: Support vector machines (SVMs) are hyperplane classifiers defined in a kernel induced feature space. The high computational and space requirements for solving the conventional SVM problem prohibit its use in applications involving large datasets. Core vector machine (CVM) is a suitable technique for scaling an SVM for large-scale pattern classification problems. But in applications where the datasets are unbalanced, the performance of CVM is observed to be poor both in terms of generalisation and training time. In such scenarios, the CVM performance highly depends on the orderings of data points belonging to the two classes within the dataset. In this paper, we propose two training schemes which improve the performance of CVM irrespective of the orderings of patterns belonging to different classes within the dataset. These methods employ a selective sampling-based training of CVM using novel kernel-based clustering algorithms. Empirical studies made on several synthetic and real world datasets show that the proposed strategies improve the performance of CVM on large datasets.

Keywords: support vector machine; SVM; CF tree; clustering; kernel function; selective sampling.

Reference to this paper should be made as follows: Asharaf, S. (2013) 'Selective sampling-based training schemes for core vector machines', *Int. J. Granular Computing, Rough Sets and Intelligent Systems*, Vol. 3, No. 1, pp.20–43.

Biographical notes: S. Asharaf is an Assistant Professor at Indian Institute of Information Technology and Management – Kerala. He received his Master of Engineering and PhD degrees from Indian Institute of Science, Bangalore. He is one of the recipients of IBM best PhD student award, 2006. He has over eight years of teaching/research and industry experience in information technology. He has several research publications in reputed international journals and conferences. His areas of interest include machine learning, data analytics, soft computing, web mining and web business models.

1 Introduction

Kernel-based methods are considered as effective techniques by researchers in machine learning and exploratory data analysis. In these methods, kernel functions are used to compute the inner product of data vectors in an implicitly defined kernel induced feature space. By choosing a suitable kernel function any machine learning algorithm that requires only the inner product between data vectors can be transformed into a kernel-based method and this technique is known as kernel trick. One of the most celebrated kernel-based methods is support vector machine (SVM) for classification. SVMs are hyperplane classifiers defined in a kernel induced feature space. They achieve optimal separation of patterns by margin maximisation. Even though several novel classification algorithms (James and Dimitrijev, 2012; Gou et al., 2012) have been proposed in the recent past, SVM-based methods still enjoy popularity among practitioners and researchers due to the firm theoretical foundation in statistical learning theory (Vapnik, 1998), good generalisation performance in many domains, the geometric interpretability and ease of application in different fields.

The classification problems involved in data mining applications typically deal with massive collection of data and hence the main issue in using SVM here is that of scalability. The SVM problem is usually formulated as a quadratic programming (QP) problem. The existing solution strategies for this problem have an associated time and space complexity that is (at least) quadratic in the number of data points. This makes the SVM very expensive to use even on datasets having a few thousands of elements.

Several attempts are made to reduce the time and space complexities of SVM to make the training feasible for large datasets. Some of the major directions explored in the literature are

a chunking (Vapnik, 1998; Platt, 1999)

b incremental SVM formulations (Diehl and Cauwenberghs, 2003; Tax and Laskov, 2003; Fung and Mangasarian, 2002; Cauwenberghs and Poggio, 2001)

c low rank approximations of kernel matrix (Smola and Scholkopf, 2000; Williams and Seeger, 2001)

d replacing the QP problem with a simpler optimisation problem (Mangasarian and Musicant, 2001a, 2001b; Suykens and Vandewalle, 1999; Fung and Mangasarian, 2003; Mangasarian and Thompson, 2006)

e reducing the training set size used with the SVM (Lee and Mangasarian, 2001; Cao and Boley, 2003; Wang et al., 2005; Yu et al., 2003; Pavlov et al., 2000)

f other scaling up methods like Kernel-Adatron (Friess et al., 1998) and simple SVM (Viswanathan et al., 2003).

Core vector machine (CVM) (Tsang et al., 2005) is a suitable technique for scaling up SVM to handle large datasets. In CVM, the quadratic optimisation problem involved in SVM is formulated as an equivalent minimum enclosing ball (MEB) problem. The MEB problem is then solved by using a faster approximation algorithm introduced by Badou and Clarkson (2002). The CVM is observed to have given good performance when the training dataset has roughly equal number of points from the two classes involved (balanced dataset). When the training dataset is unbalanced, the CVM gives good

performance only when the data points are obtained by uniformly sampling from a set of well behaved distributions (such as normal distributions) (Loosli and Canu, 2007). But in unbalanced datasets where the data points come from some arbitrary distributions and are not well distributed within the dataset, the performance of CVM is observed to be highly dependent on the orderings of points belonging to the two classes within the dataset. It is observed that when the data points belonging to the two classes in an unbalanced dataset are not uniformly distributed within the dataset, CVM gives poor performance both in terms of generalisation and training time. Existence of unbalanced training datasets is not rare in real world applications. In this case, a scenario involving most of the data points coming from well behaved distributions and having them well distributed within the dataset is far from the reality. Hence, CVM has to be used with some preprocessing for the training dataset to overcome such difficulties involved.

In this paper, we propose two selective sampling-based training schemes for CVM namely incremental clustering-based CVM (ICBCVM) and hierarchical clustering-based CVM (HCBCVM) for improving the performance of a two class non-linear CVM. These methods can handle large balanced/unbalanced datasets irrespective of the orderings of points from the two classes involved within the dataset.

The ICBCVM method employs a kernel-based incremental clustering (KBIC) algorithm to generate cluster abstractions of the training data in an arbitrary kernel induced feature space. Here, initially the CVM is trained using the cluster abstractions obtained from KBIC. Then only those clusters that can contribute to a refinement of the training process are expanded using a date base scan. The final training of CVM is then done using the training set obtained by this expansion process.

The HCBCVM method uses a novel kernel-based hierarchical clustering (KBHC) (Asharaf et al., 2006) algorithm to generate hierarchical cluster abstractions of the training data in Gaussian kernel induced feature space. Here the training process starts with a high level abstraction of the training data generated in the Gaussian kernel induced feature space by the KBHC algorithm. It is continued by selective declustering (expansion) of only those clusters that are needed during the subsequent iterations of the training process using the cluster hierarchy generated by KBHC. This process continues till there are no more clusters whose expansion will bring some improvement in the training process.

The rest of the paper is organised as follows. In Section 2, a brief discussion on CVMs is given. Section 3 discusses the KBHC algorithms namely KBIC and KBHC. In Section 4, the proposed ICBCVM and HCBCVM methods are introduced. Results are given in Section 5 and Section 6 deals with conclusions.

2 Core vector machines

CVM applies kernel methods to data intensive applications involving large datasets. In CVM, the quadratic optimisation problem involved in SVM is formulated as an equivalent MEB problem. It is then solved using a fast approximate MEB finding algorithm employing the concept of core sets (Tsang et al., 2005).

Given a set of data points $S = \{x_i\}_{i=1}^{N}$ where $x_i \in R^d$ (here d is the dimenisionality of the input space), the MEB of S [denoted as $MEB(S)$] is the smallest ball that contains all the points in S. Let k be a kernel function with the associated feature map ϕ defining a

high dimensional kernel induced feature space. Now the primal problem for the MEB in the kernel induced feature space to find the MEB $B(a, R)$ with centre a and radius R can be stated as

$$\begin{aligned} &\min \quad R^2 \\ &\text{s.t.} \quad \left\|\left(x_i\right)-a\right\|^2 \le R^2 \quad \forall i \end{aligned} \tag{1}$$

The corresponding Wolfe dual form (Fletcher, 2000) is

$$\begin{aligned} &\min \quad \sum_{i,j=1}^{N} \alpha_i \alpha_j k\left(x_i, x_j\right) - \sum_{i=1}^{N} \alpha_i k\left(x_i, x_i\right) \\ &\text{s.t.} \quad \sum_{i=1}^{N} \alpha_i = 1 \quad \alpha_i > 0 \quad \forall i \end{aligned} \tag{2}$$

Now consider a situation where

$k(x, x) = \kappa$, a constant.

