An Effective Support Vector Machines (SVM) Performance Using Hierarchical Clustering

Mamoun Awad, Latifur Khan, Farokh Bastani, and I-Ling Yen

Department of Computer Science

University of Texas at Dallas

Richardson, TX 75083-0688

Email: [maa013600, lkhan, bastani, ilyen]@utdallas.edu

ABSTRACT

Support Vector Machines map training data to higher dimensional space, and then find the maximal marginal hyper-plane to separate the data. However, the training time for SVM to compute the maximal marginal hyper-plane is at least $O(N^2)$ with the data set size N, which makes it non-favorable for large data sets. This paper presents a study for enhancing the training time of SVM, specifically when dealing with large data sets, using hierarchical clustering analysis. We use the Dynamically Growing Self-Organizing Tree (DGSOT) algorithm for clustering because it has proved to overcome the drawbacks of traditional hierarchical clustering algorithms (e.g., hierarchical agglomerative clustering). Clustering analysis helps find the boundary points, which are the most qualified data points to train SVM, between two classes. We present a new approach of combination of SVM and DGSOT, which starts with an initial training set and expands it gradually using the clustering structure produced by the DGSOT algorithm. We propose variations of this approach by adding extra steps to disqualify unimportant data points. First, we exclude points based on measuring the distance between data points from different trees. Next, we exclude unimportant data points based on the heterogeneity of the clusters. We compare our approach with the Rocchio Bundling technique in terms of accuracy loss and training time gain using two benchmark real data sets. We show that our proposed variations contribute significantly in improving the training process of SVM with high generalization accuracy and outperform the Rocchio bundling technique.

1. INTRODUCTION

Support Vector Machines (SVM) technique is one of the most powerful classification techniques that was successfully applied to many real world problems [2, 3]. Support Vector Machines are based on the idea of mapping data points to a high dimensional feature space where a separating hyper-plane can be found. This mapping can be carried on by applying the kernel trick which implicitly transforms the input space into another high dimensional feature space. The hyper-plane is computed by maximizing the distance of the closest patterns, i.e., margin maximization, avoiding the problem of overfitting. The separating hyper-plane, i.e., maximal margin, is found using a quadric programming routine which is computationally very expensive. Furthermore, this routine depends on the data set size, taking impractical time when dealing with huge data sets.

Many applications, such as Data Mining and Bio-Informatics, require the processing of huge data sets. The training time of SVM is a serious obstacle for this kind of data sets. According to [4], it would take years to train SVM on a data set of size one million records. Many proposals have been submitted to enhance SVM to increase its training performance [8, 9], either by random selection or by approximation of the marginal classifier. However, they are still not feasible with large data sets where even multiple scans of entire data set are too expensive to perform, or they end up losing the benefits of using an SVM by over-simplification [4].

This paper proposes a new approach for enhancing the training process of SVM when dealing with large data sets. It is based on the combination of SVM and clustering analysis. The idea is as follows: SVM computes the maximal margin separating data points; hence, only those patterns closest to the margin can affect the computations of that margin, while other points can be discarded without affecting the final result. Those points lying close to the margin are called support vectors (see Section 3 for more details). We try to approximate these points by applying clustering analysis.

Traditionally, clustering algorithms can be classified into two main types [2], namely, hierarchical clustering and partitioning clustering. Partitioning, also called flat clustering, directly seeks a partition of the data which optimizes a predefined numerical measure. In partitioning clustering, the number of clusters is predefined, and determining the optimal number of clusters may involve more computational cost than clustering itself. Furthermore, a priori knowledge may be necessary for initialization and the avoidance of local minima. Hierarchical clustering, on the other hand, does not require a predefined number of clusters or a priori knowledge. Hence, we favor hierarchical clustering over flat clustering. Hierarchical clustering employs a process of successively merging smaller clusters into larger ones (agglomerative, bottom-up), or successively splitting larger clusters (divisive, top-down). Agglomerative algorithms are more expensive than divisive algorithms and, since we need a clustering algorithm that grows top-down fashion and is computationally less expensive, we favor the use of divisive hierarchal clustering over agglomerative algorithms.

In the course of our approaches, we do not use the original data set to train SVM; instead, we use the tree node references generated by the clustering algorithm. Since the size of the hierarchical tree is significantly less than the size of the original data set, the training process will be very fast. But this might cause a degradation of the accuracy of the resulting classifier.

However, the de-clustering process we apply would mitigate that by adding new training examples, which are the children of the support vectors. By applying training and de-clustering repeatedly, the training process becomes fast and the training set grows gradually to improve accuracy. From now on when we say de-clustering we mean that the expansion phase of the clustering algorithm is invoked.

We propose three alternatives to train SVM based on the combination of the Dynamically Growing Self-Organizing Tree (DGSOT) algorithm and SVM: First, we generate a hierarchical clustering tree for each class up to a certain level. We use the nodes' references in the upper levels of both trees to train SVM. After training, we compute the support vector references and the accuracy of the classifier. If the accuracy is not satisfied, we de-cluster those support vector references by adding their children to the training set. Second, we add one more search step to determine the support vectors by measuring the distance between nodes from both trees. This step will exclude distant nodes from both trees, based on a certain threshold. Third, we generate only a single hierarchical clustering tree and determine the most qualified nodes to train and de-cluster nodes based on the heterogeneity of nodes. Heterogeneous nodes are those nodes that have data points assigned to them from different classes.

We have reported results here with three benchmark data sets. For example, we have tested our approaches with Rocchio bundling technique, recently proposed to classify documents by reducing the number of data points [15]. Note that both our approaches and the Rocchio bundling method reduce the number of data points before feedings those data points as support vectors to SVM for training. We have observed that our approaches outperform the Rocchio bundling technique in terms of accuracy and processing time. In addition, our approaches perform well with respect to the basic SVM in terms of processing time.

The main contributions of this work are a follows: First, to reduce the training time of SVM, we propose a new support vector selection technique using clustering analysis. Second, we propose various support vector selection strategies based on combining the de-clustering (expansion of cluster) and SVM training phases. Finally, we compare our approaches with a newly proposed data reduction technique, Rocchio Bundling, on two real data sets and demonstrate impressive results.

The organization of the rest of this paper is as follows. In Section 2, we briefly review a number of related works. In Section 3, we briefly review SVM. In Section 4 we briefly present the DGSOT algorithm. In Section 5 we present our approaches in enhancing the training process of SVM. In Section 6, we present our results. In Section 7, we summarize the paper and outline future research directions.

2. RELATED WORK

Support Vector Machines (SVM) have proved a great success in many areas, such as protein classification and face recognition. However, the training time for SVM is at least $O(N^2)$ with the training data set size N, which makes it non-favorable for large data sets. Many methods have been proposed to enhance SVM to increase its training performance [8, 9], either by random selection or by approximation of the marginal classifier. However, they are still not feasible with 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 over-simplification [4].

Random sampling has been used to enhance the training of SVM. Sub-sampling speeds up a classifier by randomly removing training points. Balacazar et al. [16, 17, 18] have used random sampling successfully to train SVM in many applications in the data mining field. Shih et al. [15] have used sub-sampling in classification using Rocchio Algorithm along with other data reduction techniques. Sub-sampling surprisingly has led to an accurate classification in their experiments with several data sets. However, Yu et al. [4] showed that random sampling could hurt the training process of SVM, especially when the probability distribution of training and testing data were different.

Yu et al. [4] have used the idea of clustering, using BIRCH [14], to fit a huge data set in the computer memory and train SVM on the tree's nodes. The training process of SVM starts at the end of building the hierarchical tree causing expensive computations, especially when the data cannot fit in the computer memory or the data is not distributed uniformly. In that case, clustering parameters are tuned and the tree is reconstructed. The main objective of the clustering algorithm is to reduce the expensive disk access cost; on the other hand, our main focus is to approximate support vectors in advance. Furthermore, our use of clustering analysis goes in parallel with training SVM, i.e., we do not wait until the end of building the hierarchical tree in order to start training SVM.

Bagging is another approach [13] in which the data set is partitioned into several partitions. Classifying each partition can be done rapidly because it is done over a relatively small data set. But testing becomes slower, since a test data point needs to be tested against all the classifiers.
Recently, the Rocchio Bundling algorithm has been proposed for the same purpose (more details are given in Section 6.4 and [15]). We show that our approaches outperform the Rocchio bundling algorithm.

3. SUPPORT VECTOR MACHINES OVERVIEW

Support Vector Machines (SVM) are learning systems that use a hypothesis space of linear functions in a high dimensional feature space, trained with a learning algorithm from optimization theory. This learning strategy, introduced by Vapnik and co-workers, is a very powerful method that in the few years since its introduction has already outperformed most other systems in a wide variety of applications. SVM is based on the idea of hyper-plane classifier, or linear separability. Suppose we have N training data points {(x_1, y_1), (x_2, y_2), (x_3, y_3) ... (x_N, y_N)}, where $x_i \in R^d$ and $y_i \in \{+1,-1\}$.

We would like to find a linear separating hyper-plane classifier as in Equation 3.1. Furthermore, we want this hyper-plane to have the maximum separating margin with respect to the two classes. This problem can be formalized as in Equations 3.2 and 3.3.

$$f(x) = sign(w \cdot x - b) \qquad (3.1)$$

$$Minimize_{(w,b)} \ 1/2 w^T w \qquad (3.2)$$

$$Subject\ to\ y_i(w \cdot x_i - b) \geq 1 \qquad (3.3)$$

$$\text{Maximize } L(w,b,\alpha) \equiv \tfrac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i y_i (w x_i - b) + \sum_{i}^{N} \alpha_i \qquad (3.4)$$

$$f(x) = sign(wx + b) = sign((\sum_{i}^{N} \alpha_i y_i (x_i . x) + b) \qquad (3.5)$$

This is a convex quadratic programming problem (in w, b) in a convex set. We can solve the Wolfe dual instead as in Equation 3.4 with respect to α, subject to the constraints that the gradient of $L(w,b,\alpha)$ with respect to the primal variables w and b vanish and $\alpha \geq 0$. The primal variables are eliminated from $L(w,b,\alpha)$ (see [11] for more details). When we solve α_i we can get $w = \sum_{i}^{N} \alpha_i y_i x_i$ and we can classify a new object x using Equation 3.5. Note that the training vectors x_i occur only in the form of dot product; there is a Lagrangian multiplier α_i for each training point, which reflects the importance of the data point. When the maximal margin hyper-plane is found, only points that lie closest to the hyper-plane will have $\alpha_i > 0$ and these points are called support vectors. All other points will have $\alpha_i = 0$ (see Figure 3.1). This means that the representation of the hypothesis/classifier is given only by those points that lie closest to the hyper-plane and they are the most important data points that serve as ***support vectors***. Their values can also be used to give an independent bound on the reliability of the hypothesis/classifier [7]. Figure 3.1 shows two classes and their boundaries, i.e., margins. The support vectors are represented by solid objects while the empty objects are non-support vectors. Notice that the margins are only affected by the support vectors, i.e., if we move the empty objects around such that they do not cross the margins, the margins will not change.

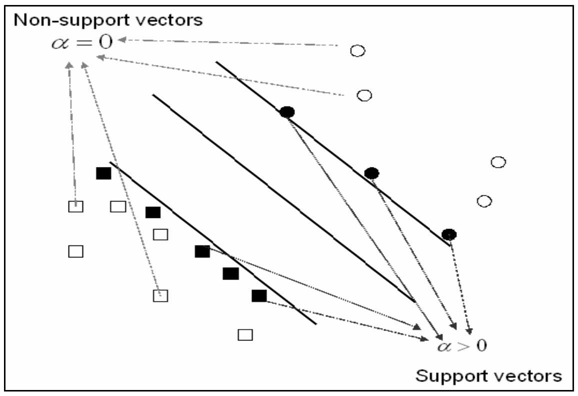


Figure 3.1: The value of α_i for support vectors and non-support vectors

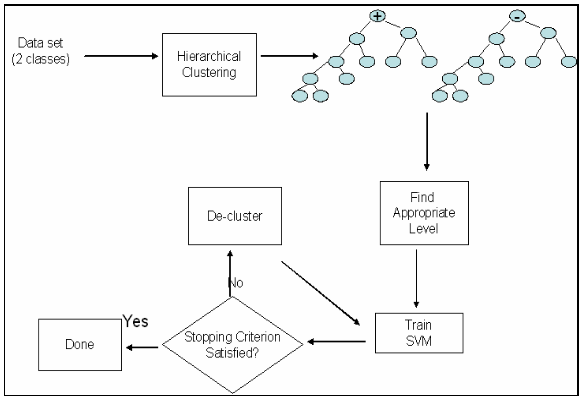


Figure 5.1: Hierarchical clustering with SVM

4. DYNAMICALLY GROWING SELF-GROWING TREE ALGORITHM (DGSOT)

The DGSOT is a tree structure self-organizing neural network. It is designed to discover the correct hierarchical structure in an underlying data set. The DGSOT grows in two directions, namely, vertical and horizontal. In the direction of vertical growth, the DGSOT adds descendents. In the vertical growth of a node *(x),* only two children are added to the node. The need is to determine whether these two children, added to node *x,* are adequate to represent the proper hierarchical structure of that node. Thus, in horizontal growth, the DGSOT strives to determine the number of siblings of these two children needed to represent data associated with the node, *x.* Thus, the DGSOT chooses the right number of children (sub-clusters) of each node at each hierarchical level during the tree construction process. During combined vertical and horizontal growth a learning process is invoked in which data will be distributed among newly created children of node *x.* Further, Luo et al. [12] proposed a K-level up Distribution (KLD) mechanism. The KLD mechanism helps the data incorrectly clustered in early hierarchical clustering stages to have a chance to be re-evaluated during the later hierarchical growing stages. Thus, the final cluster results will be more accurate (see [12] for more details).

5. OUR APPROACH

Our approach for enhancing the training process of SVM is to find those support vectors or approximate them in advance. We use the hierarchical structure produced by the DGSOT algorithm [12] to approximate support vectors. Then we train SVM on a small training set, which is the clusters' references of trees top levels, hence the training process is very fast. After computing the margin, we can determine whether a node is a support vector node or not (see Section 3 for details). Support vector nodes are de-clustered by adding their children nodes to the training set. We repeat this process until a stopping criterion holds true. Therefore, clustering and SVM are mingled with each other. There are several ways we can set the **stopping criteria**. For example, we can stop at a certain level of the tree, or upon reaching a certain number of nodes/support vectors, or when a certain accuracy level is attained. For accuracy level, we can stop the algorithm if the generalization accuracy over a validation set exceeds a specific value (say 98%). For this accuracy estimation, support vectors will be tested with the existing training set based on proximity. Of-course, if the training data set is too large, sampling will be done from the training data set. In our implementation, we adopt the second strategy (i.e., a certain number of support vectors) so that we can compare it with the Rocchio Bundling algorithm which reduces the data set into a specific size. In the following subsections, we present our approaches for support vectors approximation.

5.1 TWO CLUSTERING TREES BASED ON SVM (TCT_SVM)

In this approach, we repeatedly train and de-cluster nodes until we reach a stopping criterion. Figure 5.1 outlines the steps of this approach for enhancing the training process of SVM. First, we process the data set using the DGSOT algorithm to create hierarchical trees for each class up to a certain level to minimize the clustering time. Second, we start from the upper level in the hierarchical tree and use the upper level nodes to train SVM. The choice of what level to start with depends on the number of nodes in that level. Basically, we want to start with a reasonable number of nodes. Third, after each training process, we de-cluster those important nodes, i.e., support vector nodes, by including their children in the training data set. Support Vector nodes can be distinguished by their α_i values (see Section 3 for details). Notice that the number of nodes is relatively small in the top levels of the tree making the training process very fast. Also, based on a stopping criterion, we can decide whether to stop the algorithm or continue before completely building (or expanding) the hierarchical trees.

One subtlety of this approach is that clustering might take long time to finish, which overcomes any benefit of improving SVM. Our strategy here is that we do not wait unit DGOST finishes. Instead, after each epoch/iteration of the DGSOT algorithm which takes a small amount of time, we train the SVM on the generated nodes. After each training process, we can control the growth of the hierarchical tree from top to bottom because non-support vector nodes will be stopped from growing, and only support vector nodes will be allowed to grow. We have updated the DGSOT algorithm to accommodate that. Figure 5.2 shows the growth of one of the hierarchical trees during this approach. The bold nodes represent the support vector nodes. Notice that nodes 1, 2, 3, 5, 6, and 9 were allowed to expand because they are support vector nodes. Meanwhile, we stopped nodes 4, 8, 7, and 10 from growing because they are not support vector nodes. The de-cluster step is very important to increase the number of points in the data set to obtain a more accurate classifier. De-clustering means we are no longer representing data points by their cluster reference; instead, we are considering all the children references of that cluster. Figure 5.3 shows an illustrative example of de-clustering. We start training from level 2, in which we have the training set (+3,+4,+5,+6+7,-3,-4,-5,-6,-7). The dashed nodes represent the clusters' references which are not support vectors. The bold nodes represent the non support vector references. Hence, node +4 and +7 will be replaced with nodes (+11, +12) and (+14, +15, +16), respectively, in the data set. The same applies to node -5 and -7 which will be replaced with (-10, -11) and (-12, -13, -14), respectively. The new data set now is (+3,+11,+12,+5,+6,+14,+15,+16,-3,-4,-10,-11,-6,-12,-13,-14). Figure 5.4 shows the layout of those clusters around the boundaries, and Figure 5.5 shows the effect of de-clustering on the classifiers. Note that when we de-cluster nodes 4, 7,-5, and -7, the old classifier, black lines, is corrected and the new classifier, dashed lines, is more accurate now than the old one. By doing so, the classifier is adjusted accordingly creating a more accurate classifier.

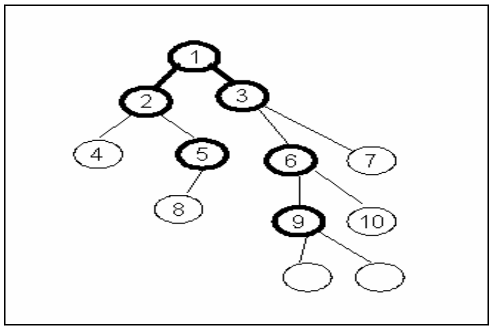


Figure 5.2 Controlling the DGSOT growth

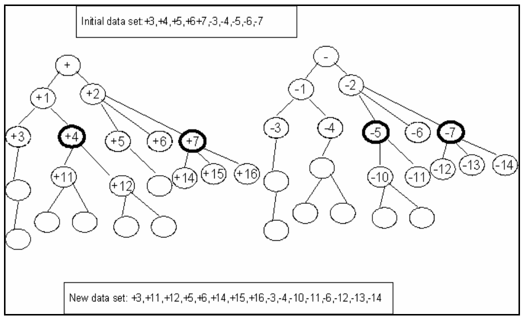

Initial data set:+3,+4,+5,+6+7,-3,-4,-5,-6,-7
+
+1
+2
+3
+4
+5
+6
+7
+11
+12
+14
+15
+16
-
-1
-2
-3
-4
-5
-6
-7
-10
-11
-12
-13
-14
New data set: +3,+11,+12,+5,+6,+14,+15,+16,-3,-4,-10,-11,-6,-12,-13,-14
Figure 5.3: Hierarchical clustering.

Figure 5.4: The layout of clusters of Figure 5.3 around the boundaries.

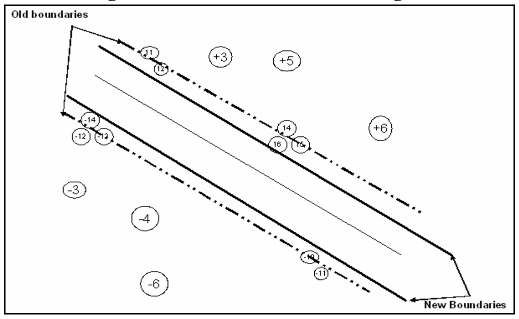


Figure 5.5: The change in the boundaries as a result of de-clustering

5.2 TWO CLUSTERING TREES WITH EXTRA DISTANCE MEASURE SVM (TCTD-SVM)

The TCTD-DVM approach is a variation of the TCT-SVM. We add an extra step before de-clustering. Specifically, we measure the distance between nodes in the training set. Since the distance between nodes lying in the boundary area is the least, we can exclude those nodes having distance more than the average distance. For example, Figure 5.6 shows two hierarchical trees representing two classes. After training, we determine that nodes (+1, +2, +3, -1, -2, -3) are support vector nodes. We measure the Euclidean distance between two nodes, one is a positive node and the other is a negative node in the training set. For example, we measure the distance between [+1, -2], [+1,-2], [+1,-3], [+2,-1], [+2, -2], [+2, -3],…. Ultimately, we will find that nodes +2, +1, +3, -1, and -2 are the closest to each other; furthermore nodes +2 and -3 are distant from every other node in the other tree. Hence, we remove nodes +2 and -3 from the training set or, at least, we do not de-cluster them because they will not make any contribution to the margin computation. Since the distance varies even between support vectors, we can choose those nodes (one from the positive tree and one from the negative tree) with distance less than the average distance between nodes from both trees.

5.3 ONE CLUSTERING TREE BASED SVM (OCT-SVM)

In the OCT-SVM approach, we choose the most qualified nodes in a hierarchical tree to train SVM based on a single tree for the whole data set. Specifically, we study the relationship between the data points in each cluster to find out whether that cluster is a support vector cluster or not. The idea behind this approach is that when we cluster the data set, positive and negative data points together, and generate the hierarchical tree for it, some of the clusters will have negative and/or positive data points assigned to them. We are interested in those clusters having data points from different classes, because those clusters most probably lie in the marginal area between the two classes. But, how can we determine whether a heterogeneous node is positive or negative? We need this because training examples bear classes. Here, we simply use a majority voting mechanism to determine the class of the node. Specifically, if the number of positive data points of a node is greater than the number of negative data points of that node, the node will be declared to be a positive class. We break the tie arbitrarily.

This approach differs from the previous two as follows. First, we do not construct two hierarchical trees, instead we construct only one. Second, we always start from the top level of the tree, and repeatedly train and de-cluster support vector nodes. Since the top levels of the tree span more area in the space, most probably we find that they are heterogeneous. After we find those heterogeneous nodes in an initial level, we de-cluster those nodes. Furthermore, we only include those heterogeneous children in the training set and discard the rest. Figure 5.7 shows one hierarchical clustering tree spanning the data set. The clusters depicted in big circles are the nodes of one of the top levels, say *l* level of the tree. Notice that clusters lying in the margin area are heterogeneous, e.g. clusters A, B, and D. Meanwhile, clusters lying far from the margin area are homogeneous, e.g., clusters E and C. In OCT-SVM, we discard clusters C and E because they are not heterogeneous. Meanwhile, we train SVM on clusters A, B, and D. Furthermore, when we de-cluster a node we do not include all its children. For example, suppose that cluster A is a support vector node; then we just include the heterogeneous sub-clusters 1, and 2, and discard the rest.

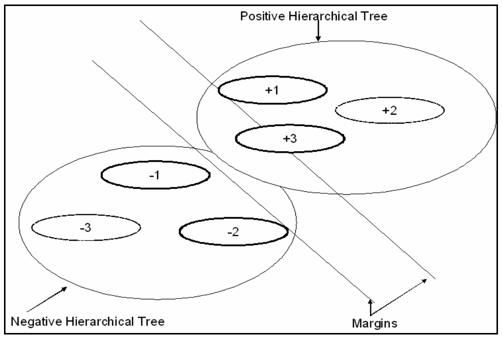


Figure 5.6: Support vectors from both trees are closest.

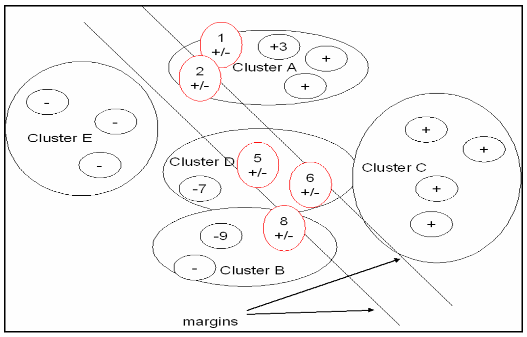


Figure 5.7: heterogeneous clusters in the hierarchical tree

6 EXPERIMENTAL RESULTS

We would like to quantify the relationships between speed and accuracy for our reduction techniques, TCT-SVM, TCTD-SVM, OCT-SVM, and Incremental SVM, with Rocchio Bundling (see Section 6.4 for more details) and random selection techniques. Our implementation of SVM is based on the Kernel-Adatron [19] algorithm using polynomial kernel function.

6.1 EXPERIMENTAL DATA SETS

We have evaluated the performance of our different approaches on several standard classification data sets as well as on synthetic data sets. The first data set is the Yeast data set created by The Institute of Molecular and Cellular Biology, Osaka University [20]. The Yeast data set contains 1484 data points representing protein localization sites. Each data point has 9 features. There are 9 different classes in this data set (see [20] for details). The second data set is the Optical Recognition of Handwritten data which set has 3823 data points; each data point has 64 features. