Data Mining and Knowledge Discovery, 11, 295–321, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.
DOI: 10.1007/s10618-005-0005-7

Making SVMs Scalable to Large Data Sets using Hierarchical Cluster Indexing

HWANJO YU hwanjoyu@cs.uiowa.edu
Department of Computer Science, University of Iowa

JIONG YANG jiong@eecs.cwru.edu
Department of Computer Science, Case Western Reserve University

JIAWEI HAN hanj@cs.uiuc.edu
XIAOLEI LI xli10@uiuc.edu
Department of Computer Science, University of Illinois at Urbana-Champaign

Received January 1, 2004; Accepted June 1, 2005

Published online: 19 August 2005

Abstract. Support vector machines (SVMs) have been promising methods for classification and regression analysis due to their solid mathematical foundations, which include two desirable properties: margin maximization and nonlinear classification using kernels. However, despite these prominent properties, SVMs are usually not chosen for large-scale data mining problems because their training complexity is highly dependent on the data set size. Unlike traditional pattern recognition and machine learning, real-world data mining applications often involve huge numbers of data records. Thus it is too expensive to perform multiple scans on the entire data set, and it is also infeasible to put the data set in memory. This paper presents a method, *Clustering-Based SVM (CB-SVM)*, that maximizes the SVM performance for very large data sets given a limited amount of resource, e.g., memory. CB-SVM applies a hierarchical micro-clustering algorithm that scans the entire data set only once to provide an SVM with high quality samples. These samples carry statistical summaries of the data and maximize the benefit of learning. Our analyses show that the training complexity of CB-SVM is quadratically dependent on the number of support vectors, which is usually much less than that of the entire data set. Our experiments on synthetic and real-world data sets show that CB-SVM is highly scalable for very large data sets and very accurate in terms of classification.

1. Introduction

Support vector machines (SVMs) have been promising methods for data classification and regression analysis (Vapnik, 1998; Burges, 1998; Joachims, 1998; Chang and Lin 2001; Smola and Scholkopf, 1998; Yu et al., 2002). Their successes can be attributed to their solid mathematical foundations, which yield several salient properties:

A preliminary version of the paper, "*Classifying Large Data Sets Using SVM with Hierarchical Clusters*", by *H. Yu, J. Yang*, and J. Han, appeared in *Proc. 2003 Int. Conf. on Knowledge Discovery in Databases (KDD'03)*, Washington, DC, August 2003. However, this submission has substantially extended the previous paper and contains new and major-value added technical contribution in comparison with the conference publication.

- *Margin maximization*: The classification boundary functions of SVMs maximize the margin, which, in machine learning theory, corresponds to maximizing the *generalization* performance. (See Section 2 for more details.)
- *Systematic nonlinear classification via kernel tricks*: SVMs efficiently handle nonlinear classifications using kernel tricks, which implicitly transforms the input space into a high dimensional feature space.

The success of SVMs in machine learning naturally leads to its possible extension to large-scale data mining. However, despite SVMs' prominent properties, they are not as favorably used in large-scale data mining as in pattern recognition or machine learning. The main reason is that the training complexity of SVMs is highly dependent on the size of the data set. (It is known to be at least quadratic to the number of data points. Refer to Chang and Lin (2001) for more discussions on the complexity of SVMs.) Many real-world data mining applications often involve millions or billions of data records. For such large data sets, existing SVM implementations (1) generate system failures due to memory blowup, or (2) take "forever" to finish training with unbounded memory. In our experiments in Section 5, two efficient SVM implementations, LIBSVM and Active SVM, both generated system failures for large data sets.

Researchers have proposed various revisions of SVMs to increase the training efficiency by either mutating or approximating it. However, these solutions are still not appropriate for very large data sets where even multiple scans of the data are too expensive to perform. (see Section 7 for the discussions on related work.)

This paper presents a new approach for scalable and reliable SVM classification, called *Clustering-Based SVM (CB-SVM)*. It is designed specifically for handling very large data sets given a limited amount of system resource, e.g., memory. When the size of the data set is large, traditional SVMs tend to perform worse when trained with the entire data than with a set of fine-quality samples (Schohn and Cohn, 2000). The SVM-based selective sampling (or active learning) techniques try to select the training data intelligently to maximize the performance of SVM, but they normally require many scans of data set (Schohn and Cohn, 2000; Tong and Koller, 2000) (Section 7). Our CB-SVM applies a hierarchical micro-clustering algorithm that scans the entire data set only once to provide an SVM with high quality *samples*. These samples carry statistical summaries of the data such that the learning quality of the SVM is maximized. CB-SVM is scalable in terms of the training efficiency while maximizing the performance of SVM with bounded memory. The hierarchical micro-clusters conceptually play the role of indexing an SVM. By using the hierarchical micro-clustering algorithm *BIRCH* (Zhang et al., 1996), CB-SVM also automatically controls the trade-off between memory size and classification accuracy.

The key idea of CB-SVM is to use the hierarchical micro-clustering technique to get finer description at places close to the classification boundary and coarser description at places far from the boundary. This is efficiently processed as follows. CB-SVM first constructs two micro-cluster trees called *CF-trees* from positive and negative training data respectively. In each tree, a node in a higher level is a summarized representation of its children nodes. After constructing the two trees, CB-SVM starts training an SVM only from the root nodes of the CF-trees. Once it generates a "rough" boundary from the root nodes, it selectively declusters only the data summaries near the classification boundary into lower (or finer) levels in the CF-tree structure. The hierarchical representation of the data summaries is a perfect base structure for CB-SVM to effectively perform the selective declustering. CB-SVM repeats this selective declustering process down to the leaf level.

CB-SVM can be used to classify very large data sets of relatively low dimensionality, such as streaming data or data in large data warehouses. It performs especially well where random sampling is not effective. This occurs when the "important" data occur infrequently or when the incoming data includes irregular patterns, which results in different distributions between training and testing data. We discuss this more in detail in Section 5.1.3. Our experiments on the network intrusion data set (Section 5.2), a good example which shows that random sampling could hurt, show that CB-SVM is scalable for very large data sets while still yielding high classification accuracy. Due to the limitations of the clustering-based indexing structure, CB-SVM performs not as well for high dimensional data. Developing an indexing structure for high dimensional data is an important future research direction.

To the best of our knowledge, the proposed method is currently the only SVM for very large data sets that tries to generate the best results with bounded memory.

The remainder of the paper is organized as follows. We first overview SVM in Section 2. In Section 3, we introduce a hierarchical micro-clustering algorithm for very large data sets, originally studied by Zhang et al. (1996). In Section 4, we present the CB-SVM algorithm that applies the hierarchical micro-clustering algorithm to a standard SVM to make the SVM scalable for very large data sets. Section 5 demonstrates experimental results on artificial and real data sets. Section 6 presents an extension of CB-SVM to the SVM nonlinear kernels. We discuss the related work in Section 7 and conclude our study in Section 8.

2. SVM overview

In machine learning theory, given a set of training data set $\{(x, y)\}$ (y is the label of x), the optimal class boundary function (or hypothesis) $g(x)$ is the one that gives the best *generalization performance*, i.e., performance on *unseen* examples rather than the training data. Classification performance on the training data is not regarded as a good evaluation measure for a hypothesis, because the hypothesis is usually *overfitted* for that particular set of data through training. SVM is a supervised learning method that tries to maximize the generalization performance by maximizing the *margin* (Burges, 1998; Vapnik, 1998). The *margin* in SVM denotes the distance from the class boundary to the closest data points in the feature space. In situations where a linear boundary is insufficient, SVM also supports nonlinear classification using kernel tricks.

In SVM, the problem of computing a margin-maximizing boundary function is specified by the following quadratic programming (QP) problem:

$$\text{minimize}: \quad W(\alpha) = -\sum_{i=1}^{l}\alpha_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j k(x_i, x_j)$$

$$\text{subject to}: \sum_{i=1}^{l} y_i \alpha_i = 0$$
$$\forall i : 0 \leq \alpha_i \leq C$$

l denotes the number of training data. α is a vector of l variables, where each component α_i corresponds to a training data (x_i, y_i). Lastly, C is the soft margin parameter which controls the influence of the outliers (or noise) in the training data.

The kernel $k(x_i, x_j)$ for linear boundary function is $x_i \cdot x_j$, a scalar product of two data points. The nonlinear transformation of the feature space is performed by replacing $k(x_i, x_j)$ with an advanced kernel, such as a polynomial kernel $(x^T x_i + 1)^p$ or a RBF kernel $\exp(-\frac{1}{2\sigma^2}||x - x_i||^2)$. An advanced kernel is a function that operates on the input data but has the effect of computing the scalar product of their images in a usually much higher-dimensional feature space (or even an infinite-dimensional space). This has several advantages. First, it allows one to work implicitly with hyperplanes in highly complex spaces, which could yield better classification results. Second, it is an attractive computational short-cut, because it foregoes the expensive creation of a complicated feature space.

Another characteristic of SVM is that its boundary function is described by the *support vectors* (SVs), which are data points located closest to the class boundary. The above QP problem computes a vector α, where each element specifies a weight on each data point (x_i, y_i). The set of data points whose corresponding α_i is greater than zero is exactly the set of SVs. The data points whose α_i's are less than or equal to zero do not contribute to the boundary function at all. In other words, computing an SVM boundary function can be viewed as finding data points with nonzero weights in α.

There have been many attempts to revise the original QP formulation so that it can be solved by a QP solver more efficiently (Mangasarian and Musicant, 2000; Fung and Mangasarian, 2001; Agarwal, 2002). (See Section 7 for more details.) In contrast to those papers, we do not revise the original QP formulation of SVM. Instead, we try to provide a smaller but high quality data set that is beneficial to effectively computing the SVM boundary function by applying a hierarchical clustering algorithm. Our CB-SVM algorithm substantially reduces the total number of data points for training an SVM while trying to keep the high quality SVs that best describe the boundary.

3. Hierarchical micro-clustering algorithm for large data sets

The hierarchical micro-clustering algorithm we present here and will apply to our CB-SVM in Section 4 was originally studied by Zhang et al. (1996) and is called *BIRCH*. It introduces the concept of a "micro-cluster" (Zhang et al., 1996), which stores a statistical summary representation of a group of data points that are so close together that they are likely to belong to the same cluster. Our hierarchical micro-clustering algorithm has the following characteristics.

- It constructs a micro-cluster tree, called *CF* (*Clustering Feature*) *tree*, in one scan of the data set given a limited amount of resources (i.e., available memory and time constraints) by incrementally and dynamically clustering incoming multi-dimensional data points.

- Since a single scan of data does not allow backtracking, localized inaccuracies may exist depending on the order of data input. Fortunately, the CF tree captures the major distribution patterns of the data and provides enough information for CB-SVM to perform.
- It handles noise or outliers (data points that are not a part of the underlying distribution) effectively as a by-product of the clustering.

Other hierarchical clustering algorithms have also been developed, including STING (Wang et al., 1997), CURE (Guha et al., 1998), Chameleon (Karypis et al., 1999). STING is a grid-based clustering method in which the spatial data is divided with equal space without considering the data distribution. Chameleon has shown to be powerful at discovering arbitrarily shaped clusters of high quality, but its complexity in the worst case is $O(n^2)$, where n is the number of data points. CURE produces high-quality clusters with complex shapes, and its complexity is also linear to the number of objects. However, its parameter in general has a significant influence on the results. Setting them to maximize performance is a non-trivial task.

The CF tree of BIRCH carries spherical shapes of hierarchical clusters and captures the statistical summaries of the entire data set. It provides an efficient and effective data structure for CB-SVM. BUBBLE, an revision of the BIRCH algorithm, was proposed later (Ganti et al., 1999) for arbitrary metric spaces. It constructs a similar structure called CF^* tree for clustering. CF^* tree assigns an actual object in a cluster as the cluster center to facilitate clustering in any distance space. Due to its similarities to the original CF tree, it is an alternative structure for CB-SVM.

3.1. *Clustering feature and CF tree*

First, we define some basic concepts. Given N d-dimensional data points in a cluster: $\{x_i\}$ where $i = 1, 2, \ldots, N$, the centroid C and radius R of the cluster are defined as,

$$C = \frac{\sum_{i=1}^{N} x_i}{N} \tag{1}$$

$$R = \left(\frac{\sum_{i=1}^{N} ||x_i - C||^2}{N}\right)^{\frac{1}{2}} \tag{2}$$

where R is the average distance from all the member points to the centroid.

The concept of the *clustering feature* (CF) tree is at the core of the hierarchical micro-clustering algorithm which makes the clustering incremental without expensive computations. A CF vector is a triplet which summarizes the information that a CF tree maintains for a cluster.

Definition 1 (Clustering feature (Zhang et al., 1996)). Given n d-dimensional data points in a cluster: $\{x_i\}$ where $i = 1, 2, \ldots, n$, the CF vector of the cluster is defined as a triple: $CF = (n, LS, SS)$, where n is the number of data points in the cluster, LS is the linear sum of the n data points, i.e., $\sum^{n}{}_{i=1} x_i$, and SS is the component-wise square sum of the n data points, i.e., $\sum^{n}{}_{i=1} x_i{}^2$.

Theorem 1 *(CF Additivity Theorem (Zhang et al., 1996)). Assume that* $CF_1 = (n_1, LS_1, SS_1)$ *and* $CF_2 = (n_2, LS_2, SS_2)$ *are the CF vectors of two disjoint clusters. Then the CF vector of the cluster formed by merging the two disjoint clusters is:*

$$CF_1 + CF_2 = (n_1 + n_2, LS_1 + LS_2, SS_1 + SS_2) \tag{3}$$

Refer to Zhang et al., 1996 for the proof.

From the CF definition and additivity theorem, we can see that the CF vectors of clusters can be stored and calculated incrementally and accurately as clusters are merged. The centroid C and the radius R of each cluster can be also computed from the CF vector.

As mentioned before, the CF serves as an efficient summary for a set of data points. It can save space significantly for densely packed data points, and it is also sufficient for calculating all the necessary information for building the hierarchical micro-clusters. All of these properties combine to facilitate computing an SVM boundary for very large data sets.

3.1.1. CF tree. A CF tree is a height-balanced tree with two parameters: branching factor b and threshold t. A CF tree of height $h = 3$ is shown in the right side of Figure 1. Each nonleaf node consists of at most b entries of the form $(CF_i, child_i)$, where (1) $i = 1, 2, \ldots, b$, (2) $child_i$ is a pointer to its i-th child node, and (3) CF_i is the CF of the cluster represented by $child_i$. A leaf node has only a *leaf entry*, which contains just the CF for that node. The tree is hierarchical in the following sense: the CF at any node contains information for all data points in that node's subtree. Lastly, all leaf entries have to satisfy the threshold t constraint, which restricts the radius of an entry to be less than t.

The tree size is a function of t. The larger t is, the smaller the tree is. The branching factor b can be determined by memory page size such that a leaf or a nonleaf node fits in a page.

The CF tree is a compact representation of the data set, because each entry in a leaf node is not a single data point but a cluster, which absorbs many data points within a radius of t or less.

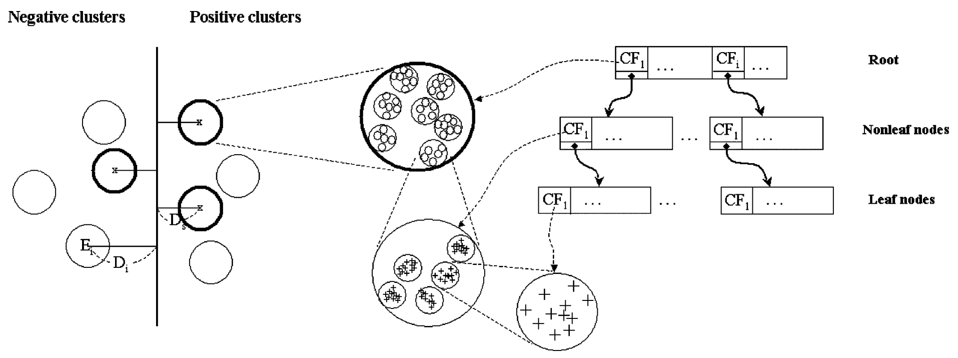

Figure 1. Example of the SVM boundary trained from the root entries of positive and negative trees.

3.2. *Algorithm description*

A CF tree is built up dynamically as new data objects are inserted. Insertion is similar to that of a B+-tree. The sketch is given below.

1. *Identifying the appropriate leaf*: Starting from the root, it traverse the CF tree down to the leaf level by choosing the child node whose centroid is closest to the new data object at each level.
2. *Modifying the leaf*: If the leaf entry can absorb the new data object without violating the t threshold condition, update the CF vector of the entry and terminate. Otherwise, add a new leaf entry. If adding a new entry violates the b threshold (i.e., too many children), split the parent node by choosing the farthest pair of entries as seeds and redistribute the remaining entries based on the closeness.
3. *Modifying the path to the leaf*: If the previous step of modifying a leaf entry caused a node split, check the parent node for satisfaction of the branching factor constraint. If the parent node violates the b threshold as well, split it and recursively traverse back up to the root while performing the same checks.

Due to the limited number of entries in a node, a highly skewed input could cause two subclusters that should have been in one cluster split across different nodes, and vice versa. These infrequent but undesirable anomalies are handled in the original BIRCH algorithm by further refinement with additional data scans. However, we do not perform such refinements, because the infrequent and localized inaccuracy do not heavily impact the performance of CB-SVM.

3.2.1. Threshold determination. The choice of the threshold t is crucial for building the CF tree whose size fits in the available memory; because if t is too small, we run out of memory before all the data are scanned. The original BIRCH algorithm initially sets t very low, and iteratively increases t until the tree fits into memory. Zhang et al. proved that rebuilding the tree with a larger t requires a re-scan of the data inserted in the tree so far and at most h extra pages of memory, where h is the height of the tree (Zhang et al., 1996). The heuristics for updating t_i is also provided in (Zhang et al., 1996). Due to the space limitation and to keep the focus of the paper, we skip the details of our implementation. In our experiments, we set the initial threshold t_1 intuitively based on the number of data points n, the dimensionality d, and the value range r_d of each dimension such that t_1 is proportional to $n \times d \times r_d$. Usually, the tree of t_1 fits in memory.

3.2.2. Outlier handling. After the construction of a CF tree, the leaf entries that contain far fewer data points than average are considered to be *outliers*. A non-trivial amount of outliers or noise in the training data is undesirable because it complicates the boundary function and could prevent the SVM boundary from converging in the quadratic programming. Therefore, we would like to eliminate these outliers during training. The precise determination of outlier entries is achieved by a threshold value. A non-zero but small outlier threshold can significantly improve the classification performance of CB-SVM. This is especially evident when the data set size is large compared to the number of dimensions and the boundary functions are simple. The rationale behind it is analogous to having a low VC dimension in machine learning theory.

We enable the outlier handling with a low threshold in our experiments in Section 5 because the type of data that we target has a large number of data points with relatively low dimensionality, and the boundary function is linear to the VC dimension $m + 1$ where m is the number of dimensions. See Section 5 for more details.

3.2.3. Analysis. A CF tree that fits in memory can have at most $\frac{M}{P}$ nodes, where M is the size of memory and P is the size of a node. The height h of a tree is $\log_b \frac{M}{P}$, independent of the data set size. If we assume that memory is unbounded and the number of the leaf entries is equal to the number of data points N (i.e., a very small t threshold), then $h = \log_b N$.

Insertion of a node into a tree requires the examination of $O(b * h)$ entries, as traversing through the tree needs at most $b * h$ examinations. The cost per entry is proportional to the dimension d. Thus, the cost of inserting N data points is $O(N \times d \times b \times h)$. In case of rebuilding the tree due to the poor estimation of t_1, additional re-insertions of the data already inserted has to be added into the cost. Then the cost becomes $O(k \times N \times d \times b \times h)$ where k is the number of the rebuildings. If we only consider the dependence on the data set size, the computation complexity of the algorithm is $O(N)$. Experiments from the original BIRCH algorithm have also shown the linear scalability of the algorithm with respect to the number of data points.

4. Clustering-based SVM (CB-SVM).

In this section, we present the CB-SVM algorithm which trains a very large data set using the hierarchical micro-clusters (i.e., CF tree) to construct an accurate SVM boundary function.

The key idea of CB-SVM can be viewed as similar to that of selective sampling (or active learning), i.e., selecting the data that maximizes the benefit of learning. Classical selective sampling for SVMs chooses and accumulates the *low margin data* at each round, which are points that are close to the boundary in the feature space. Low margin data are of interest because they have higher chances to become the SVs of the boundary for the next round (Tong and Koller, 2000; Schohn and Cohn, 2000). Based on this idea, we decluster the node entries near the boundary to get finer sampling close to the boundary and coarser sampling far from the boundary. In this manner, we induce the SVs, the description of the class boundary, as fine as possible while keeping the total number of training data points as small as possible.

While traditional selective sampling needs to scan the entire data set at each round to select the closest data point, CB-SVM runs on the CF tree, which can be constructed in a single scan of the entire data set and carries the statistical summaries that facilitate efficient and effective construction of an SVM boundary. The sketch of the CB-SVM algorithm is as follows.

1. Construct two CF trees from positive and negative data set independently.

2. Train an SVM boundary function using the centroids of the root entries, i.e., the entries in the root node of the two CF trees. If the root node contains too few entries, train using the entries of the nodes at the second levels of the trees.
3. Decluster the entries near the boundary into the lower level in the CF trees. The newly declustered child entries are added to the training set.
4. Construct another SVM from the centroids of the entries in the training set, and repeat from step 3 until nothing is added to the set.

The CF tree is a suitable base structure for CB-SVM to efficiently perform selective declustering. The clustered data provide better summaries for SVM than random samples because random sampling is susceptible to biased or skewed input. Random samples may generate undesirable outputs especially when the probability distributions of training and testing data are not similar, which is common in practice. We discuss this in detail in Section 5.

4.1. CB-SVM description

Let us first consider linearly separable cases.

Let *positive tree* T_p and *negative tree* T_n be the CF trees built from the positive data set and the negative data set respectively. We first train an SVM boundary function g using only the *centroids of the root entries* in T_p and T_n. Note that each entry (or cluster) E_i contains all the necessary information to efficiently compute its centroid C_i and radius R_i. Figure 1 shows an example of the SVM boundary with the root clusters and the corresponding positive tree.

After the initial boundary function g is computed, we find the *low margin clusters* that are close to the boundary and decluster them into finer levels in the CF tree. Let *support clusters* be clusters whose centroids are SVs of the boundary g, e.g., the bold circles in figure 1. Let D_s be the distance from the boundary to the centroid of a support cluster s. If we are using hard margins, D_s is constant for all s. Also let D_i be the distance from the boundary to the centroid of cluster E_i. Then, we consider a cluster E_i which satisfies the following constraint as a *low margin cluster*.

$$D_i - R_i < D_s \tag{4}$$

where R_i is the radius of the cluster E_i. Descriptively, a low margin cluster is a cluster whose closest distance to the classification boundary is less than D_s.

The clusters that satisfy the constraint Eq. (4) are of interest, because their subclusters could become support clusters of the boundary. Figure 2 shows an example. On the left side, five clusters (circles with gray parts) initially satisfy the constraint Eq. (4); three of them are the support clusters, shown in bold. These five clusters are declustered into finer levels, shown in the right side of the figure. The smaller and more precise support clusters found in the next iteration are shown in bold. Note that the CF tree uses the "average" radius instead of the "maximal" radius of a cluster for efficiency (see Eq. (2).). Thus, it is possible that a few sub-clusters are outside the circles.

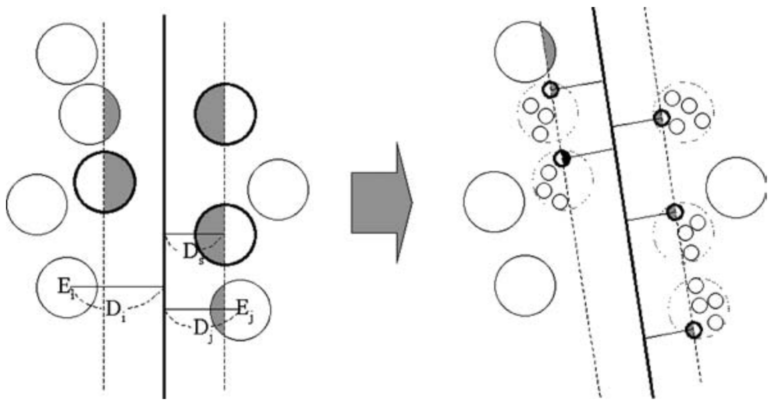

Figure 2. Declustering of the low margin clusters.

The subclusters whose parent clusters do not satisfy the constraint Eq. (4) *cannot* become support clusters of the boundary g in the next iteration. This is because their parent clusters are farther away from the boundary than the current SVs. From this, we have the following remark.

Remark 1 (Declustering constraint) For any cluster E_i, let R_i be the radius of the cluster and D_i be the distance from the class boundary to the centroid of the cluster. Given a separable set of positive and negative clusters $E = \{E_i{}^+\} \cup \{E_i{}^-\}$ and the SVM boundary g of the set, the subclusters of E_i have the possibilities to be the support clusters of the boundary g only if $D_i - R_i < D_s$, where D_s is the distance from the boundary to the centroid of a support cluster in the case of hard constraints.

The example we illustrated in Figure 2 was a linearly separable case with hard constraints on the SVM. In practice, soft constraints are usually necessary to cope with noise in the training set. Using soft constraints generates the SVs with different distances from the boundary. That is, the margin $f(s)/\|w\|$ (Vapnik, 1998) will be different for different support vector s. Thus, we use the average margin, $D_s{}'$ for the declustering constraint:

$$D'_s = \frac{\sum_{x \in S} \frac{f(x)}{||w||}}{|S|} \tag{5}$$

where S is the set of support vectors.

Tables 1 and 2 describe the CB-SVM algorithm.

4.2. *CB-SVM analysis*

As discussed in Section 3.2.3, building a CF tree has cost $O(N)$ where N is the number of data points. We disregard the number of dimensions in our analysis since it is linearly dependent. Once the CF tree is built, the training time of CB-SVM becomes dependent on the number of leaf entries in the tree instead of the number of data points.

Table 1. CB-SVM.

```
Input: - positive data set $\mathcal{P}$, negative data set $\mathcal{N}$
Output: - a boundary function $g$
Functions:
- $HC(\mathcal{S})$: return a hierarchical cluster tree $T$ from a data set $\mathcal{S}$
- getRootEntries($T$): return the entries in the root node of a tree $T$
- getChildren($\mathcal{S}$): return the children entries of an entry set $S$
- getLowMargin($g$, $\mathcal{S}$): return the low margin entries from a set $S$ which are close to the boundary $g$
  (See Table 2)
Algorithm:
// Construct a positive and a negative tree from $\mathcal{P}$ and $\mathcal{N}$ respectively
1. $T_p = HC(\mathcal{P})$; $T_n = HC(\mathcal{N})$;
   // Put the positive and negative root entries in initial training set $\mathcal{S}$
2. $\mathcal{S} := \{$getRootEntries($T_p$) $\cup$ getRootEntries($T_n$)$\}$;
3. Repeat
   3.1. $g :=$ SVM.train($\mathcal{S}$); // construct a boundary $g$
   3.2. $\mathcal{S}' :=$ getLowMargin($g$, $\mathcal{S}$); // compute the low margin entries $\mathcal{S}'$ from $\mathcal{S}$ using $g$
   3.3. $\mathcal{S} := \mathcal{S} - \mathcal{S}'$; // exclude the low margin data from $\mathcal{S}$
   3.3. $\mathcal{S}' :=$ getChildren($\mathcal{S}'$);
   3.4. Exit if $\mathcal{S}' = \emptyset$;
   3.5. $\mathcal{S} := \mathcal{S} \cup \mathcal{S}'$; // include the children of the low margin data to the training set $\mathcal{S}$
4. Return $g$;
```

Table 2. getLowMargin(g, $\mathcal{S}$).

```
Input: - a boundary function $g$, a entry set $\mathcal{S}$
Output: - a set of the low margin entries $\mathcal{S}'$
Algorithm:
   // return the average distance of the support vectors from the boundary $g$
1. $D_{SV}$ := getAveDistanceOfSVs($g$);
   // return the data whose margin is smaller than $D_{SV}$
2. $\mathcal{S}'$ := getLowerMarginData($D_{SV}$, $\mathcal{S}$);
3. Return $\mathcal{S}'$;
```

Let us assume $t(SVM) = O(N^2)$ where $t(\Psi)$ is the training time of algorithm Ψ. The number of the leaf entries is at most b^h. Thus, $t(SVM)$ becomes $O(b^{2h})$.

Let *support entries* be leaf entries that contain SVs. Let r be the percentage of training entries that are support entries, averaged over all training iterations. Formally, $r = 1/k \sum_{i=1}^{k} s_i/b_i$ where b_i is the number of training entries and s_i is the number of support entries in iteration i of k total. Usually, $s \ll b$ and $0 < r \ll 1$ for standard SVMs with large data sets.

Theorem 2 (Training Complexity of CB-SVM). *If the number of leaf entries in a CF tree is equal to the number of training data points N, then CB-SVM trains asymptotically 1/r^{2h-2} times faster than standard SVMs given the CF tree, where r is the average percentage of SVs in all training iterations and h is the height of the tree* ($h = \log_b N$).

Proof. If we approximate the number of iterations in CB-SVM $I \approx h$ (the height of CF tree), then the training complexity of CB-SVM given the CF tree is:

$$t(\text{CBSVM}) = \sum_{i=1}^{h} t_i(\text{CBSVM})$$

where t_i(CBSVM) is the training complexity of the i-th iteration of CB-SVM. The number of training data points N_i at the i-th iteration is:

$$\begin{aligned} N_i &= b - s + bs - s^2 + \cdots + bs^{i-2} - s^{i-1} + bs^{i-1} \\ &= (b - s)(1 + s + s^2 + \cdots + s^{i-2}) + bs^{i-1} \\ &= (b - s)\frac{s^{i-1} - 1}{s - 1} + bs^{i-1} \end{aligned}$$

where b is the number of data points in a node, and s is the number of the SVs among the data. If we assume t(SVM) $= O(N^2)$, by approximation of $s{-}1 \approx s$,

$$\begin{aligned} t_i(\text{CBSVM}) &= O\left(\left[bs^{i-2} + 1 - \frac{b}{s} - s^{i-1} + bs^{i-1}\right]^2\right) \\ &= O\left(\left[bs^{i-1}\right]^2\right) \end{aligned}$$

If we accumulate the training time of all iterations,

$$\begin{aligned} t(\text{CBSVM}) &= O\left(\sum_{i=1}^{h} [bs^{i-1}]^2\right) = O\left(b^2 \sum_{i=0}^{h-1} s^{2i}\right) \\ &= O\left(b^2 \frac{s^{2h} - 1}{s^2 - 1}\right) \approx O\left(b^2 s^{2h-2}\right) \end{aligned}$$

If we replace s with br since $r = s/b$,

$$t(\text{CBSVM}) = O\left(\left[b^h r^{h-1}\right]^2\right) = O\left(b^{2h} r^{2h-2}\right)$$

Therefore, t(CBSVM) trains asymptotically $1/r^{2h-2}$ times faster than t(SVM) which is $O(b^{2h})$ for $N = b^h$. □

From Theorem 2, where $t(\text{CBSVM}) \approx O(b^2\, s^{2h-2})$, we can see that the training time of CB-SVM is quadratically dependent on the number of support vectors (s^h) which is much less than that of the entire data (b^h). Normally $r \ll 1$, especially for very large data sets. So, the performance difference between CB-SVM and a standard SVM goes higher as the data set becomes larger.

5. Experimental evaluation

In this section, we provide empirical evidence of our analysis of CB-SVM using synthetic and real data sets and discuss the results. All the experiments are conducted on a Pentium III 800 MHz machine with 906 MB memory.

5.1. Synthetic data set

5.1.1. Data generator. To verify the performance of CB-SVM in a realistic environment while providing visualization of the results, we perform binary classifications on two-dimensional data sets generated as follows.

1. We randomly created K clusters each with the following properties. (1) The center point C is randomly chosen in the range $[C_l, C_h]$ for each dimension independently. (2) The radius R is randomly chosen in the range of $[R_l, R_h]$. (3) The number of points n in each cluster is also randomly chosen in the range of $[n_l, n_h]$.
2. Let C_i^x be the X-axis value of cluster E_i's centroid, and let θ be a threshold value between C_l and C_h. We labeled cluster E_i as *positive* if $C_i^x < \theta - R_i$ and *negative* if $C^x{}_i > \theta + R_i$. Clusters whose C_i^x did not satisfy either of these conditions, i.e., clusters that lied across the threshold value on the X-axis, were completely removed. In this manner, we forced the clusters to be linearly separable.
3. Once the characteristics of each cluster are determined, data points for the cluster are generated according to a 2D independent normal distribution whose mean is the center C and whose standard deviation is the radius R. The class label of each point is inherited from the label of its cluster. Note that due to the properties of the normal distribution, the maximum distance between a point in the cluster and the centroid is unbounded. In other words, a point may be arbitrarily far from its belonging cluster. We refer to these points as "outsiders". Because of these points, the data set becomes not completely linearly separable, which is more realistic.

5.1.2. SVM parameter setting. We used the LIBSVM[1] (version 2.36) implementation and used ν-SVM with linear kernel. We enabled the shrinking heuristics for fast training (Joachims, 1998). ν-SVM has an advantage over standard SVMs: the parameter ν has a semantic meaning which denotes the upper bound of the noise rate and the lower bound of the SV rate in training data (Chang and Lin, 2001). Thus, a very low ν (e.g., 0.01) performs very well when the data size is very large and noise is relatively small. For fair evaluation, we optimized ν using the Hold-out testing technique for each method (Devroye et al., 1996). Hold-out testing divides the training data into two sets: one for training and another for

Table 3. Data generation parameters for figure 3.

Parameter	Values
Number of clusters K	50
Range of C $[C_l, C_h]$	[0.0, 1.0]
Range of R $[R_l, R_h]$	[0.0, 0.1]
Range of n $[n_l, n_h]$	[0, 10000]
θ	0.5

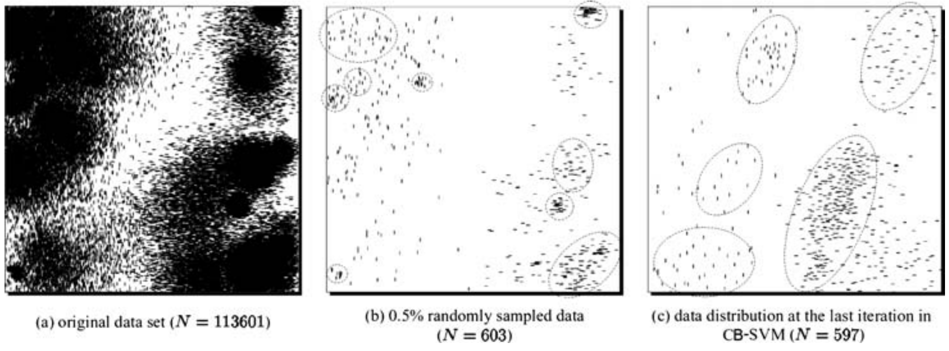

Figure 3. Synthetic data set in a two-dimensional space. '|': positive data; '—': negative data.

validating. In our experiments, we used two-thirds for training and one-third for validating. The testing data is generated separatly. It is computationally efficient while estimating the generalization performance fairly well (Devroye et al., 1996). The alternative, cross validation, takes too long as the size of our data set is very large.

5.1.3. Results and discussion on a "large" data set. Figure 3(a) shows an example data set generated according to the parameters in Table 3. Data generated from the clusters on the left side are positive ('|'), and data on the right side are negative ('—').

Figure 3(b) shows 0.5% randomly sampled data from the original data set of Figure 3(a). As mentioned previously, using random samples for training could hurt the SVM performance. We give two particular reasons below.

1. In practice, the areas around the class boundaries tend to be sparse, because cluster centers (which are dense) are unlikely to cross over the class boundaries. As a result, because random sampling tries to reflect the original data distribution, many points near the cluster centers will be sampled but few points around the boundaries will be included. As figure 3(b) shows, most of the samples came from dense areas in figure 3(a). These points only increase the training time of SVM; they do not contribute to the SVs of the boundaries.

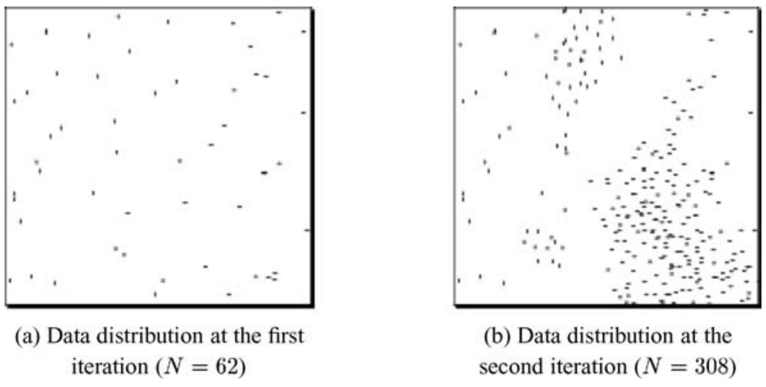

Figure 4. Intermediate results of CB-SVM. '|': positive data; '−': negative data.

2. Random sampling is ineffective when the probability distributions of training and testing data are different. The samples only reflect the distribution of the training data and could miss significant regions of the testing data. We show an example in Section 5.2.

Figure 3(c) shows the training data points at the last iteration in CB-SVM. We set $t_1 = 0.01$, $b = 100$, and the outlier threshold to the standard deviation. It generated a CF tree of $h = 3$, and CB-SVM iterated three times.

Recall that the training data points of CB-SVM are not the actual data but rather summaries of their clusters. Thus they tend not to have narrowly focused data points as it does in the random sampling. Also, the areas far from the boundaries, which are unlikely to contribute to the SVs, have very sparse data points, because the clusters representing those areas would not be declustered in the process of CB-SVM.

Figures 4(a) and (b) show the intermediate data points that CB-SVM generated at the first and second iterations respectively. The data points in figure 4(a) are the centroids of the root entries, which are very sparse. Figure 4(b) shows denser points around the boundary which are declustered into the second level of the CF tree. Finally, figure 3(c) shows an even better data distribution for SVM by declustering the support entries to the leaf level.

For fair evaluation, we generated a testing set using the same clusters and radii but different probability distributions by randomly re-assigning the number of points for each cluster. We report the absolute number of false predictions (# of false negative + # of false positive) on the testing data set, because the data size is so big (compared to the number of false predictions) that the relative accuracy measure would not show much difference between them.

Table 4 shows the performance results on the testing data set. *CB-SVM based on the clustering-based samples outperforms the standard SVM with the same number of random samples*. The "Number of data points" for CB-SVM in Table 4 denotes the number of training data points at the last iteration as shown in Figure 3(c). The "*training time*" for CB-SVM in the table indicates only time spent on SVM training. Notice that it is almost equal to that of 0.5% random samples, reasonable because both methods generated similar number of data points. The "*sampling time*" for CB-SVM indicates the time spent on

Table 4. Performance results on synthetic data set (# of training data = 113,601, # of testing data = 107,072). FP:false positive; FN:false negative; Sampling time for CB-SVM: time for constructing the CF tree.

	Original	CB-SVM	0.5% samples
Number of data points	113601	597	603
SVM Training time (sec.)	163.223	0.003	0.003
Sampling time (sec.)	0.0	10.841	4.301
# of false predictions	69	85	240
(# of FP, # of FN)	(49, 20)	(72, 13)	(217, 23)

Table 5. Data generation parameters for the very large data set.

Parameter	Values
Number of clusters K	100
Range of C $[C_l, C_h]$	[0.0, 1.0]
Range of R $[R_l, R_h]$	[0.0, 0.1]
Range of n $[n_l, n_h]$	[0, 1000000]
θ	0.5

gathering the 597 data points of Figure 3(c). As one would expect, this takes longer than random sampling, because it involves the construction of CF trees.

Note that the long construction time of the CF tree is partly caused by our non-optimized implementation of the hierarchical micro-clustering algorithm. However, as we will show in the next section, this construction cost will become less of an issue when the data size is very large. This is because SVM training is $O(N^2)$ while CF tree construction is only $O(N)$. Thus as N increases, the cost of training will dominate the cost of building the CF tree.

5.1.4. Results and discussion on a "very large" data set. We generated a much larger data set according to the parameters of Table 5 to verify the performance of CB-SVM compared to RAN (random sampling), SEL (Schohn and Cohn, 2000) (selective sampling or active learning with SVM), and ASVM (Mangasarian and Musicant, 2000) (active support vector machine). ASVM is one of the most efficient linear SVM algorithms.[2] Table 6 shows the performance results of these methods on the "very large" data set. We could not run the standard SVM on more than 5% of the entire data set due to our limited resources (i.e., available memory and time). For ASVM, we could only run it with up to 10% of the data set. Running it with 20% exceeded our system resources and generated a system error. We ran SEL 10 times and recorded the average. We discuss SEL and ASVM further in Section 7.

SEL and CB-SVM show lower error rates than RAN and ASVM. The total training time of CB-SVM (T-Time + S-Time) was much shorter than that of SEL, because SEL needs to scan the entire data set at each iteration to select the closest data point. Because SEL requires many iterations to acquire sufficient training data, the I/O cost becomes quite expensive. SEL's error rates are close to that of CB-SVM as their basic idea is similar, *Table 6*. Performance results on the very large data set (# of training data = 23,066,169, # of testing data = 233,890). S-Rate: sampling rate; T-Time: training time (Sec.); S-Time: sampling time (Sec.); ASVM: active SVM; SEL: selective sampling.

S-Rate	# of data	# of errors	T-time	S-time
RAN (0.0001%)	23	6285	0.00012	823.21
RAN (0.001%)	226	2353	0.00098	825.29
RAN (0.01%)	2333	1119	0.03	826.41
RAN (0.1%)	23273	1011	6.41	834.49
RAN (1%)	230380	1009	1189.03	837.98
RAN (5%)	1151714	1008	20754.19	843.28
ASVM (1%)	230380	1161	26.39	838.49
ASVM (5%)	1152901	1031	187.03	831.13
ASVM (10%)	2303769	995	521.38	833.91
SEL	307	952	54841.93	
CB-SVM	2893	876	1.641	2531.35

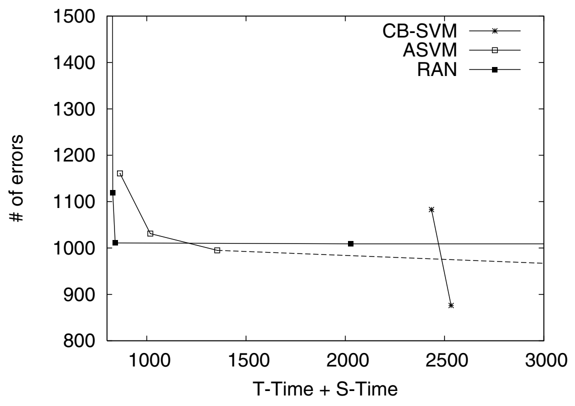

Figure 5. Performance on the very large data set.

which implies that *for large data sets, SVM may perform better with a set of fine quality samples than a large amount of random samples.*[3]

Figure 5 shows the error rates against total training time for RAN, ASVM, and CB-SVM. We vary the sampling rate for RAN, ASVM, and also the parameter t_1 for CB-SVM to trade-off training time and error rate. ASVM can train on more samples than RAN with the same training time. Thus ASVM outperformed RAN for a large data set given the same training time. The dashed line of ASVM in Figure 5 denotes estimated error rates given the previous data; We could not actually test it due to the memory limitation of ASVM. CB-SVM outperforms ASVM as the data becomes very large. The training time of CB-SVM is mainly the CF tree construction time.

5.2. Real data set.

In this section, we experiment on a network intrusion detection data set from the UCI KDD archive which was used for the KDD Cup in 1999.[4] This data set consists of about five million training data and three hundred thousand testing data. As previously noted, CB-SVM works better than random sampling especially when the training and testing data have different distributions. The network intrusion data set is a good example, because the testing data is not from the same probability distribution as the training data, and it also includes attack types not seen in the training data. The datasets contain a total of 24 training attack types, with an additional 14 types in the test data only. This is because they were collected in different time periods, which makes the task more realistic (and more difficult). Our experiments on this data set show that our method based on the clustering-based samples significantly outperforms the random sampling having the same number of samples.

5.2.1. Experiment setup. Each data object consists of 34 continuous features. We normalized the feature values to a value between 0.0 and 1.0 by dividing them by their maximum values. We set $t_1 = 0.5$ for the CF tree because the number of features in this data set is about 50 times larger than that in our synthetic data sets. The outlier threshold was tuned with a lower value, because the outliers in the network intrusions could have valuable information. However, tuning the outlier threshold involves some heuristics depending on the data set and the boundary function. Further definition and justification on the heuristics for specific types of problems is a subsequent future work. We used the linear kernel which performed very well (over 90% accuracy) for this data set.

5.2.2. Results. Our task is to distinguish normal network connections from attacks. Table 7 shows the performance results of RAN, ASVM, SEL, and CB-SVM. Running SVM with a larger amount of samples did not improve the performance much for the same reason as discussed in Section 5.1.4. SEL and CB-SVM generated better results than RAN and ASVM, and the total training time of CB-SVM was much smaller than that of SEL. We run SEL with the same parameters as in Section 5.1.4.

6. CB-SVM for nonlinear classification

In this section, we present a new declustering constraint for SVM nonlinear kernels, so that CB-SVM is also applicable to nonlinear classification by replacing function *getLowMargin*(g,S) (Table 2) in the CB-SVM algorithm of Table 1 with a new *getLowMargin*(g,S) function (Table 8).

The current declustering constraints of CB-SVM are not directly applicable to SVM nonlinear kernels, because the statistical summaries (e.g., centroid and radius) computed in the input space cannot be used in the new feature space transformed by nonlinear kernels: distances in the new feature space are different from those in the input space. Thus, the declustering constraint $D_i - R_i < D_s$ is not meaningful in the new feature space. Computing the radius in the nonlinear kernel space is possible by using the kernel trick for distances introduced in Scholkopf et al. (2000). However, maintaining the centroid in the kernel space cannot be done incrementally. Thus, we introduce an *artificial sampling technique* for the

Table 7. Performance results on the network intrusion data set (# of training data = 4,898,431, # of testing data = 311,029). S-Rate: sampling rate; T-Time: training time (Sec.); S-Time: sampling time (Sec.); SEL: selective sampling.

S-Rate	# of data	# of errors	T-Time	S-Time
RAN (0.01%)	515	25092	0.12	503.11
RAN (0.1%)	4917	24930	6.89	505.91
RAN (1%)	49204	24834	610.49	511.78
RAN (5%)	245364	24829	15938	519.73
ASVM (1%)	49204	25129	109.76	508.93
ASVM (5%)	245364	24887	646.18	512.79
ASVM (10%)	490566	24740	1438.51	513.49
SEL	747	22942	94049.93	
CB-SVM	4090	20938	7.71	4794.84

Table 8. getLowMargin(g, $\mathcal{S}$) for nonlinear kernels.

```
Input: - a boundary function $g$, a entry set $\mathcal{S}$
Output: - a set of the low margin entries $\mathcal{S}'$
Algorithm:
1. $\mathcal{S}' = \emptyset$;
2. Do loop for $s$ in $\mathcal{S}$;
       // return artificial samples created around the border of cluster $s$
   2.1. $BS$ = createBorderSamples($s$);
       // if any of the samples on the border is misclassified, the cluster needs to be declustered.
   2.2. if any of the samples in $BS$ is misclassified by $g$, then $\mathcal{S}' = \mathcal{S}' \cup s$;
3. Return $\mathcal{S}'$;
```

declustering constraint for nonlinear kernels. We present the artificial sampling technique in the Section 6.1 and verify the performance of our CB-SVM using a commonly-used nonlinear classification data sets: the checkerboard (Section 6.2) and the covertype data set (Section 6.3).

6.1. Declustering constraint for nonlinear kernels

An SVM classification function with a nonlinear kernel is a nonlinear curve in input space but a linear function (i.e., a hyperplane) in feature space. To identify the clusters located close to the boundary, we generate *artificial samples* around the border of each cluster and test them using the current boundary function. If the samples generated from a cluster show inconsistent classification results for the current boundary function, it implies that the current boundary passes through the cluster. Such clusters are then considered the *low margin clusters*, which are candidates for declustering. Note that the basic idea here is the

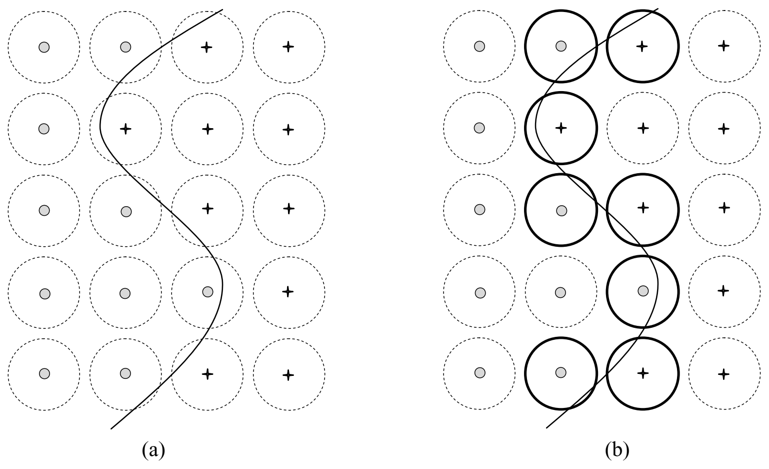

Figure 6. Example of nonlinear boundary function in input space. '+': positive cluster center, 'o': negative cluster center.

same as the linear case: find clusters close to the current boundary function. It is just the discovery process that is different.

To illustrate, consider Figure 6(a) which shows an example of a nonlinear function in a two-dimensional input space. ('+' denotes a positive cluster center and 'o' denotes a negative cluster center.) SVM with a nonlinear kernel may draw a nonlinear function boundary shown in the figure, which correctly classifies two groups of the center points. The boundary passes through eight clusters which are shown in bold circles in Figure 6(b). To determine whether a cluster intersects the boundary function, we generate artificial samples around the border of the cluster. If the boundary passes through the cluster, the artificial samples would show inconsistent classification results. In Figure 6(b), all eight clusters would show inconsistent results: they all have portions of their borders on different sides of the boundary function. To maximize the accuracy of this new declustering constraints while minimizing the number of artificial samples for efficiency, we only sample from the border of each cluster. Since we know the center C and radius R of each cluster, we can easily control the sampling process.

6.2. Experiment on a checkerboard data set

To evaluate the effectiveness of the new declustering constraint for nonlinear kernels, we generated an artificial data set, checkerboard (Figure 8(a)), which is commonly used to evaluate SVM nonlinear kernels, especially the Gaussian kernels. Gaussian kernels are known to be the most complex and powerful.[5]

To create the artificial samples in the checkerboard data, we generated two samples for each dimension of a cluster as described in Figure 7. More samples might be needed as the number of dimensions increases and the boundary becomes more complex. More artificial samples result in longer training time because there are more data to be evaluated to check

```
Input:     - a cluster $s$: cluster center vector $s_c$, radius $s_r$
Output:    - a set of artificial samples $BS$

Algorithm:
1. $BS = \emptyset$;
2. Do loop for each dimension $i$ of $s_c$;
   2.1. Replace $i$th element $e_i$ in $s_c$ with $e_i + s_r$ and it to $BS$;
   2.2. Replace $i$th element $e_i$ in $s_c$ with $e_i - s_r$ and it to $BS$;
3. Return $BS$;
```

Figure 7. Example of *createBorderSamples(s)* which generates two samples for each dimension.

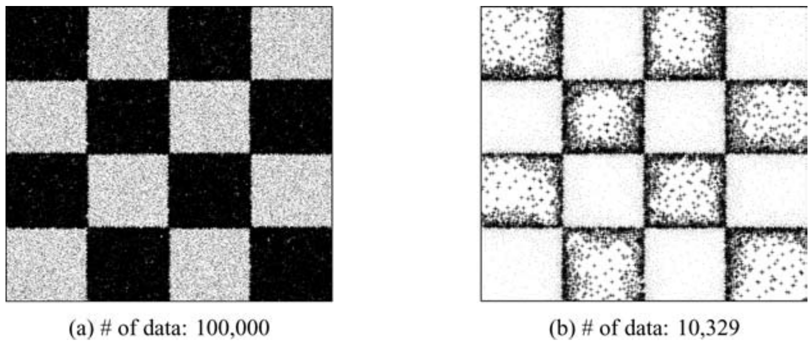

Figure 8. Experiment on a checkerboard data set. '$+$': positive data; '$\cdot$': negative data.

the de-clustering constraint. We chose to sample two data per dimension based on the knowledge that each dimension has two extreme points from the centroid. If we project all the points down to a single dimension, given the centroid C, there exists two points D_1 and D_2 such that $|C - D_1| = |C - D_2| = R$ in that projection. This heuristic performs very well on our nonlinear data sets which have relatively small dimensions.

Using the new declustering constraint for nonlinear kernels, Figure 9(a), (b), and (c) show intermediate stages of CB-SVM on the checkerboard data set. Figure8(b) shows the data distribution after the 4-th iteration, which reduced the total number of training data ten times but generated the same quality of support vectors for the boundary. Thus, the total training time was significantly reduced while the quality of classification function was maintained.

Figure 10 shows the error rates against the training time for RAN and CB-SVM. Since the data distribution of training and testing set are the same, RAN with more samples performed very well. However, CB-SVM required much shorter training times to achieve similar accuracy numbers as it excluded unnecessary data points located far from classification boundary.

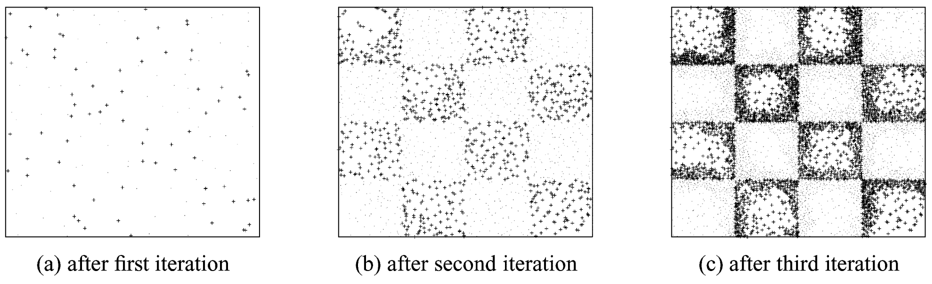

Figure 9. Intermediate results of CB-SVM.

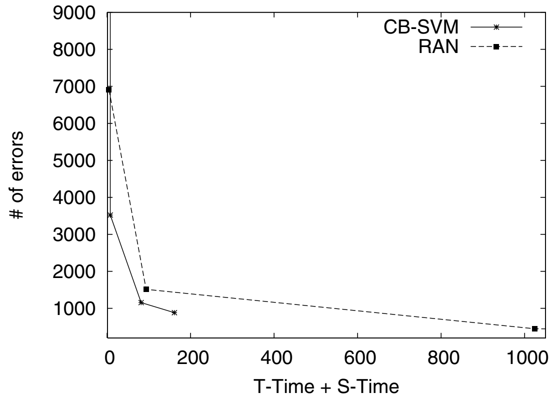

Figure 10. Performance on the checkerboard data set.

6.3. Experiment on a real data set

In this section, we experiment using the covertype data set from the UCI machine learning repository[6]. This data set consists of about 581,000 data objects. We randomly divided them into a big training set and a small testing set to evaluate the training performance of SVM and CB-SVM with a nonlinear kernel. A linear kernel generated very poor performance (e.g., below 40% accuracy) while the Gaussian kernel performed very well (e.g., over 80% accuracy). Table 10 and Figure 11 show the experiment results. RAN performs very well with a higher sampling rate. However, CB-SVM takes a much shorter training time to achieve similar results.

7. Related work

Our work is in some respect related to: (1) SVM fast implementations, (2) SVM approximations, (3) on-line SVM or incremental and decremental SVM for dynamic environments, (4) selective sampling (or active learning) for SVM, and (5) random sampling techniques for SVM.

Table 9. Performance results on the checkerboard data set (# of training data = 100,000, # of testing data = 100,000). Time: training time + sampling time (Sec.).

S-Rate	# of errors	Time
RAN (1%)	6908	4.7
RAN (10%)	1514	93.59
RAN (50%)	448	1023.81
RAN (100%)	259	4109.69
CB-SVM	27830	7.3
	3522	8.8
	1511	82.64
	884	162.39

Table 10. Performance results on the checkerboard data set (# of training data = 569,382, # of testing data = 11630). Time: training time + sampling time (sec.).

S-Rate	# of errors	Time
RAN (1%)	2166	158.09
RAN (2%)	1852	993.46
RAN (10%)	1114	35994.7
RAN (20%)	894	139625
CB-SVM	5378	70.39
	2219	72.92
	1449	868.82

Many algorithms and implementation techniques have been developed for training SVMs efficiently since the running time of the standard QP algorithms grows too fast. Most effective heuristics to speed up SVM training are to divide the original QP problem into small pieces, thereby reducing the size of each QP problem. Chunking, decomposition (Joachims, 1998; Collobert and Bengio, 2001), and sequential minimal optimization (Platt, 1998) are some well-known techniques. Our CB-SVM algorithm runs on top of these techniques to handle very large data sets by condensing the training data into the statistical summaries of data groups such that coarse summaries are made for "unimportant" data and fine summaries are made for "important" data.

SVM approximation is a class of methods that tries to improve the computational efficiency while keeping the semantics intact. It does this by altering the QP formulation to the extent that it keeps the original SVM semantics but makes it easier to solve by a QP solver (Fung and Mangasarian, 2001; Agarwal, 2002). Active SVM (Mangasarian and Musicant, 2000) is a particular example that is especially designed for handling very large data sets. Active SVM is currently known as one of the most efficient linear SVM algorithms for such domains. However, it is limited to linear classification, and it does not perform as well as

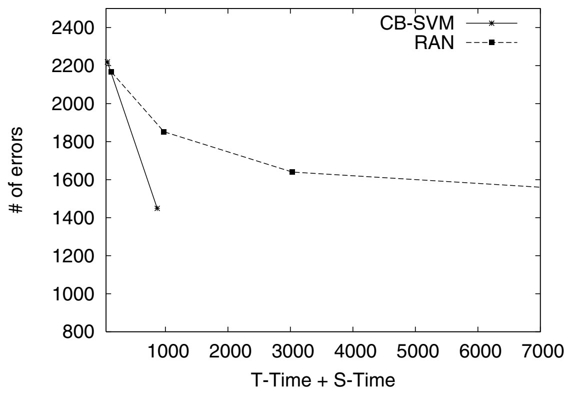

Figure 11. Performance on the covertype data set.

CB-SVM on very large data sets partially because it does not handle the limits of system resources well. Also its training complexity is quadratic to the number of dimensions.

Online SVMs or incremental and decremental SVMs have been developed to handle dynamically incoming data efficiently (Syed et al., 1999; Cauwenberghs and Poggio, 2000; Kivinen et al., 2001). For these methods, a SVM model is usually incrementally constructed and maintained. However, in these scenarios, newer data tend to have a higher impact on the SVM model than older data. More recent data have a higher chance to be the SVs of the SVM model than older data. Therefore, these methods are not suitable for analysis tasks where equal treatment of all data is desired.

Selective sampling or active learning is yet another class of methods. It tries to intelligently sample a small number of training data from the entire data set that maximizes the quality of learning, i.e., learning maximally with a minimum number of data points (Greiner et al., 1996; Tong and Koller, 2000; Schohn and Cohn, 2000). The core of the active learning technique is to select the data intelligently such that the degree of learning is maximized by the data. A common active learning paradigm iterates a training and testing process as follows: (1) construct a model by training an initially given data set, (2) test the entire data set using the model, (3) by analyzing the testing output, select the data (from the entire data set) that will maximize the degree of learning for the next round, (4) accumulate the data to the training data set, and train them to construct another model, and (5) repeat from steps (2) to (5) until the model becomes accurate enough. The idea of the SVM selective sampling is to select data close to the boundary in the feature space at each round, because the data near the boundary have higher chances to be SVs in the next round, i.e., a higher chance to refine the boundary (Tong and Koller, 2000; Schohn and Cohn, 2000). They iterate until there exists no data nearer to the boundary than the SVs. However, an active learning system needs to scan the entire data set at every iteration to select the data, which generates too much I/O cost for very large data sets.

Some random sampling techniques (Watanabe et al., 2001; Lee and Mangasarian, 2001) have been also developed to reduce the training time of SVM for large data sets. For document classification, Bundled-SVM (Shih et al., 2002) is developed to reduce the size of training data by merging multiple documents into one. Other sampling algorithms have been developed to mine data streams (Domingos and Hulten, 2000) and to quickly find interesting patterns in databases (Scheffer and Wrobel, 2002). They draw samples until the result becomes similar to what one would get from the entire dataset.

To our best knowledge, CB-SVM is the only method that tries to maximize the SVM performance given limited system resources.

8. Conclusions and future work

This paper proposes a new method called CB-SVM (Clustering-Based SVM) that integrates a scalable clustering method with an SVM method and effectively runs SVM for very large data sets. Existing SVMs cannot feasibly handle such data sets due to their high complexity on the data size. CB-SVM applies a hierarchical micro-clustering algorithm that scans the entire data set only once to provide an SVM with high quality micro-clusters that carry the statistical summaries of the data. CB-SVM tries to generate the best SVM boundary for very large data sets given bounded system resources. It uses the philosophy of hierarchical clustering where progressive deepening can be conducted when needed to find high quality boundaries for the SVM. Our experiments on synthetic and real data sets show that CB-SVM is very scalable for very large data sets while generating high classification accuracy.

However, CB-SVM suffers in classifying high dimensional data due to the fact that clustering methods do not scale well to high dimensional data. Developing a high dimensional indexing structure for SVM is an interesting direction for future work.

Acknowledgments

The work was supported in part by National Science Foundation under grants No. IIS-02-09199/IIS-03-08215 and an IBM Faculty Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

Notes

1. http://www.csie.ntu.edu.tw/∼cjlin/libsvm
2. See http://www.cs.wisc.edu/dmi/asvm/ for the ASVM implementation.
3. We ran the SEL with $\delta = 5$ (starting from one positive and one negative sample and adding five samples at each round), which gave fairly good results among others. δ is commonly set below ten. If δ is too high, its performance converges slower, which ends up with a larger amount of training data to achieve the same accuracy, and if δ is too low, SEL may need to undergo too many iterations (Schohn and Cohn, 2000; Tong and Koller, 2000).
4. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
5. In machine learning theory, the model complexity or the power of a classification function is often measured by VC-dimension. SVMs with Gaussian kernels have infinite VC-dimensions (Burges, 1998), meaning that it is able to classify arbitrary partitionings of a data set.
6. http://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/

References

Agarwal, D.K. 2002. Shrinkage estimator generalizations of proximal support vector machines. In Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'02), pp. 173–182.

Burges, C.J.C. 1998. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2:121–167.

Cauwenberghs G. and Poggio, T. 2000. Incremental and decremental support vector machine learning. In Proc. Advances in Neural Information Processing Systems (NIPS'00), pp. 409–415.

Chang, C.-C. and Lin, C.-J. 2001. Training nu-support vector classifiers: Theory and algorithms. Neural Computation, 13:2119–2147.

Collobert, R. and Bengio, S. 2001. SVMTorch: Support vector machines for large-scale regression problems. Journal of Machine Learning Research, 1:143–160.

Devroye, L. Gyorfi, L., and Lugosi, G. (Eds.), A Probabilistic Theory of Pattern Recognition. Springer-Verlag, 1996.

Domingos P. and Hulten, G. 2000. Mining high-speed data streams. In Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'00).

Fung, G. and Mangasarian, O.L. 2001. Proximal support vector machine classifiers. In Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'01), pp. 77–86.

Ganti, V. Ramakrishnan, R., and Gehrke, J. 1999. Clustering large datasets in arbitrary metric spaces. In Proc. Int. Conf. Data Engineering (ICDE'98).

Greiner, R. Grove, A.J., and Roth, D. 1996. Learning active classifiers. In Proc. Int. Conf. Machine Learning (ICML'96), pp. 207–215.

Guha, S. Rastogi, R. and Shim, K. 1998. CURE: An efficient clustering algorithm for large databases. In Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD'98), pp. 73–84.

Joachims, T. 1998a. Text categorization with support vector machines. In Proc. European Conf. Machine Learning (ECML'98), pp. 137–142.

Joachims, T. 1998b. Making large-scale support vector machine learning practical. In Advances in Kernel Methods: Support Vector Machines, A.J. Smola B. Scholkopf, C. Burges, (Eds.) Cambridge, MA: MIT Press.

Karypis, G. Han, E.-H., and Kumar, V. 1999 Chameleon: Hierarchical clustering using dynamic modeling. Computer, 32:(8)68–75.

Kivinen, J. Smola, A.J., and Williamson, R.C. 2001. Online learning with kernels. In Proc. Advances in Neural Information Processing Systems (NIPS'01), pp. 785–792.

Lee Y.-J. and Mangasarian, O.L. 2001. RSVM: Reduced support vector machines. In SIAM Int. Conf. Data Mining.

Mangasarian, O.L. and Musicant, D.R. 2000. Active support vector machine classification. Tech. Rep., Computer Sciences Department, University of Wisconsin at Madison.

Platt, J. 1998. Fast training of support vector machines using sequential minimal optimization. In Advances in Kernel Methods: Support Vector Machines, A.J. Smola B. Scholkopf, and C. Burges (Eds.) Cambridge, MA: MIT Press.

Scheffer T. and Wrobel, S. 2002. Finding the most interesting patterns in a database quickly by using sequential sampling. Journal of Machine Learning Research.

Schohn, G. and Cohn, D. 2000. Less is more: Active learning with support vector machines. In Proc. Int. Conf. Machine Learning (ICML'00), pp. 839–846.

Scholkopf, B. Williamson, R.C. Smola, A.J., and Shawe-Taylor, J. 2000. SV estimation of a distribution's support. In Proc. Advances in Neural Information Processing Systems (NIPS'00), pp. 582–588.

Shih, L. Chang, Y.-H. Rennie, J., and Karger, D. 2002. Not too hot, not too cold: The bundled-svm is just right!. In Proc. the Workshop on Text Learning at the Int. Conf. on Machine Learning.

Smola, A.J. and Scholkopf, B. 1998. A tutorial on support vector regression. Tech. Rep., NeuroCOLT2 Technical Report NC2-TR-1998-030.

Syed, N. Liu, H., and Sung, K. 1999. Incremental learning with support vector machines. In Proc. the Workshop on Support Vector Machines at the Int. Joint Conf. on Articial Intelligence (IJCAI'99).

Tong, S. and Koller, D. 2000. Support vector machine active learning with applications to text classification. In Proc. Int. Conf. Machine Learning (ICML'00), pp. 999–1006.

Vapnik, V.N. 1998. Statistical Learning Theory. John Wiley and Sons.

Wang, W. Yang, J., and Muntz, R.R. 1997. STING: A statistical information grid approach to spatial data mining. In Proc. Int. Conf. Very Large Databases (VLDB'97), pp. 186–195.

Watanabe, O. Balczar, J.L. Dai, Y. 2001. A random sampling technique for training support vector machines. In Int. Conf. Data Mining (ICDM'01), pp. 43–50.

Yu, H. Han, J., and Chang, K.C. 2002. PEBL: Positive-example based learning for Web page classification using SVM. In Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'02), pp. 239–248.

Zhang, T. Ramakrishnan, R., and Livny, M. 1996. BIRCH: An efficient data clustering method for very large databases. In Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD'96), pp. 103–114.