This is true for kernels like Gaussian given by

$$k\left(x_i, x_j\right) = e^{-q\left\|x_i - x_j\right\|^2}$$

where $\|\cdot\|$ represents the L_2 norm and q is a user given parameter.

The dot product kernel like polynomial kernel given by $k(x_i, x_j) = (< x_i \cdot x_j > + 1)^{\lambda}$ with normalised inputs x_i and x_j also satisfy the above said condition. Here λ is a non-negative integer and $< \cdot >$ represents the dot product.

The Wolfe dual form of the MEB problem can now be written as

$$\begin{aligned} &\min \quad \sum_{i,j=1}^{N} \alpha_i \alpha_j k\left(x_i, x_j\right) \\ &\text{s.t.} \quad \sum_{i=1}^{N} \alpha_i = 1 \quad \alpha_i \ge 0 \quad \forall i \end{aligned} \tag{3}$$

When the kernel function satisfies $k(x, x) = \kappa$, any QP of the above form can be regarded as an MEB problem.

2.1 A two class SVM problem as an MEB problem

Given a training dataset $S = \{(x_i, y_i)\}_{i=1}^{N}$ where $x_i \in R^d$ and $y_i \in \{+1, -1\}$, the primal for the two class SVM problem can be stated as

$$\begin{aligned} &\min \quad \|w\|^2 + b^2 - 2\rho + C\sum_{i,j=1}^{N} \xi_i^2 \\ &\text{s.t.} \quad y_i\left(w'\phi\left(x_i\right) + b\right) \ge \rho - \xi_i \quad \forall i \end{aligned} \tag{4}$$

The Wolfe dual form is

$$\begin{aligned} \min \quad & \sum_{i,j=1}^{N} \alpha_i \alpha_j \left(y_i y_j k\left(x_i, x_j\right) + y_i y_j + \frac{\delta_{ij}}{C} \right) \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i = 1 \quad \alpha_i \geq 0 \quad \forall i \end{aligned} \tag{5}$$

Here δ_{ij} is the Kronecker delta function.

The above equation can be rewritten as

$$\begin{aligned} \min \quad & \sum_{i,j=1}^{N} \alpha_i \alpha_j \tilde{k}\left(x_i, x_j\right) \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i = 1 \quad \alpha_i \geq 0 \quad \forall i \end{aligned} \tag{6}$$

where

$$\tilde{k}\left(x_i, x_j\right) = y_i y_j k\left(x_i, x_j\right) + y_i y_j + \frac{\delta_{ij}}{C}.$$

When $k(x, x) = \kappa$ is satisfied this transformed kernel function $\tilde{k}$ satisfies the condition

$$\tilde{k}(x, x) = \kappa_1, \text{ some constant.}$$

Hence, the above mentioned problem is an MEB problem. We now describe the algorithm to find approximate MEB.

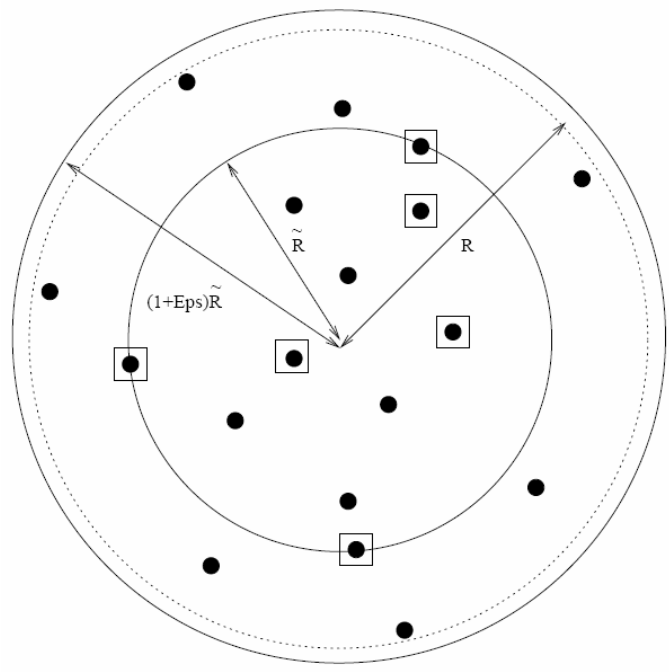
2.2 Approximate MEB finding algorithm

The traditional algorithms for finding exact MEBs are not efficient for $d > 30$ (Tsang et al., 2005) and hence the CVM method adopts a faster approximation algorithm introduced by Badou and Clarkson (2002). It returns a solution within a multiplicative factor of $(1 + \epsilon)$ to the optimal value, where ϵ is a small positive number.

The $(1 + \epsilon)$ approximation of the MEB problem is obtained by solving the problem on a subset of the dataset called *core set*. Let $B_S(a, R)$ be the exact MEB with centre a and radius R for the dataset S and $B_Q(\tilde{a}, \tilde{R})$ be the MEB with centre $\tilde{a}$ and radius $\tilde{R}$ found by solving the MEB problem on a subset of S called core set (Q). Given an $\epsilon > 0$, a ball $B_Q(\tilde{a}, (1+\epsilon)\tilde{R})$ is an $(1 + \epsilon)$-approximation of $MEB(S) = B_S(a, R)$ if $S \subset B_Q(\tilde{a}, (1+\epsilon)\tilde{R})$ and $\tilde{R} \leq R$.

Formally, a subset $Q \subseteq S$ is a core set of S if an expansion by a factor $(1 + \epsilon)$ of its MEB contains S (i.e., $S \subset B_Q(\tilde{a}, (1+\epsilon)\tilde{R})$) as shown in Figure 1.

Figure 1 The dotted circle is the exact MEB of the entire data



Note: The inner circle is the exact MEB of core set (the set of points enclosed in squares) and its (1 + ϵ) expansion (the outer circle) covers all points.

The approximate MEB finding algorithm uses a simple iterative scheme: at the t^{th} iteration, the current estimate $B_Q(\tilde{a}_t, \tilde{R}_t)$ is expanded incrementally by including that data point in S that is farthest from the centre $\tilde{a}$ and falls outside the (1 + ϵ)-ball $B_Q(\tilde{a}_t, (1+\epsilon)\tilde{R}_t)$. The computation to find the farthest point becomes very expensive when the number of data points in S is very large. Hence, to speed up the process CVM uses a probabilistic method. Here a random sample S' having 59 points is taken from the points that fall outside the (1 + ϵ)-ball $B_Q(\tilde{a}_t, (1+\epsilon)\tilde{R}_t)$. Then the point in S' that is farthest from the centre $\tilde{a}_t$ is taken as the approximate farthest point from S. The iterative strategy to include the farthest point in the MEB is repeated until all the points in S are covered by $B_Q(\tilde{a}_t, (1+\epsilon)\tilde{R}_t)$. The set of all such points that got added forms the core set of the dataset.

The CVM method is observed to have given good performance when the training dataset is balanced in terms of the number of points belonging to the two classes involved. When the training dataset is unbalanced, the CVM gives good performance only when the data points are obtained by uniformly sampling from a set of well behaved distributions (such as normal distribution). But in unbalanced training datasets where the data points come from some arbitrary distributions and are not well distributed (not uniformly sampled from the a set of distributions) within the dataset, the performance of CVM is observed to be highly dependent on the orderings of points belonging to the two classes within the dataset. In most of these orderings, the CVM gives poor performance

both in terms of generalisation and computational expense. The bad performance of CVM in this scenario is due to the poorly sampled subset S' of the dataset S, that is used for finding the farthest point from the current centre $\tilde{a}_t$ of the MEB found at any iteration t. Usually in real world applications we may have to deal with unbalanced training datasets belonging to some arbitrary distributions. Here a scenario where the points from the two classes are well distributed within the dataset is far from reality. Hence, CVM has to be used with some preprocessing for the training dataset to reduce the difficulties arising due to the data ordering dependent behaviour of CVM in unbalanced datasets. The proposed ICBCVM and HCBCVM schemes employ such an approach.

3 Kernel-based clustering methods

The novel kernel-based clustering schemes namely KBIC used in ICBCVM and KBHC used in HCBCVM are discussed in this section.

3.1 Kernel-based incremental clustering

The KBIC algorithm is a kernalised form of leader (Spath, 1980) algorithm. It can construct cluster abstractions in any arbitrary kernel induced feature space using a single dataset scan. KBIC starts with a singleton cluster containing any arbitrarily chosen initial data point. For each cluster, the first data point that gets assigned to that cluster is taken as its representative. At any point, the algorithm assigns the current data point to its most similar cluster or the data point itself may get added as a singleton cluster. The new singleton cluster is added when the current data point does not get qualified to be added to any of the currently available clusters based on a user given similarity (dissimilarity) threshold (*Thr*). The dissimilarity function used in KBIC is the squared Euclidean distance between the images of a pair of points x_i and x_j in the kernel induced feature space. It is given by

$$\begin{aligned} D(x_i, x_j) &= \left\| \phi(x_i) - \phi(x_j) \right\|^2 \\ &= k(x_i, x_i) - 2k(x_i, x_j) + k(x_j, x_j) \end{aligned} \tag{7}$$

This is computed using a Mercer kernel function $k(x_i, x_j)$ giving the dot product $\phi(x_i) \cdot \phi(x_j)$ in the kernel induced feature space.

The KBIC algorithm uses a set data structure namely *CP* to maintain the cluster prototypes obtained. To describe the algorithm, let us define N and n_c as the number of points in the dataset and the current number of clusters respectively. Now the algorithm can be given as

---

KBIC (Training Data, Parameters)

1 Initialize: $n_c = 1$, $CP = \{x_1\}$

2 For each data point x_i, $i = 2...N$ do

- Find the Winning Cluster j for x_i from the available clusters $C_k \in CP$, $k = 1..n_c$ as

$$j = \min_k \quad D(C_k, x_i)$$

- Assign x_i to the winning cluster j if
  $D(C_j, \text{xi}_\text{i}) <= Thr$
- Else add a singleton cluster as
  $n_c = n_c + 1$, $CP = CP \cup \{x_i\}$

3.2 Kernel-based hierarchical clustering

KBHC algorithm is a kernalised form of BIRCH (Zhang et al., 1996). Using a single dataset scan this algorithm can generate a hierarchical cluster indexing structure similar to the CF tree in a Gaussian kernel induced feature space. The concept of modified clustering feature (MCF) and MCF tree are at the core of KBHC. Here each MCF is a triple that summarises the information that we maintain about a cluster in the kernel induced feature space.

Given the data points $\{x_i\}_{i=1}^N$ we find μ, the data space representation of the best prototype in the least squares error sense for a cluster C defined in the kernel induced feature space, by solving the following problem:

$$\min_{\mu} J(\mu) = \min_{\mu} \sum_{x_i \in C} \left\| \phi(x_i) - \phi(\mu) \right\|^2 \tag{8}$$

where ϕ is the non-linear transformation implicitly achieved by the kernel function. The objective function in equation (8) can now be expanded using Mercer kernel $k(x, y) = \phi(x) \cdot \phi(y)$ giving the dot product in the kernel induced feature space.

For Gaussian kernel we have $k(x, x) = 1$ and hence can write the objective function as

$$\min_{\mu} J(\mu) = \max_{\mu} \sum_{x_i \in C} k(x_i, \mu) \tag{9}$$

The solution of this problem results in the fixed point equation

$$\mu = \frac{\sum_{x_i \in C} k(x_i, \mu) x_i}{\sum_{x_i \in C} k(x_i, \mu)} \tag{10}$$

This minimisation is performed by starting with an initial guess for μ and then iteratively recomputing the μ until some stopping criterion is met. The stopping criterion used can be based on a user given tolerance (*Tol*) defining a threshold for the change in μ values in consecutive iterations.

The MCF vector representing a cluster C_i in the Gaussian kernel induced feature space can now be defined as a triple:

$$MCF_i = (N_i, LS_i, \mu_i)$$

where N_i is the number of data points in the cluster C_i and LS_i is the linear sum which is detailed in the next section. For each cluster found, KBHC maintains only its MCF vector from which the needed statistics used in computations involving that cluster can be obtained.

3.2.1 Merging clusters

KBHC constructs the hierarchical cluster abstraction of a dataset by arranging the MCF vectors in the form of an MCF tree. In this cluster hierarchy the MCF vectors corresponding to the lower level clusters are merged to form the higher level abstractions. The merging process involved here can be explained as follows. Let

$$MCF_j = \left(N_j, LS_j, \mu_j\right), \quad j = 1 \ldots m$$

be the MCF vectors of m disjoint clusters, $\{C_j\}_{j=1}^{m}$, which are to be merged. The KBHC algorithm finds the MCF vector

$$MCF_{new} = \left(N_{new}, LS_{new}, \mu_{new}\right)$$

for the merged cluster C_{new} by minimising the objective function given in equation (8). Here

$$N_{new} = \sum_{j=1}^{m} N_j$$

The initial value of μ_{new} used in the minimisation process is computed as

$$\mu_{new}^{(0)} = \frac{\sum_{j=1}^{m} N_j \mu_j}{\sum_{j=1}^{m} N_j}$$

At iteration t, the algorithm computes the new value of the prototype vector $\mu_{new}^{(t)}$ as

$$\mu_{new}^{(t)} = \frac{\sum_{x_k \in C_{new}} k\left(x_k, \mu_{new}^{(t-1)}\right) x_k}{\sum_{x_k \in C_{new}} k\left(x_k, \mu_{new}^{(t-1)}\right)} \tag{11}$$

The iterations continue till

$$\left\| \mu_{new}^{(t-1)} - \mu_{new}^{(t)} \right\| < Tol$$

To simplify this computation process we will now make an assumption

$$k\left(x_k, \mu_{new}^{(t)}\right) \approx k\left(\mu_j, \mu_{new}^{(t)}\right) \quad \forall x_k \in C_j \quad j = 1, 2 \ldots m \tag{12}$$

With this assumption it requires only one kernel evaluation per cluster and also reduces the space requirement significantly because it has to maintain only the MCF vector for any cluster instead of all the data points belonging to that cluster. Empirically this is found to be a good approximation. Using this assumption we can write equation (11) as

$$\mu_{new}^{(t)} \approx \frac{\sum_{j=1}^{m}\left(k\left(\mu_j, \mu_{new}^{(t-1)}\right)\sum_{x_k \in C_j} x_k\right)}{\sum_{j=1}^{m} N_j k\left(\mu_j, \mu_{new}^{(t-1)}\right)} \tag{13}$$

Note that a linear sum appears in equation (13) and this is defined as the LS_i entry of the MCF vector representing cluster C_i, where

$$LS_i = \sum_{x_k \in C_i} x_k .$$

The MCF tree is a height balanced tree with two parameters: branching factor B and threshold T. Here each node contains at most B entries of the form $\{[MCF_i, child_i]\}_{i=1}^{B}$. Here MCF_i is the MCF of a subcluster of the cluster represented by this node and $child_i$ is the pointer to this subcluster if it is a non-leaf node and *NULL* pointer if it is a leaf node. So a node represents a cluster made up of all the subclusters represented by its MCF entries. All the entries in a leaf node must satisfy a *threshold requirement*, with respect to a threshold value T: the radius r of the cluster has to be less than T. The radius r_i of a cluster C_i is given by

$$r_i = \left(\frac{\sum_{x_j \in C_i} \left\|\phi(x_j) - \phi(\mu_i)\right\|^2}{N_i}\right)^{\frac{1}{2}} \tag{14}$$

For each cluster C_i this is computed using the μ_l, $l = 1 \ldots p$ values of its p subclusters. Now using the assumption given in equation (12) we can write equation (14) as

$$r_i = \left(\frac{\sum_{l=1}^{p} N_l\left(2 - 2k\left(\mu_l, \mu_i\right)\right)}{\sum_{l=1}^{p} N_l}\right)^{\frac{1}{2}}$$

The tree size is a function of T. Larger the value of T, smaller is the tree size. Once the space requirement for a data point is known, the space required for an MCF vector can be calculated. Now the parameters B and T can be determined based on the size of the tree that can be accommodated in the available memory. In the proposed KBHC algorithm we treat B and T as user given.

The MCF tree is dynamically built as new data points are inserted. The construction of the MCF tree starts with a single leaf node having only one MCF vector representing a singleton cluster containing any arbitrarily chosen initial data point. Then tree is grown by inserting the data points one at a time as in BIRCH. Except the cluster merging process as explained above the MCF tree construction of KBHC proceeds in the same manner as the CF tree construction in BIRCH.

3.2.2 KBHC with buffering

In the KBHC method discussed above, while constructing the MCF tree every data point has to be checked for possible inclusion within an already generated cluster abstraction (MCF vector) found so far. This checking process requires multiple levels of comparison from the root until the leaf with the most similar cluster (MCF vector) of the MCF tree. If the data point does not get added to an existing cluster it will get added as a new cluster and this information may have to be communicated up until the root in the hierarchy given by the MCF tree. To speed up the MCF tree construction process here we employ a buffering scheme which is explained below.

A data buffer F of size L is used here. Initially, L data points are read into the buffer from the dataset. The MCF tree construction process starts with an MCF tree having a singleton cluster containing an arbitrarily chosen data point from the dataset. At any iteration t the algorithm selects the data point $x_t \in F$ which is farthest from the centre of the cluster represented by the root of the currently obtained MCF tree for addition. If x_t gets added to an existing cluster MCF_r of the MCF tree, the algorithm removes all those points from F which are within a distance of T (the threshold value used by KBHC) from the cluster represented by MCF_r. Once this is done, more points are read from the dataset to fill the buffer F and the process repeats till all the points in the dataset are considered.

4 Clustering-based CVMs

The ICBCVM method and HCBCVM method are introduced in this section.

4.1 Incremental clustering-based CVM

This is a two phase algorithm. In the first phase KBIC is used to generate a high level description of the data (clusters) in an appropriate kernel induced feature space. The cluster prototypes obtained are used to train a CVM and the corresponding core set and support vectors (SVs) are identified. In the second phase the training set for the subsequent training of CVM is obtained by a declustering (expansion) process that expands all those clusters that falls on or outside the MEB found by the previous CVM training process as shown in Figure 2. The declustering phase expands a cluster i with cluster prototype(leader) x_{rep} if

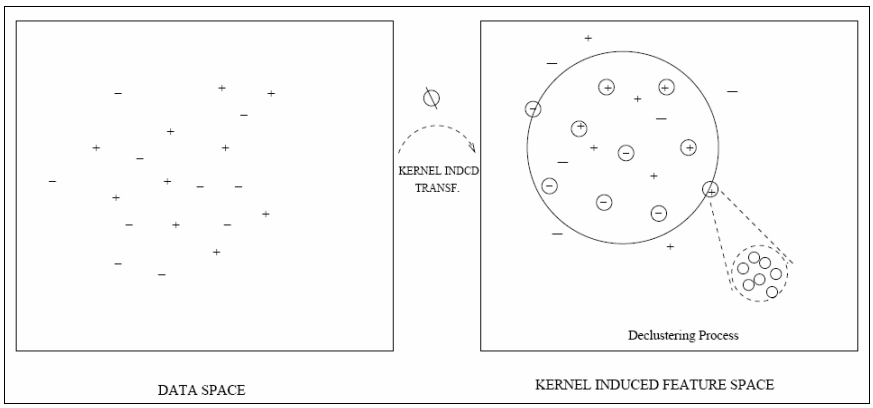
$$G\left(x_{rep}\right) \geq R \tag{15}$$

where $G(x_{rep})$ is the distance of the cluster prototype x_{rep} from the centre of the MEB and R is the radius (Tsang et al., 2005) of the MEB computed in the previous iteration.

The distance $G(x_i)$ of a point x_i from the centre a of the MEB is given by

$$\begin{aligned} G^2\left(x_i\right) &= \left\|\phi\left(x_i\right) - a\right\|^2 \\ &= k\left(x_i, x_i\right) - 2\sum_{j=1}^{n} \alpha_j k\left(x_j, x_i\right) + \sum_{j,k=1}^{n} \alpha_j \alpha_k k\left(x_j, x_k\right) \end{aligned} \tag{16}$$

Figure 2 ICBCVM: data space to kernel induced feature space transformation (implicitly done by the kernel function) and the declustering process in the kernel induced feature space



Note: Here the core points are shown inside small circles.

This is obtained using the expression

$$a = \sum_{i=1}^{n} \alpha_i \phi(x_i)$$

given by the KKT conditions on the Lagrangian for equation (1). Here n is the number of points in the core set from which the MEB was obtained in the previous iteration.

Now the radius R of the MEB computed at current iteration can be given as

$$R = G(x_i) \text{ where } 0 < \alpha_i < C \tag{17}$$

Now the algorithm can be stated as

ICBCVM(Data, parameters)

1 Generate cluster abstractions of the training data using KBIC.

2 Train SVM with the cluster prototypes.

3 Expand the clusters (declustering) near the boundary to obtain the refined training set.

4 Train SVM on the new training set obtained.

Please note that a proper selection of the user defined parameter *threshold* used by KBIC is crucial for the ICBCVM method and can be chosen based on the available RAM size.

4.2 Hierarchical clustering-based CVM

HCBCVM is aimed at scaling CVMs with Gaussian kernel function to handle very large datasets. This algorithm employs a selective sampling strategy for the training process as in CBSVM (Yu et al., 2003). Here KBHC with Buffering is used to construct two hierarchical cluster indexing structures (MCF trees) in Gaussian kernel induced feature space from the datasets corresponding to the positive and negative classes as shown in Figure 3. This CVM training process starts with the μ values computed from the MCF entries of the root node. At any iteration of the selective sampling-based training of CVM, the training set for the subsequent iterations is obtained by selectively declustering

the MCF entries corresponding to those clusters that are either on the boundary of the MEB or outside as shown in Figure 4. The declustering phase expands a cluster represented by MCF_i if

$$G(\mu_i) \geq R$$

where $G(\mu_i)$ is the distance of the i^{th} cluster from the centre of the MEB as given by equation (16) and R is the radius (Tsang et al., 2005) of the MEB computed in the previous iteration as given by equation (17). The selective sampling process and retraining of CVM continues till leaf nodes of the MCF trees are encountered.

Figure 3 HCBCVM: data space to kernel induced feature space transformation (implicitly done by the kernel function) and the MCF trees constructed for positive class and negative class in kernel induced feature space

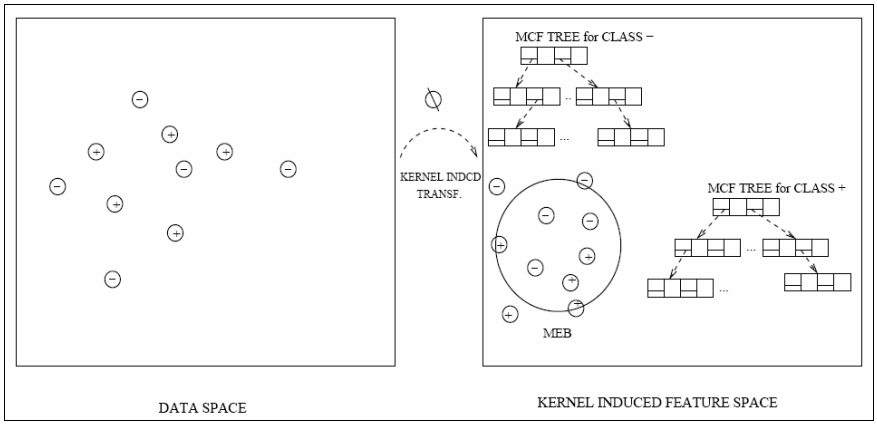
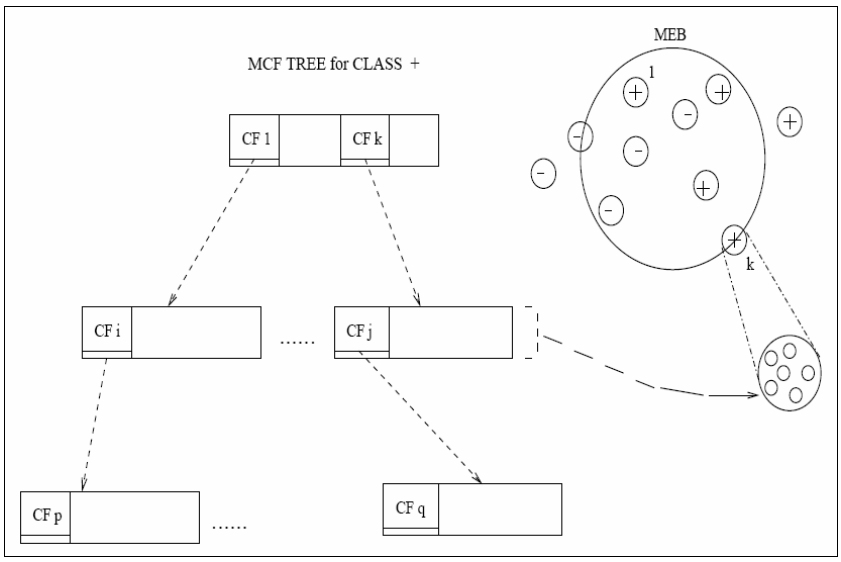


Figure 4 HCBCVM: declustering of clusters (MCF vectors) falling on or outside the MEB found in the current iteration of CVM training to get the training set for the subsequent training step



4.3 The algorithm

The HCBCVM algorithm employs two sets of user defined parameters. They are

- the parameters B, T, tolerance (Tol) and buffer_size (B_S) used in MCF tree construction
- the parameter q of the Gaussian kernel function.

Now we summarise the HCBCVM algorithm as

HCBCVM(Training data, Parameters)

1 Construct two MCF trees from positive and negative class examples respectively using KBHC with Buffering algorithm.

2 Perform CVM training using the iterative selective sampling based training strategy discussed above.

In a recent technical report by Loosli and Canu (2007), it was pointed out that the performance of CVM is highly dependent on the choice of its hyper parameters viz. the slack trade-off (C), the tolerance (ϵ) and the kernel parameter q when Gaussian (RBF) kernel function is used. In the case of RBF kernel functions the bandwidth (q) can be estimated using the distance between points from opposite classes (Tsang et al., 2005). All the CVM experiments cited in Tsang et al. (2005) have used a fixed ϵ value of 10^{-6}. About the choice of parameter C, some of their observations are

- The $\frac{\delta_{ij}}{C}$ term in the modified kernel function $\tilde{k}$ used by CVM has a non-negligible regularisation effect in the performance of CVM (Loosli and Canu, 2007).
- The stopping tolerance ϵ used in CVM is scaled (Loosli and Canu, 2007) by a factor of $\frac{1}{C}$ when compared to the ϵ value used in ν-SVM formulation.
- For small C values, the kernel $\tilde{k}$ becomes well conditioned and thus CVM gives good generalisation performance. But due to the scaling factor $\frac{1}{C}$ in the stopping tolerance, the training time is high in this case.
- Regarding training time medium C value is the best configuration, with some compromise in generalisation performance
- For large C values, CVM gives poor generalisation performance.

They have also observed that the effect of the hyper parameter C increases with the increase in training set size (Loosli and Canu, 2007). It may be noted that the proposed ICBCVM and HCBCVM methods train the CVM using a reduced training set obtained by the selective sampling strategy. Hence, the variability in performance of CVM with the choice of the hyper parameter C will be less in ICBCVM and HCBCVM compared to the CVM.

Since ICBCVM and HCBCVM work with separate cluster abstractions made for each of the two classes, the problems caused due to the unbalancing in datasets are less here.

Further, the use of cluster abstractions generated for the training data in the CVM training process reduces the effect of different ordering of data points within the dataset in the selective sampling-based CVM training.

5 Experimental results

Experiments are done with four synthetic datasets and four real world datasets. In all the experiments, we used CVM Tsang et al. (2005) implementation available at 'http://www.cs.ust.hk/~ivor/cvm.html'. We compared the results obtained with CVM trained on the full dataset, the proposed ICBCVM algorithm and the HCBCVM algorithm. In all the experiments we used Gaussian kernel function. For ICBCVM and HCBCVM, the reported training time includes the time used for generating cluster abstractions of the training data and time taken by the selective sampling-based training of CVM. All the experiments were done on a Intel Xeon(TM) 3.06 GHz machine with 2 GB RAM.

5.1 Synthetic datasets

To evaluate the performance of CVM and the proposed ICBCVM and HCBCVM methods we have generated four synthetic datasets. One of them has equal number of data points from both the classes involved. The rest three synthetic datasets generated are unbalanced in terms of the number of data points belonging to two classes involved. All the four synthetic datasets generated have 400,000 data points in the training set and 100,000 points in the test set. These are obtained from ten multivariate (five dimensional) normal distributions. In the case of balanced synthetic dataset (BSYN), five of these distributions are labelled as belonging to class +1 and the rest five as belonging to class –1. In the unbalanced synthetic datasets (USYN1, USYN2 and USYN3), the points from only one of the distributions are labelled as class +1 and all the points form the other nine distributions are labelled as class –1. To study the effect of different orderings of data points belong to the two classes within the dataset on the performance of CVM, we have made two orderings for the balanced dataset BSYN and four orderings for each of the unbalanced datasets USYN1, USYN2 and USYN3. A detailed description of the ordering strategy used will be explained later. Please note that the performance of ICBCVM and HCBCVM is independent of the orderings.

To discuss the synthetic data generation process, let us make use of the following formalisations. Let the dataset D be made up of disjoint chunks of points as

$$D = d_1 \cup d2 \cup \ldots d_k \text{ for some } k \geq 1$$

where each d_i is a chunk of points generated from one of the ten normal distributions mentioned above. Further, each of these ten normal distributions may contribute multiple disjoint chunks in the dataset. Now for training datasets, we have $| D |$ = 400,000 and for the test set $| D |$ = 100,000. Here $| \cdot |$ denote the cardinality.

The number of points in each of the chunks d_i are different in the four synthetic datasets. Now the generation of the synthetic datasets viz. BSYN, USYN1, USYN2 and USYN3 can be explained as

5.1.1 BSYN

In BSYN, each chunk d_i has only one data point. So we have

$$D = d_1 \cup d_2 \cup \ldots d_k$$

where $| d_i | = 1$.

So we have $k = 400{,}000$ for the training set and $k = 100{,}000$ for the test set.

Each of these data points (here each chunk) is generated by uniformly selecting (using a uniform random number generator between 1 and 10) one distribution from the ten normal distributions used for the data generation. A single data point is generated from this distribution and it is given the label (class +1 or class –1) of that distribution (as explained above).

5.1.2 USYN1

As in BSYN here also each chunk d_i is having only one data point. So we have

$$D = d_1 \cup d_2 \cup \ldots d_k$$

where $| d_i | = 1$.

So we have $k = 400{,}000$ for the training set and $k = 100{,}000$ for the test set.

Each of these data points (here each chunk) is generated by uniformly selecting (using a uniform random number generator between 1 and 10) one distribution from the ten normal distributions used for the data generation. A single data point is generated from this distribution and it is given the label (class +1 or class –1) of that distribution (as explained above).

5.1.3 USYN2

In this dataset we have

$$D = d_1 \cup d_2 \cup \ldots d_k$$

where $| d_i | = 40{,}000$ for the training set and $| d_i | = 10{,}000$ for the test set and hence $k = 10$.

In the case of both training set and test set, the points in each of the chunks d_i, $i = 1 \ldots 9$ are generated from one of the nine distributions labelled as belonging to class –1 and the chunk d_{10} is generated from the distribution labelled as class +1.

5.1.4 USYN3

In the case of USYN3 we have

$$D = d_1 \cup d_2 \cup \ldots d_k$$

where for the training set we have

$$d_i = \begin{cases} 40{,}000 & \text{if } i \text{ is even} \\ 4{,}000 & \text{if } i \text{ is odd} \end{cases}$$

and for the test set we have

$$d_i = \begin{cases} 10,000 & \text{if } i \text{ is even} \\ 1,000 & \text{if } i \text{ is odd} \end{cases}$$

Hence, $K = 19$.

In the case of both training set and test set, the points in each of the even numbered chunks d_i, $i = 2, 4,\ldots,18$ are generated from one of the nine distributions labelled as belonging to class –1 and the points in the odd numbered chunks d_i, $i = 1, 3,\ldots,19$ are generated from the distribution labelled as class +1.

Thus, we get four datasets which are composed of different size chunks generated from the ten multivariate normal distributions. For the balanced synthetic dataset (BSYN) we have made two orderings viz. the points belonging to class +1 at the end (EBSYN) and the points from the two classes distributed (DBSYN). For each of the unbalanced synthetic datasets (USUN1, USYN2 and USYN3) four orderings of points belonging to the two classes viz. the points belonging to class +1 at the beginning (BUSYN), at the middle (MUSYN), at the end (EUSYN) and distributed (DUSYN), are made. For each of these orderings the performance of CVM is evaluated for BSYN, USYN1, USYN2 and USYN3. The parameters used in these experiments are given in Table 1 and the results in Table 2, Table 3 and Table 4 and Table 5 for BSYN, USYN1, USYN2 and USYN3 respectively.

From the results on BSYN, it can be seen that the performance of CVM is not affected by the orderings of data points from different classes within the dataset. Here ICBCVM also gave a comparable generalisation performance. The HCBCVM gave good generalisation performance at a much lesser computational expense.

Table 1 Parameters used with synthetic datasets for CVM, ICBCVM and HCBCVM algorithms

Dataset	*CVM*	*ICBCVM*			*HCBCVM*					
	C	*q*	*Thr1*	*Thr2*	*B*	*BS*	*T1*	*T2*	*Tol*	*q*
BSYN	1,000	0.25	1.17	1.173	50	50	0.85	0.83	0.0001	0.25
USYN1	1,000	0.0075	0.065	0.0.04	100	50	0.095	0.97	0.0001	0.1
USYN2	10,000	0.0075	0.059	0.0375	50	100	0.002	0.39	0.0001	0.055
USYN3	10,000	0.0075	0.059	0.0375	50	100	0.08	0.39	0.0001	0.055

Table 2 Results on balanced synthetic dataset

Dataset	*Algorithm*	*Training*			*Testing*	
		#TR	*#SVs*	*TT*	*CT*	*GP*
ICBCVM on BSYN		387,755	77	137	1	99.988
HCBCVM on BSYN		10,018	27	14	1	99.985
EBSYN	CVM	400,000	105	162.73	2.46	99.994
DBSYN	CVM	400,000	105	173.38	2.46	99.994

Note: The abbreviations used are: number of training points (# TR), number of support vectors (# SVs), training time (TT) in seconds, classification time (CT) for test data in seconds and generalisation performance (GP) on test data in percentage.

Table 3 Results on unbalanced synthetic dataset 1

Dataset	*Algorithm*	*Training*			*Testing*	
		#TR	*#SVs*	*TT*	*CT*	*GP*
ICBCVM on USYN1		62,575	1,203	375	17	98.454
HCBCVM on USYN1		35,263	17	88	1	99.594
BUSYN1	CVM	400,000	1,699	6,910.16	24.63	99.831
MUSYN1	CVM	400,000	1,703	7,174.55	24.79	99.832
EUSYN1	CVM	400,000	1,699	8,082.96	24.63	99.831
DUSYN1	CVM	400,000	1,700	7,785.12	24.77	99.834

Note: The abbreviations used are: number of training points (# TR), number of support vectors (# SVs), training time (TT) in seconds, classification time (CT) for test data in seconds and generalisation performance (GP) on test data in percentage.

Table 4 Results on unbalanced synthetic dataset 2

Dataset	*Algorithm*	*Training*			*Testing*	
		#TR	*#SVs*	*TT*	*CT*	*GP*
ICBCVM on USYN2		59,840	641	716	10	97.315
HCBCVM on USYN2		35,825	32	81	3	98.63
BUSYN2	CVM	400,000	257	1,400	8	90.029
MUSYN2	CVM	400,000	48	16	3	98.609
EUSYN2	CVM	400,000	253	14,875	7	90.163
DUSYN2	CVM	400,000	416	25,947	11	90.138

Note: The abbreviations used are: number of training points (# TR), number of support vectors (# SVs), training time (TT) in seconds, classification time (CT) for test data in seconds and generalisation performance (GP) on test data in percentage.

Table 5 Results on unbalanced synthetic dataset 3

Dataset	*Algorithm*	*Training*			*Testing*	
		#TR	*#SVs*	*TT*	*CT*	*GP*
ICBCVM on USYN3		51,981	607	408	14	99.023
HCBCVM on USYN3		13,105	24	145	4	98.805
BUSYN3	CVM	400,000	321	16,153	11	92.097
MUSYN3	CVM	400,000	632	26,717	19	89.14
EUSYN3	CVM	400,000	401	20,135	13	91.909
DUSYN3	CVM	400,000	260	4,898	9	94.957

Note: The abbreviations used are: number of training points (# TR), number of support vectors (#SVs), training time (TT) in seconds, classification time (CT) for test data in seconds and generalisation performance (GP) on test data in percentage.

In the case of USYN1, the results show that the performance of CVM is good irrespective of the orderings of patterns belonging to the two classes within the dataset. In all the orderings of USYN1, CVM gave good generalisation performance at comparable computational expense. This is due to the success of the sampling method used in CVM on this dataset having uniform distribution of points generated from the most of the

normal distributions (all except the one labelled as class +1 which is placed differently in the four different orderings of USYN1) within the dataset. Here ICBCVM and HCBCVM methods also gave reasonable generalisation performance at lesser computational expense.

In the case of USYN2 and USYN3, the performance of CVM is highly dependent on the orderings of the points within the dataset. Further, in most of these orderings CVM gives only moderate generalisation performance at a very high computational expense. This behaviour can be attributed to the sensitivity of the sampling method used in CVM on the orderings of data points within the dataset. Thus, the different orderings of points within the dataset results in highly variable performance for the CVM. For both USYN2 and USYN3, the ICBCVM and the HCBCVM algorithms are found to give good performance compared to CVM at a lesser computational expense.

5.2 Real world datasets

The real world datasets used are: IJCNN1 and Adult(a9a) datasets from the LIBSVM page available at 'http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html' and the extended USPS and forest cover type datasets available at 'http://www.cs.ust.hk/~ivor/cvm.html'. The parameters used for the experiments with IJCNN1 dataset, adult dataset and extended USPS dataset are shown in Table 6.

Table 6 Parameters used with IJCNN1 dataset, adult dataset and extended USPS dataset for CVM, ICBCVM and HCBCVM algorithms

Dataset	*CVM*	*ICBCVM*			*HCBCVM*					
	C	*q*	*Thr1*	*Thr2*	*B*	*BS*	*T1*	*T2*	*Tol*	*q*
IJCNN1	10,000	0.75	0.25	0.15	40	100	0.1	0.09	0.0001	3.0051
Adult	0.1	0.0075	0.115	0.115	60	135	1.75	9e-5	0.0001	2.2
EUSPS	100	0.0004	0.22	0.27	60	150	0.96	0.89	0.0001	0.00569

5.2.1 IJCNN1 dataset

This dataset pertains to a two class problem with 49,990 patterns in the training set and 91,701 patterns in the test set. Each pattern here is described using 22 numerical features. The results obtained on this dataset are shown in Table 7.

5.2.2 Adult dataset

This dataset pertains to a two class problem with 32,561 patterns in the training set and 16,281 patterns in the test set. Each pattern here is described using 123 numerical features. The results obtained on this dataset are shown in Table 7.

5.2.3 Extended USPS dataset

This dataset pertains to a two class problem with 266,079 patterns in the training set and 75,383 patterns in the test set. Each pattern here is described using 676 numerical features. The results obtained on this dataset are shown in Table 7.

Table 7 Results on IJCNN1, adult and extended USPS datasets

Dataset	*Algorithm*	*Training*			*Testing*	
		#TR	*#SVs*	*TT*	*CT*	*GP*
IJCNN1	CVM	49,990	866	372	26.4	95.832
	ICBCVM	16,637	485	62	10	95.832
	HCBCVM	4,641	310	22	13	96.003
Adult	CVM	32,561	21,595	5,216	94	85.265
	ICBCVM	1,317	829	193	3	81.365
	HCBCVM	16,923	12,438	3,249	167	83.25
EUSPS	CVM	266,079	996	470	134	99.556
	ICBCVM	208,885	834	467	69	99.4773
	HCBCVM	3,376	239	182	75	99.014

Note: The abbreviations used are: number of training points (# TR), number of support vectors (# SVs), training time (TT) in seconds, classification time (CT) for test data in seconds and generalisation performance (GP) on test data in percentage.

5.2.4 Forest cover type dataset

This dataset pertains to a two class problem with 522,910 patterns in the training set and 58,102 patterns in the test set. Each pattern here is described using 54 numerical features. Here the HCBCVM got trained in 3,166 seconds using 46,058 pattern obtained from the original training set. In this experiment the HCBCVM algorithm gave a generalisation performance of 90.105%. The parameter values used in the experiment are: B = 80, buffer size = 200, T1 = 1.982, T2 = 1.982, tolerance = 0.001 and q = $1e-4$.

To study the scalability of CVM, ICBCVM method and HCBCVM method we have done experiments with IJCNN1 dataset, EUSPS dataset and a synthetic dataset generated using the technique employed to generate USYN3. In these experiments, we have sampled the training set keeping the class distributions. Training samples of different sizes are obtained and the corresponding training time taken by CVM, ICBCVM and HCBCVM algorithms are recorded. A plot of the increase in training time (in seconds) with the increase in training samples size for IJCNN1 dataset, EUSPS dataset and a synthetic dataset are shown in Figure 5, Figure 6 and Figure 7 respectively. It can be observed that the increase in training time for ICBCVM and HCBCVM are comparatively slower when compared to that of CVM with the increase in the training set size. This is particularly visible in experiments with IJCNN1 and synthetic dataset.

From all these empirical results we have observed that the proposed ICBCVM method and the HCBCVM method improves the performance of CVM both in terms of generalisation results and the training time involved. Both ICBCVM and HCBCVM methods are able to produce good results even in unbalanced datasets irrespective of the orderings of data points belonging to the different classes within the dataset. Since these methods work with cluster abstractions generated using incremental clustering techniques they are scalable and hence can handle larger datasets compared to the ones handled by CVM.

Figure 5 A plot of training time (in seconds) of CVM, ICBCVM and HCBCVM with the training set size for IJCNN1 dataset

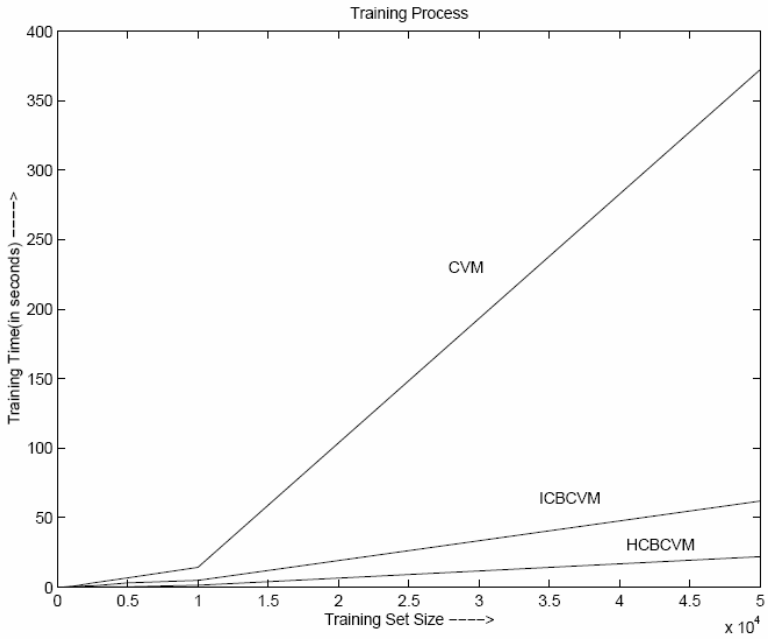


Figure 6 A plot of training time (in seconds) of CVM, ICBCVM and HCBCVM with the training set size for EUSPS dataset

