Classifying Large Data Sets Using SVMs with Hierarchical Clusters[*]

Hwanjo Yu
Department of Computer Science
University of Illinois
Urbana-Champaign, IL 61801
USA
hwanjoyu@uiuc.edu

Jiong Yang
Department of Computer Science
University of Illinois
Urbana-Champaign, IL 61801
USA
jioyang@cs.uiuc.edu

Jiawei Han
Department of Computer Science
University of Illinois
Urbana-Champaign, IL 61801
USA
hanj@cs.uiuc.edu

ABSTRACT

Support vector machines (SVMs) have been promising methods for classification and regression analysis because of their solid mathematical foundations which convey several salient properties that other methods hardly provide. However, despite the prominent properties of SVMs, they are not as favored for large-scale data mining as for pattern recognition or machine learning because the training complexity of SVMs is highly dependent on the size of a data set. Many real-world data mining applications involve millions or billions of data records where even multiple scans of the entire data are too expensive to perform. This paper presents a new method, *Clustering-Based SVM (CB-SVM)*, which is specifically designed for handling very large data sets. CB-SVM applies a hierarchical micro-clustering algorithm that scans the entire data set only once to provide an SVM with high quality samples that carry the statistical summaries of the data such that the summaries maximize the benefit of learning the SVM. CB-SVM tries to generate the best SVM boundary for very large data sets given limited amount of resources. Our experiments on synthetic and real data sets show that CB-SVM is highly scalable for very large data sets while also generating high classification accuracy.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*

Keywords

support vector machines, hierarchical cluster

1. INTRODUCTION

Support vector machines (SVMs) have been promising methods for data classification and regression [23, 4, 14, 6, 20, 24]. Their success in practice is drawn by its solid mathematical foundations which convey the following two salient properties:

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

The success of SVMs in machine learning naturally leads to its possible extension to the classification or regression problems for mining a huge amount of data. However, despite the prominent properties of SVMs, they are not as favored for large-scale data mining as for pattern recognition or machine learning because the training complexity of SVMs is highly dependent on the size of data set. (It is known to be at least quadratic to the number of data points. Refer [6] for more discussions on the complexity of SVMs.) Many real-world data mining applications involve millions or billions of data records. The following example shows how unscalable a standard SVM is on a large data set.

EXAMPLE 1. *The forest cover type data set from UCI KDD archive*[1] *is composed of 581012 data instances with 54 attributes – 10 quantitative and 44 binary attributes. Figure 1 shows the training time of an SVM on different numbers of training data randomly sampled fron the original data set. From the graphs, we can infer that it would take years for an SVM to train a million data. (We used LIBSVM*[2] *version 2.36, and run the SVM with the RBF kernel which gave fairly good results among others. We ran them using a Pentium III 800Mhz with 906Mb memory.)*

Researchers have proposed various revisions of SVMs to increase the training efficiency by mutating or approximating it. However, they are still not feasible with very large data sets where even multiple scans of the entire data set are too expensive to perform, or they end up losing the benefits of using an SVM by the over-simplifications. (See Section 6 for the discussions on related work.)

[*]The work was supported in part by U.S. National Science Foundation NSF IIS-02-09199, Univ. of Illinois, and an IBM Faculty Award.

[1]http://kdd.ics.uci.edu/databases/covertype/covertype.html
[2]http://www.csie.ntu.edu.tw/∼cjlin/libsvm

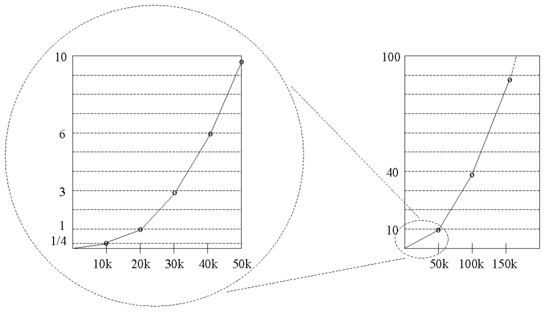


Figure 1: Non-scalability of SVMs. x-axis: # of data points; y-axis: training time in hours

This paper presents a new approach for scalable and reliable SVM classification. The method, called *Clustering-Based SVM (CB-SVM)*, is specifically designed for handling very large data sets. When the size of the data set is large, SVMs tend to perform worse with training from the entire data than training from a fine quality of samples of the data set [19]. Selective sampling (or active learning) techniques with SVMs try to sample the training data intelligently to maximize the performance of SVMs, but they normally require many scans of the entire data set [19, 22] (Section 6). Our CB-SVM using the similar idea applies a hierarchical micro-clustering algorithm that scans the entire data set only once to provide an SVM with high quality samples that carry the statistical summaries of the data such that the summaries maximize the benefit of learning the SVM. CB-SVM is scalable in terms of the training efficiency while maximizing the performance of SVMs.

The key idea of CB-SVM is to use a hierarchical micro-clustering technique to get finer description closer to the boundary and coarser description farther from the boundary, which can be efficiently processed as follows: CB-SVM first constructs two micro-cluster 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 start training an SVM only from the root nodes. Once it generates the "rough" boundary from the root nodes, it selectively decluster only the data summary near to the boundary into lower (or finer) levels using the tree structure. The hierarchical representation of the data summaries is a perfect base structure for CB-SVM to perform the selective declustering effectively. CB-SVM repeats this selective declustering to the leaf level.

CB-SVM can be used for linear classification or regression analysis for very large data sets, including streaming data or data in large data warehouses, especially where random sampling hurts the performance because of infrequently occurring important data or irregular patterns of incoming data, which causes different probability distributions between training and testing data. We discuss this more 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 also generating high classification accuracy. Based on the best of our knowledge, the proposed method is currently the only SVM for very large data sets which tries to generate the best results given limited amount of resources.

The remainder of the paper is organized as follows. We first provide an overview of SVMs in Section 2. In Section 3, we introduce a hierarchical micro-clustering algorithm for very large data sets, originally exploited by T. Zhang et al. [25]. 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. We discuss related work in Section 6 and conclude our study in Section 7.

2. SVM OVERVIEW

In machine learning theory, the "optimal" class boundary function (or hypothesis) $h(x)$ given a limited number of training data set $\{(x, y)\}$ (y is the label of x) is considered the one that gives the best *generalization* performance which denotes the performance on "unseen" examples rather than on the training data. The performance on the training data is not regarded as a good evaluation measure for a hypothesis because the hypothesis ends up *overfitting* when it tries to fit the training data too hard. When a problem is easy to classify and the boundary function is complicated more than it needs to be, the boundary is likely overfit. When a problem is hard and the classifier is not powerful enough, the boundary becomes underfit. SVMs are excellent examples of supervised learning that tries to maximize the generalization by maximizing the *margin* and also supports nonlinear separation using advanced kernels, by which SVMs try to avoid overfitting and underfitting [4, 23]. The *margin* in SVMs denotes the distance from the boundary to the closest data in the feature space.

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

$$
\begin{aligned}
&minimize: \quad W(\alpha) = -\sum_{i=1}^{l} \alpha_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j k(x_i, x_j) \\
&subject\ to: \qquad \sum_{i=1}^{l} y_i \alpha_i = 0 \\
&\qquad\qquad\qquad \forall i: 0 \le \alpha_i \le C
\end{aligned}
$$

The number of training data is denoted by l, α is a vector of l variables, where each component α_i corresponds to a training data (x_i, y_i). C is the soft margin parameter controlling the influence of the outliers (or noise) in 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 polynomial kernel $(x^T x_i + 1)^p$ or RBF kernel $exp(-\frac{1}{2\sigma^2}||x - x_i||^2)$. The use of an advanced kernel is an attractive computational short-cut, which forgoes an expensive creation of a complicated feature space. 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), which allows one to work implicitly with hyperplanes in such highly complex spaces.

Another characteristic of SVMs is that its boundary function is described by the *support vectors* (SVs) which are the data the closest to the boundary. The above QP problem computes a vector α, each element of which specifies the weight of each data, and the SVs is the data whose corresponding α is greater than zero. In other words, the other data rather than the SVs do not contribute to the boundary function, and thus computing an SVM boundary function can be viewed as finding the SVs with the corresponding weights to describe the class boundary.

There have been many attempts to revise the original QP formulation such that it can be solved by a QP solver more efficiently [8, 1]. (See Section 6 for more details.) We do not revise the original QP formulation of SVMs. Instead, we try to provide a smaller but high quality data set that is beneficial to computing the SVM boundary function effectively 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 of SVs that describes the boundary the best.

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 exploited by T. Zhang et al. at 1996 [25], which is named *BIRCH*. The concept of a "micro-cluster" is similar to those in [25, 12], which denotes a statistically summarized representation of a group of data which 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 the single scan of the data does not allow backtracking, localized inaccuracies may exist depending on the order of data input. However, the CF tree captures the major distribution patterns of the data and provides enough information for CB-SVM to perform well.

- It handles noise or outliers (data points that are not part of the underlying distribution) effectively as a by-product of the clustering.

Further improved hierarchical clustering algorithms have been developed including CURE [10] or Chameleon [15]. Chameleon has shown to be very powerful at discovering arbitrarily shaped clusters of high quality, but its complexity is in the worst case $O(n^2)$ 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, but its parameter setting in general has a significant influence on the results. The CF tree of BIRCH carries the spherical shapes of hierarchical clusters and captures the statistical summaries of the entire data set. Thus it provides an efficient and effective structure for CB-SVM to run.

3.1 Clustering Feature and CF Tree

We start from defining 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:

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

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

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

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

DEFINITION 1 (CLUSTERING FEATURE). *[T. Zhang et al. [25]] Given n d-dimensional data points in a cluster: $\{x_i\}$ where i = 1, 2, ... , 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., $\sum_{i=1}^{n} x_i$, and SS is the square sum of the n data points, i.e., $\sum_{i=1}^{n} x_i^2$.*

THEOREM 1 (CF ADDITIVITY THEOREM). *[T. Zhang et al. [25]] 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 that is 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 [25] for the proof.

From the CF definition and additivity theorem, we know 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 of the cluster.

The CF is a summary of a cluster–a set of data points. Managing only this CF summary is efficient, saves spaces significantly, and is sufficient for calculating all the information for building the hierarchical micro-clusters which will facilitate computing an SVM boundary for a very large data set.

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 showing in the right side of Figure 2. 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 subcluster represented by this child. A *leaf entry*, the entry in a leaf node, only has a CF without a child pointer. So, a leaf or a nonleaf node represents a cluster made up of all the subclusters represented by its entries. The threshold t is a constraint for the leaf entries to satisfy such that the radius of an entry in a leaf node has 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 a page size such that a leaf or nonleaf node fit in a page.

This 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 subcluster (which absorbs many data points with radius under a specific threshold t).

3.2 Algorithm Description

A CF tree is built up dynamically as new data objects are inserted. The ways that it inserts a data into the correct subcluster, merges leaf nodes, and manages nonleaf nodes are similar to those in a B+-tree, which can be sketched as follows:

1. **Identifying the appropriate leaf**: Starting from the root, it descends the CF tree by choosing the child node whose centroid is the closest.

2. **Modifying the leaf**: If the leaf entry can absorb the new data object without violating the threshold condition, updates just the CF vector of the entry. If not, add a new entry. If adding a new entry causes a node split, split by choosing the farthest pair of entries as seeds, and redistributing the remaining entries based on the closest criteria.

3. **Modifying the path to the leaf**: It updates the CF vectors of each nonleaf entry on the path to the leaf. Node split in the leaf causes an insertion of a new nonleaf entry into the parent node, and if the parent node becomes split, a new entry is inserted into the higher level node. Likewise, this occurs recursively to the root.

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 can be handled in the original BIRCH algorithm by further refinement with additional data scans. However, we do not perform this further refinement because the infrequent and localized inaccuracy do not impact the performance of CB-SVM much.

3.2.1 Determination of threshold

The choice of the threshold t is crucial for building the tree in the right size which 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 in the memory. T. Zhang 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 [25]. The heuristics for updating t_i is also provided in [25]. Due to space limitations and to keep the focus of the paper, we skip the details of those. In our experiments, we set the initial threshold t_1 intuitively based on the number of data points n and dimensions d and the value range r_d of each dimension such that t_1 is proportional to $n * d * r_d$, and the tree of t_1 mostly fits in the memory.

3.2.2 Outlier handling

After the construction of a CF tree, the leaf entries that contains far fewer data points than average are considered to be outliers. A low setting of outlier threshold can increase the classification performance of CB-SVM especially when the number of data is relatively large compared to the number of dimensions and the type of boundary functions are simple (which is related to having a low VC dimension in machine learning theory) because the non-trivial amount of noise in the training data which may not be separable by the simple boundary function prevents the SVM boundary from converging in the quadratic programming. For this reason, we enabled the outlier handling with a low threshold in our experiments in Section 5 because the type of data we are targetting is of large number of data points with relatively low dimensions, and the type of the boundary functions is linear with 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 a memory can have the $\frac{M}{P}$ nodes at maximum where M is the size of memory and P is the size of a node. The height h of a tree is $log_b \frac{M}{P}$ which is independent of the size of data set. However, if we assume that memory is unbounded and the number of the leaf entries is equal to the number of data points N due to a very small threshold t, then $h = log_b N$.

Insertion of a node into a tree requires the examination of b entries, and the cost per entry is proportional to the dimension d. Thus, the cost for inserting N data points is $O(N * d * b * h)$. 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 in the cost. Then the cost becomes $O(k * N * d * b * h)$ where k is the number of the rebuildings. If we only consider the dependence of the size of data set, 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 objects, and good quality of clustering of the data.

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 being similar to that of selective sampling (or active learning), i.e., selecting the data which maximizes the benefit of learning. The selective sampling for SVMs selects and accumulates the *low margin data* at each round that are close to the boundary in the feature space because the low margin data have higher chances to become the SVs of the boundary for the next round [22, 19]. Appreciating this idea, we decluster the entries near the boundary to get finer samples nearer to the boundary and coarser samples farther from the boundary. In this way, we induce the SVs, the description of the boundary, as fine as possible while keeping the total number of training data points as small as possible.

While selective sampling needs to scan the entire data set at each round to select the closest data point, CB-SVM runs based on the CF tree which can be constructed in a single scan of the entire data set and is carrying the statistical summaries that facilitates constructing an SVM boundary efficiently and effectively. The sketch of the CB-SVM algorithm follows:

1. construct two CF trees from positive and negative data set independently.
2. train an SVM boundary function from the centroids of the root entries – entries in the root node – of the two CF trees. If the root node contains too few entries, train from the entries of the nodes in the second levels of the trees.
3. decluster the entries near the boundary into the next level, and the children entries declustered from the parent entries are accumulated into the training set with the non-declustered parent entries.
4. construct another SVM from the centroids of the entries in the training set, and repeat from step 3 until nothing is accumulated.

The CF tree is a suitable base structure for CB-SVM to perform the selective declustering efficiently. The clustered data also provides better summaries for SVMs than random samples of the entire data set because the random sampling is susceptible to a biased (or skewed) input, and thus it may generate undesirable outputs especially when the probability distributions of training and testing data are not similar, which is common in practice. (The network intrusion data set from the UCI KDD repository that we experiment on in Section 5 is a good example of having substantially different distributions in training and testing data set due to the fact that in the real world, the patterns of network intrusions are very irregular.)

4.1 CB-SVM Description

Without loss of generality, let us consider linearly separable cases for the convenience of explanation.

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 h from the *centroids of the root entries* of T_p and T_n. Note that each entry (or cluster) E_i contains the CF information from which we can efficiently compute the center point C_i and the radius R_i of the cluster. Figure 2 shows an example of the SVM boundary with the root clusters and the corresponding positive tree.

With the boundary function h and the root entries, we determine the *low margin clusters* that are close to the boundary and thus needs to be declustered into the finer level. Let *support clusters* be

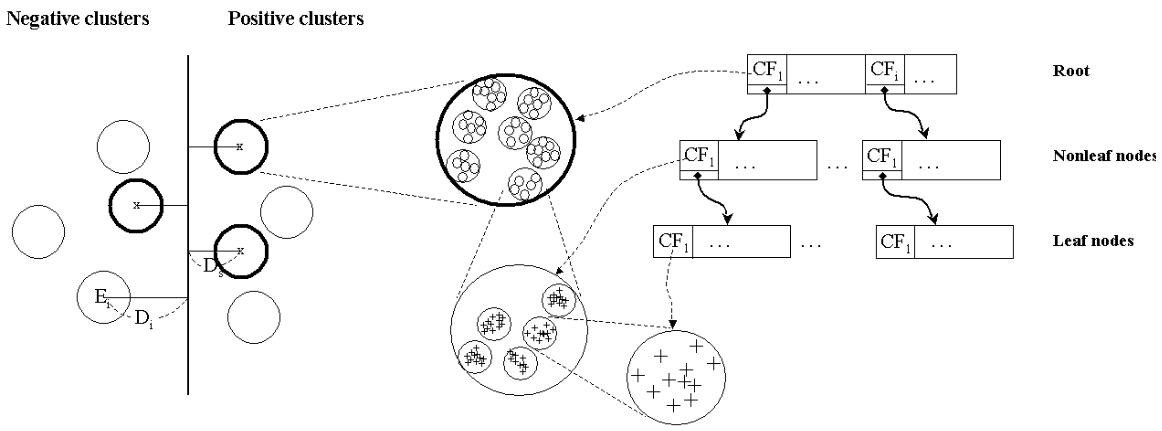


Figure 2: Example of the SVM boundary trained from the root entries of positive and negative trees.

the clusters whose center points are the SVs of the boundary h, e.g., the circles of bold lines in Figure 2. Let D_s be the distance from the boundary to the centroid of a support cluster, and let D_i be the distance from the boundary to the centroid of a 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.

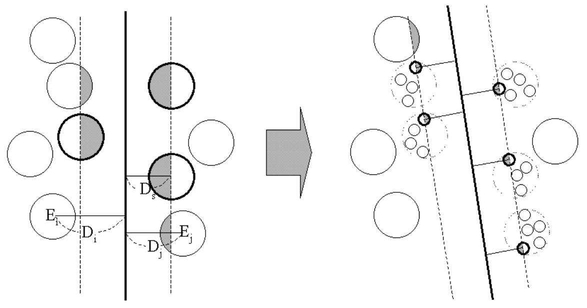


Figure 3: Declustering of the low margin clusters.

The clusters that satisfy the constraint (4) have chances for their subclusters to be the support clusters of the boundary as illustrated in Figure 3 where five clusters initially satisfied the constraint (4) (three of them were the support clusters) and thus were declustered into finer levels, which results in the right picture of Figure 3. The subclusters whose parent clusters do not satisfy constraint (4) would not be the support clusters of the boundary h because the surfaces of the parent clusters are farther than the SVs of the boundary. Thus we have the following remark.

REMARK 1 (DECLUSTERING HARD CONSTRAINT). *Let R_i and D_i be the radius of a cluster E_i and the distance from the boundary to the centroid of the cluster respectively. Given a separable set of positive and negative clusters $E = \{E_i^+\} \cup \{E_i^-\}$ and the SVM boundary h of the set, the subclusters of E_i have the possibilities to be the support clusters of the boundary h only if $D_i - R_i < D_s$, where D_s is the distance from the boundary to the centroid of a support cluster.*

The example we illustrated was a separable case with the hard constraints of SVMs. In practice, the soft constraints are necessary to cope with noise in the training set. Using the soft constraints generates the SVs having different distances to the boundary. For the declustering condition with the soft constraints of SVMs, we replace D_s with D_{ms}, *the maximum distance of all D_s*, which would include all the clusters whose subclusters have the possibilities to be the support clusters of the soft boundary h.

$$D_i - R_i < D_{ms} \tag{5}$$

The subclusters whose parent clusters do not satisfy constraint (5) would not be the support clusters of the soft boundary h because the surfaces of the parent clusters are farther than the most distant SV of the boundary.

REMARK 2 (DECLUSTERING SOFT CONSTRAINT). *For the soft constraints of SVMs, the subclusters of E_i have the possibilities to be the support clusters of the boundary h only if $D_i - R_i < D_{ms}$, where D_{ms} is the maximum distance from the boundary to the centroids of all the support clusters.*

Figures 4 and 5 describe the CB-SVM algorithm with the soft constraints of the declustering.

4.2 CB-SVM Analysis

Building a CF tree from N number of data points costs $O(k * N * d * b * h)$ where d is the number of dimensions, b the number of entries in a node, h the height of the tree, and k the number of rebuildings. Once the CF tree is built, the training time of CB-SVM becomes dependent on the number of entries instead of the number of data points.

Let us assume $t(SVM) = O(N^2)$ where $t(\Psi)$ is the training time of algorithm Ψ. (Note that $t(SVM)$ is known to be at least quadratic to N and linear to the number of dimensions. Refer to [6] for more discussion on the complexity of SVMs.) The number of the leaf entries is at most b^h. Thus, $t(SVM)$ from the leaf entries becomes $O(b^{2h})$.

Let *support entries* be the SVs when the training data are entries in some nodes. Assume that r is the average rate of the number of the support entries among the training entries. Namely, $r = s/b$ where b is the number of training entries and s is the average number of support entries among the training entries, e.g., $r = 0.01$ for $s = 10$ and $b = 1000$. Normally $s << b$ and $0 < r << 1$ for standard SVMs with large data sets.

THEOREM 2 (TRAINING COMPLEXITY OF CB-SVM). *If the number of the leaf entries of a CF tree is equal to the number of*

Input: - positive data set $\mathcal{P}$, negative data set $\mathcal{N}$
Output: - a boundary function h

Notation:
- $HC(\mathcal{S})$: a clustering algorithm that builds a hierarchical cluster tree T from a data set $\mathcal{S}$
- getRootEntries(T): return the root entries of a tree T
- getChildren($\mathcal{S}$): return the children entries of an entry set S
- getLowMargin(h, $\mathcal{S}$): return the low margin entries from a set S which are close to the boundary h (See Figure 5)

Algorithm:

```
1. $T_p = HC(\mathcal{P})$; $T_n = HC(\mathcal{N})$;
2. $\mathcal{S}$ := getRootEntries($T_p$) $\cup$ getRootEntries($T_n$);
3. Do loop
   3.1. $h$ := SVM.train($\mathcal{S}$);
   3.2. $\mathcal{S}'$ := getLowMargin($h$, $\mathcal{S}$);
   3.3. $\mathcal{S}$ := $\mathcal{S} - \mathcal{S}'$;
   3.3. $\mathcal{S}'$ := getChildren($\mathcal{S}'$);
   3.4. Exit if $\mathcal{S}' = \emptyset$;
   3.5. $\mathcal{S}$ := $\mathcal{S} \cup \mathcal{S}'$;
4. Return $h$;
```

Figure 4: CB-SVM

Input: - a boundary function h, a entry set $\mathcal{S}$
Output: - a set of the low margin entries $\mathcal{S}'$

Algorithm:

```
1. $D_{SV}$ := getMaxDistanceOfSVs($h$);
   // return the maximum distance of the support vectors
   from the boundary $h$
2. $\mathcal{S}'$ := getLowerMarginData($D_{SV}$, $\mathcal{S}$);
   // return the data whose margin is smaller than $D_{SV}$
3. Return $\mathcal{S}'$;
```

Figure 5: getLowMargin(h, $\mathcal{S}$)

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 rate of SVs and 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:

$$t(CB-SVM) = \sum_{i=1}^{h} t_i(CB-SVM)$$

where $t_i(CB-SVM)$ 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:

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

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 $s - 1 \approx s$,

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

If we accumulate the training time of all iterations,

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

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

$$t(CB-SVM) = O([b^h r^{h-1}]^2) = O(b^{2h} r^{2h-2})$$

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

Theorem 2 states that CB-SVM trains asymtotically $1/r^{2h-2}$ times faster than a standard SVM given a CF tree. The training time of CB-SVM is asymptotically equal to that of a standard SVM only if all the training data points become the SVs. The rate of the SVs is variant, depending on the type of problems, the type of kernels, the number of dimensions, the number of data points, and the SVM parameters. However, mostly $r << 1$, 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 on CB-SVM using synthetic and real data sets, and we discuss the results. All our experiments are done in a Pentium III 800Mhz machine with 906MB memory.

5.1 Synthetic data set

5.1.1 Data generator

To verify the performance of CB-SVM in realistic environments while providing visualized results, we perform binary classifications on two-dimensional data sets we generated as follows.

1. We randomly created K clusters such that for each cluster, (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 set in the range of $[R_l, R_h]$, and (3) the number of points n in each cluster is also randomly set in the range of $[n_l, n_h]$.

2. We labeled the clusters based on the X-axis value of each cluster such that cluster E_i is labeled as $positive$ if $C_i^x < \theta - R_i$ and $negative$ if $C_i^x > \theta + R_i$, where C_i^x is the X-axis value of the center C_i, and θ is the threshold value between C_l and C_h. We removed the clusters not assigned to either of positive or negative which lie across the threshold θ on X-axis. In this way, we drive the clusters to be linearly separable.

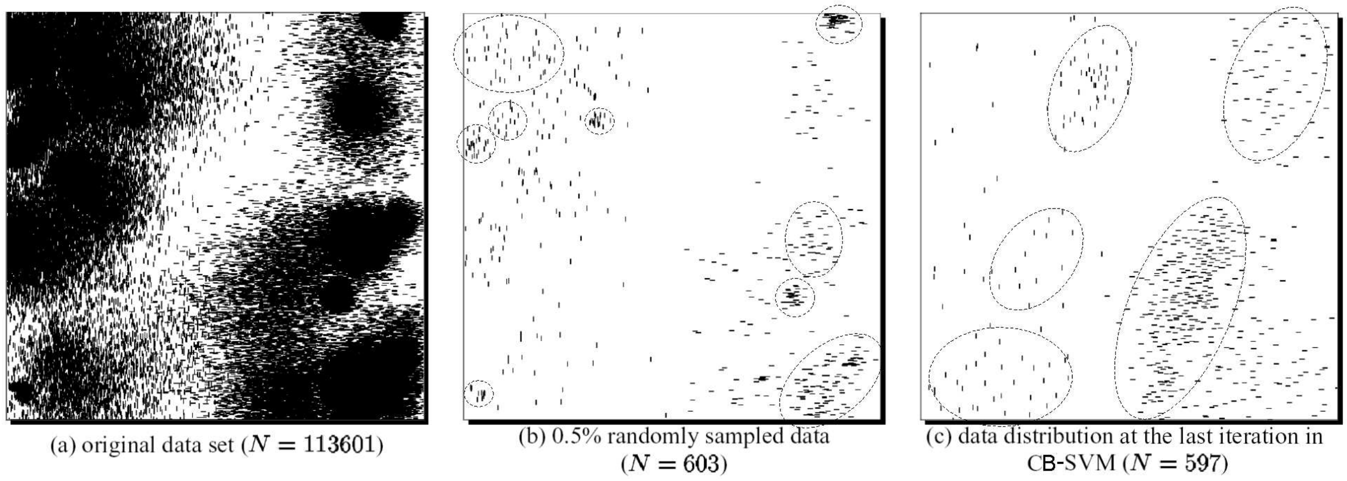


Figure 6: Synthetic data set in a two-dimensional space. '|': positive data; '−': negative data

3. Once the characteristics of each cluster are determined, the data points for the cluster are generated according to a 2-d independent normal distribution whose mean is the center C, and whose standard deviation is the radius R. The class label of each data is inherited from the label of its parent cluster. Note that due to the properties of the normal distribution, the maximum distance between a point in the cluster and the center is unbounded. In other words, a point may be arbitrarily far from its belonging cluster. We refer to the points that belongs to cluster A but located farther than the surface of A as "outsiders". Due to the outsiders, the data set becomes not completely linearly separable, which is more realistic.

5.1.2 SVM parameter setting

We use the LIBSVM version 2.36[3] for SVM implementation and use ν-SVM with linear kernel. We enabled the shrinking heuristics for fast training [13]. 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 [6]. In our experiments, we set ν very low ($\nu = 0.01$ or $\nu = 0.1$) which usually performs very well when the size of data set is large and the noise is relatively small.

5.1.3 Results and discussion on a "large" data set

Figure 6(a) shows an example of the data set generated according to the parameters of Table 1. The data generated from the clusters in the left side and in the right side are positive ('|') and negative ('−') respectively.

Figure 6(b) shows 0.5% randomly sampled data from the original data set of Figure 6(a). Random sampling could hurt the SVM performance in the following ways:

- From Figure 6(b), we know that random sampling reflects the unstable data distribution of the original data set, which includes non-trivial amount of the unnecessary data points for training an SVM. The dashed ellipses on the figure indicating densely sampled areas that reflects the original data distribution are mostly not very close to the boundary. In practice, the areas around the boundary tends to be less dense because cluster centers which are very dense are unlikely to cross over the boundary of multiple classes. Thus the unnecessary data only increases the training time of the SVM without contributing to the SVs of the boundary.

- Random sampling hurts more when the probability distributions of training and testing data are different because random sampling that only reflects the distribution of training data could miss significant regions of the testing data. For instance, in the network intrusion detection data set used for the KDD Cup at 1999[4], the testing data is not from the same probability distribution as the training data. This is because they were collected in different times of periods, which makes the task more realistic. (See Section 5.2 for more details.)

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

Table 1: Data generation parameters for Figure 6

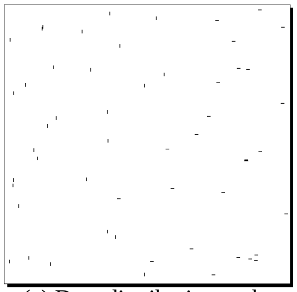
Figure 6(c) shows the training data points at the last iteration in CB-SVM. We set $t_1 = 0.01$ and $b = 100$, and set the outlier threshold with the standard deviation. It generated a CF tree of $h = 3$, and CB-SVM iterated three times.

Note that the training data points of CB-SVM are not the actual data but the summary of the clusters of them, so they tend not to have narrowly focused data points as it does in the random sampling. Also, the areas far from the boundary thus not likely to contribute to the SV will have very sparse data points because the clusters representing those areas would not be declustered in the process of CB-SVM.

Figure 7(a) and (b) show the intermediate data points that CB-SVM generated at the first and second iterations respectively. The data points in Figure 7(a) are the centroids of the root entries, which are very sparse. Figure 7(b) shows dense points around the boundary which are declustered into the second level of the CF tree. Fi-

[3]http://www.csie.ntu.edu.tw/∼cjlin/libsvm

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


(a) Data distribution at the first iteration ($N = 62$)

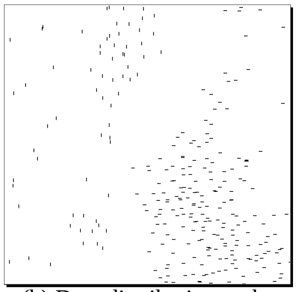

(b) Data distribution at the second iteration ($N = 308$)

Figure 7: Intermediate results of CB-SVM. '|': positive data; '$-$': negative data

	Original	CB-SVM	0.5% samples
Number of data points	113601	597	603
SVM Training time (sec.)	160.792	0.003	0.003
Sampling time (sec.)	0.0	10.586	4.111
# of false predictions (# of FP, # of FN)	69 (49, 20)	86 (73, 13)	243 (220, 23)

Table 2: Performance results on synthetic data set (# of training data = 113601, # of testing data = 107072). FP:false positive; FN:false negative; Sampling time for CB-SVM: time for contructing the CF tree

nally, Figure 6(c) shows a better data distribution for SVMs by declustering the support entries to the leaf level.

For fair evaluation, we generated a testing set using the same clusters and radiuses but different probability distributions by randomly re-assigning the number of points n for each cluster. We report the 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 prediction that the accuracy itself does not show much difference between them.

Table 2 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 2 denotes the number of training data points at the last iteration as shown in Figure 6(c). The "Training time" for CB-SVM in the table indicates the SVM training time on that data, which is almost equal to that of 0.5% random samples since both generated similar number of data points. The "Sampling time" for CB-SVM denoting the time to the construction of the 597 data points of Figure 6(c) definitly takes longer than the random sampling because it involves the construction of a CF tree. (The long construction time of the CF tree is partly caused by our non-optimized implementation of the hierarchical micro-clustering algorithm.)

However, as the data size grows, the random sample size that generates similar accuracies as that of CB-SVM also increases, for which the SVM training time ($O(N^2)$) becomes dominating over the "Sampling time" for CB-SVM ($O(N)$ with a fixed h), and thus the total training time of the SVM with random sampling ends up longer than that of CB-SVM. (See the next section.)

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 3 to verify the performance of CB-SVM compared to random sampling and ASVM (selective sampling or active learning with SVMs) for very large data sets. Table 4 shows the performance results of random sampling, ASVM, and CB-SVM on the "very large" data set. We did not run an SVM on the entire data set since it will take years to finish training. Note that due to the simple linear boundary on the very large amount of training data, random sampling does not increase the performance of SVMs at some point as the sample size increases. *ASVM and CB-SVM showed the error rates around 15% lower than the random sampling of the highest performance. The training time of CB-SVM in total (T-Time + S-Time) was shorter than that of ASVM or the random sampling of the highest performance.* ASVM [19] showed the similar results as ours since the basic idea is similar, which implies that for large data sets, *SVMs perform better with a fine quality of samples than a large amount of random samples.* However, *ASVM takes much longer than CB-SVM for very large data sets that do not fit in the memory because it needs to scan the entire data set at each round to select the closest data point, thereby generating too much I/O cost to undergo as many rounds as it needs to get enough training data.* In this experiment, we ran the ASVM 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 larger amount of training data to achieve the same accuracy, and if δ is too low, ASVM may need to undergo too many rounds [19, 22].) It underwent 61 rounds resulting in 61 times of data scans to sample 307 training data, which took 56872.213 seconds in total for training. We discuss ASVM further in Section 6.

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

Table 3: Data generation parameters for the very large data set

S-Rate	# of data	# of errors	T-Time	S-Time
0.0001%	23	6425	0.000114	822.97
0.001%	226	2413	0.000972	825.40
0.01%	2333	1132	0.03	828.61
0.1%	23273	1012	6.287	835.87
1%	230380	1015	1192.793	838.92
5%	1151714	1020	20705.4	842.92
ASVM	307	865	54872.213	
CB-SVM	2893	876	1.639	2528.213

Table 4: Performance results on the very large data set (# of training data = 23066169, # of testing data = 233890). S-Rate: sampling rate; T-Time: training time; S-Time: sampling time; ASVM: selective sampling

5.2 Real data set

In this section, we experiment on the network intrusion detection data set from the UCI KDD archive which was used for the KDD Cup at 1999[5]. This data set consists of about five millions of training data and three hundred thousands of testing data. As previously noted, CB-SVM works for very large data sets including streaming

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

data or data warehouse analysis especially where random sampling hurts the performance due to infrequent occuring important data or irregular patterns of data incoming which causes different probability distributions of training and testing data. The network intrusion data set is a good application for CB-SVM because the testing data is not from the same probability distribution as the training data, and it also includes specific attack types not 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 times of periods, which makes the task more realistic. (Some intrusion experts believe that most novel attacks are variants of known attacks and the "signature" of known attacks can be sufficient to catch novel variants.) 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 41 features (34 continuous features and 7 symbolic features). We normalized the continuous feature values into between zero and one by dividing them by their maximum values. We created one independent "zero-one" (predicate) feature for each value of the symbolic features such that "one" indicates the existence of the value. Our way of combining multi-variable features may not be the best way for SVMs. Using more sophisticated techniques for pre-processing the features could improve the performance further.

We set $t_1 = 0.5$ for the CF tree because the total number of features in this data set becomes about 50 times larger than that in our synthetic data set and the range of each value is the same. The outlier threshold in this data set was tuned with a lower value because the outliers in the network intrusion data set could have valuable information. However, tuning the outlier threshold involves some heuristics depending on the type of data set and the type of boundary function. Further definition and justification on the heuristics for specific types of problems is a subsequent future work.

We used the same SVM implementation with the same way of optimizing parameters as in the experiments on the synthetic data sets. Linear kernel also showed good performance (over 90% accuracy) in this experiment, which implies the classification on this network intrusion data set is likely separable by a linear function. We briefly discuss the usage of nonlinear kernel in CB-SVM in Section 7.

5.2.2 Results

Our task is to distinguish normal connections from attacks. Table 5 shows the performance results of random sampling, ASVM, and CB-SVM on the network intrusion data set. Running the SVM with the larger amount of samples did not improve the performance much for the same reason as discussed in Section 5.1.4. ASVM and CB-SVM also generated better results than the random sampling, and the total training time of CB-SVM is much faster than that of ASVM. (We run ASVM with the same parameters as in Section 5.1.4.)

S-Rate	# of data	# of errors	T-Time	S-Time
0.001%	47	25713	0.000991	500.02
0.01%	515	25030	0.120689	502.59
0.1%	4917	25531	6.944	504.54
1%	49204	25700	604.54	509.19
5%	245364	25587	15827.3	524.31
ASVM	747	21634	94192.213	
CB-SVM	4090	20938	7.639	4745.483

Table 5: Performance results on the network intrusion data set (# of training data = 4898431, # of testing data = 311029). S-Rate: sampling rate; T-Time: training time; S-Time: sampling time; ASVM: selective sampling

6. DISCUSSION ON RELATED WORK

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

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 [13, 7], and sequential minimal optimization [18] are most well-known techniques. Our CB-SVM algorithm runs on top of these techniques to handle very large data sets by condensing further the training data into the statistical summaries of large data groups such that coarse summary is made for "unimportant" data and fine summary is made for "important" data.

SVM approximations have been attempted to improve the computational efficiency of SVMs by altering the QP formulation to the extent that it keeps a similar semantic of the original SVM while it is faster to be solved by a QP solver [8, 1]. However, their new formulations are still not proven to be efficient and reliable enough to work with very large data sets.

On-line SVMs or incremental and decremental SVMs have been developed to handle dynamically incoming data efficiently [21, 5, 16]. In this senario that an SVM model is incrementally constructed and maintained, the newer data have a higher impact on the SVM model than older data. In other words, recent data have a higher chance to be the SVs of the SVM model than older data. Thus, for the analysis of an archive data which should treat all the data equally, they would generate undesirable outputs.

Selective sampling or active learning is to intelligently sample a small number of training data from the entire data set that maximizes the degree of learning, i.e., learning maximally with a minimum number of data points [9, 22, 19]. 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, (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 (2) to (5) until the model becomes accurate enough.

The idea of selective sampling for SVMs is to select the 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 move the boundary further [22, 19]. 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 round to select the data, which generates too much I/O cost for very large data sets.

Some random sampling techniques [2, 11] developed to reduce the training time of SVMs for large data sets are also based the same idea as the selective sampling which samples the data near the boundary with higher probabilities. They also need to scan the entire data set at each round when the samples are add in. Another method using random sampling [17] was developed for nonlinear SVMs using the random sampling technique in the kernel trick.

Based on the best of our knowledge, our proposed method is currently the only SVM for very large data sets which tries to generate the best results given limited amount of resource.

7. CONCLUSIONS AND FURTHER 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 SVMs for very large data sets. The existing SVMs are not feasible to run such data sets due to their high complexity on the data size or frequent accesses on the large data sets causing expensive I/O operations. 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 such that the summaries maximize the benefit of learning the SVM. CB-SVM tries to generate the best SVM boundary for very large data sets given limited amount of resource based on the philosophy of hierarchical clustering where progressive deepening can be conducted when needed to find high quality boundaries for 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.

CB-SVM is currently limited to the usage of linear kernels since the hierarchical micro-clusters would not be isomorphic to a new high-dimensional feature space once the space is transformed by a nonlinear kernel. In other words, the statistical summaries of data such as radius and distances computed in the input space will not be preserved in the transformed feature space. Constructing effective indexing structures for nonlinear kernels is an interesting direction of future work since it has high practical value especially for pattern recognition of large data sets, such as classifying forest cover types or the pictures from a huge amount of satellite data.

8. REFERENCES

[1] D. K. Agarwal. Shrinkage estimator generalizations of proximal support vector machines. In *Proc. 8th Int. Conf. Knowledge Discovery and Data Mining*, Edmonton, Canada, 2002.

[2] J. L. Balczar, Y. Dai, and O. Watanabe. A random sampling technique for training support vector machines. In *Proc. 13th Int. Conf. Algorithmic Learning Theory*, Washington D.C., 2001.

[3] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is “nearest neighbor” meaningful? *Lecture Notes in Computer Science*, 1540:217–235, 1999.

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

[5] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Proc. Advances in Neural Information Processing Systems*, Vancouver, Canada, 2000.

[6] C.-C. Chang and C.-J. Lin. Training nu-support vector classifiers: Thoery and algorithms. *Neural Computation*, 13:2119–2147, 2001.

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

[8] G. Fung and O. L. Mangasarian. Proximal support vector machine classifiers. In *Proc. 7th Int. Conf. Knowledge Discovery and Data Mining*, San Francisco, CA, 2001.

[9] R. Greiner, A. J. Grove, and D. Roth. Learning active classifiers. In *Proc. 13th Int. Conf. Machine Learning*, Bari, Italy, 1996.

[10] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Seatle, WA, 1998.

[11] O. W. J. L. Balczar, Y. Dai. A random sampling technique for training support vector machines. In *The 2001 IEEE Int. Conf. Data Mining*, San Jose, CA, 2001.

[12] W. Jin, A. K. H. Tung, and J. Han. Mining top-n local outliers in large databases. In *Proc. 7th Int. Conf. Knowledge Discovery and Data Mining*, San Francisco, CA, 2001.

[13] T. Joachims. Making large-scale support vector machine learning practical. In A. S. B. Scholkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[14] T. Joachims. Text categorization with support vector machines. In *Proc. 10th European Conference on Machine Learning*, Chemnitz, Germany, 1998.

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

[16] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. In *Proc. Advances in Neural Information Processing Systems*, Cambridge, MA, 2002.

[17] Y.-J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. In *First SIAM Int. Conf. Data Mining*, Chicago, IL, 2001.

[18] J. Platt. Fast training of support vector machines using sequential minimal optimization. In A. S. B. Scholkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[19] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *Proc. 17th Int. Conf. Machine Learning*, Stanford, CA, 2000.

[20] A. Smola and B. Sch. A tutorial on support vector regression. Technical report, 1998.

[21] N. Syed, H. Liu, and K. Sung. Incremental learning with support vector machines. In *Proc. the Workshop on Support Vector Machines at the International Joint Conference on Articial Intelligence*, Stockholm, Sweden, 1999.

[22] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In *Proc. 17th Int. Conf. Machine Learning*, Stanford, CA, 2000.

[23] V. N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, 1998.

[24] H. Yu, J. Han, and K. C. Chang. PEBL: Positive-example based learning for Web page classification using SVM. In *Proc. 8th Int. Conf. Knowledge Discovery and Data Mining*, Edmonton, Canada, 2002.

[25] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Montreal, Canada, 1996.