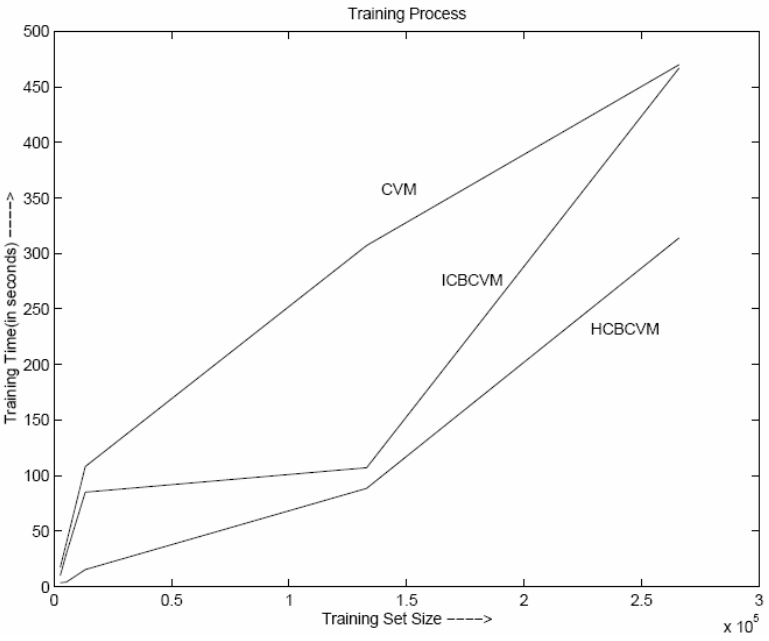
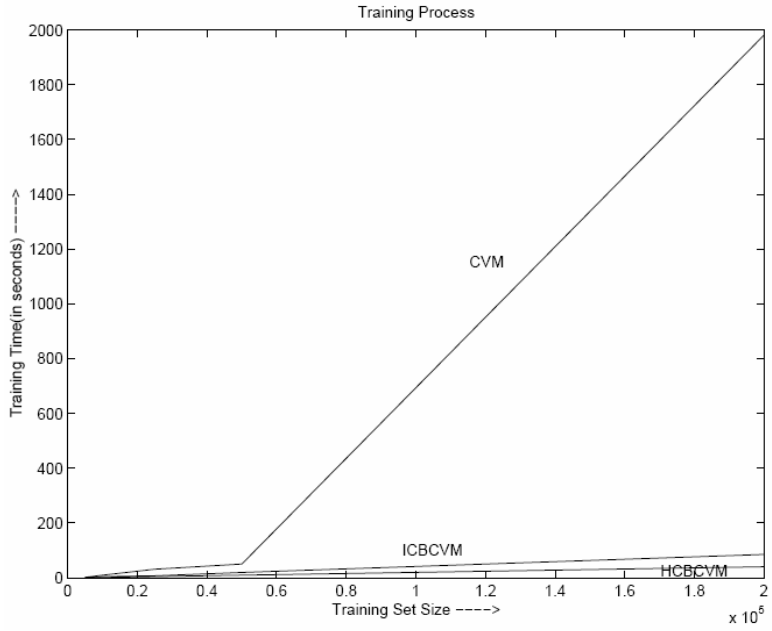


Figure 7 A plot of training time (in seconds) of CVM, ICBCVM and HCBCVM with the training set size for synthetic dataset



6 Conclusions

Two selective sampling-based training schemes for CVM to handle large datasets, irrespective of the orderings of data points belonging to different classes within the dataset, are proposed in this paper. Some merits of the proposed methods are:

- They can handle large datasets with linear/non-linear decision boundaries.
- They can handle datasets irrespective whether they are balanced/unbalanced in terms of the number of data points belonging to the two classes involved.
- They generate the required cluster abstractions in the kernel induced feature space using a single dataset scan and hence the methods are scalable.
- The performance of these methods does not depend on the orderings of data points within the dataset.

References

Asharaf, S., Murty, M.N. and Shevade, S.K. (2006) 'Cluster based core vector machine', *Proceedings of the 6th International Conference on Data Mining (ICDM 06)*, pp.1038–1042, Hong Kong.

Badou, M. and Clarkson, K.L. (2002) 'Optimal core sets for balls', *DIMACS Workshop on Computational Geometry*.

Cao, D. and Boley, D. (2003) *Training Support Vector Machine using Adaptive Clustering*, Technical Report No. 03-042, Computer Science and Engineering, University of Minnesota.

Cauwenberghs, G. and Poggio, T. (2001) 'Incremental and decremental support vector machine learning', *Advances in Neural Information Processing Systems*, Vol. 13, pp.409–415, MIT Press.

Diehl, C.P. and Cauwenberghs, G. (2003) 'SVM incremental learning, adaptation and optimization', *Proceedings of the IEEE International Joint Conference Neural Networks (IJCNN)*, pp.20–23, Portland, Oregon.

Fletcher, R. (2000) *Practical Methods of Optimization*, 2nd ed., Wiley-Interscience, New York.

Friess, T., Cristianini, N. and Cambell, C. (1998) 'The Kernel-Adatron algorithm: a fast and simple learning procedure for support vector machines', *Proceedings of the Fifteenth International Conference on Machine Learning*, pp.188–196, Madison, Wisconsin, USA.

Fung, G. and Mangasarian, O.L. (2002) 'Incremental support vector machine classification', *Proceedings of the Second SIAM International Conference on Data Mining*, pp.247–260, Arlington, VA, USA.

Fung, G. and Mangasarian, O.L. (2003) 'Finite Newton method for Lagrangian support vector machine classification', *Neurocomputing*, Vol. 55, Nos. 1–2, pp.39–55.

Gou, J., Yi, Z. and Xiong, L.D. (2012) 'A local mean-based k-nearest centroid neighbor classifier', *The Computer Journal*.

James, A.P. and Dimitrijev, S. (2012) 'Nearest neighbor classifier based on nearest feature decisions', *The Computer Journal*.

Lee, Y. and Mangasarian, O.L. (2001) 'RSVM: reduced support vector machines', *CD Proceedings of the First SIAM International Conference on Data Mining*, Chicago.

Loosli, G. and Canu, S. (2007) 'Comments on the core vector machines: fast SVM training on very large data sets', *Journal of Machine Learning Research*, Vol. 8, pp.291–301.

Mangasarian, O.L. and Musicant, D.R. (2001a) 'Lagrangian support vector machines', *Journal of Machine Learning Research*, Vol. 1, pp.161–177.

Mangasarian, O.L. and Musicant, D.R. (2001b) 'Proximal support vector machine classifiers', *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.77–86, San Francisco.

Mangasarian, O.L. and Thompson, M.E. (2006) 'Massive data classification via unconstrained support vector machines', *Journal of Optimization Theory and Applications*, Vol. 131, No. 3, pp.315–325.

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

Platt, J.C. (1999) 'Fast training of support vector machines using sequential minimal optimization', in Scholkopf, B., Burges, C. and Smola, A. (Eds.): *Advances in Kernel Methods – Support Vector Learning*, pp.185–208, MIT Press, Cambridge, MA.

Smola, A.J. and Scholkopf, B. (2000) 'Sparse greedy matrix approximation for machine learning', *Proceedings of the Seventeenth International Conference on Machine Learning*, pp.911–918, Stanford, CA, USA.

Spath, H. (1980) *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*, Ellis Horwood Limited, West Sussex, UK.

Suykens, J.A.K. and Vandewalle, J. (1999) 'Least squares support vector machine classifiers', *Neural Processing Letters*, Vol. 9, No. 3, pp.293–300.

Tax, D. and Laskov, P. (2003) 'Online SVM learning: from classification to data description and back', *Proceedings of Neural Network and Signal Processing*, pp.499–508.

Tsang, I.W., Kwok, J.T. and Cheung, P-M. (2005) 'Core vector machines: fast SVM training on very large data sets', *Journal of Machine Learning Research*, Vol. 6, pp.363–392.

Vapnik, V.N. (1998) *Statistical Learning Theory*, Wiley-Interscience, New York, USA, ISBN-10: 0471030031, ISBN-13: 978-0471030034.

Viswanathan, S.V.N., Smola, A.J. and Murty, M.N. (2003) 'Simple SVM', *Proceedings of the Twentiest International Conference on Machine Learning*, pp.760–767, Washington, DC, USA.

Wang, J., Wu, X. and Zhang, C. (2005) 'Support vector machines based on K-means clustering for real-time business intelligence systems', *International Journal of Business Intelligence and Data Mining*, Vol. 1, pp.54–64.

Williams, C.K.I. and Seeger, M. (2001) 'Using the Nystrom method to speed up kernel methods', *Advances in Neural Information Processing Systems*, Vol. 13, pp.682–688, MIT Press.

Yu, H., Yang, J. and Han, J. (2003) 'Classifying large data sets using SVM with hierarchical clusters', *Proceedings of the 2003 ACM SIGKDD*, pp.306–315, Washington, DC.

Zhang, T., Ramakrishnan, R. and Livny, M. (1996) 'BIRCH: an efficient data clustering method for very large databases', *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.103–114, Montreal, Canada, ACM Press, New York.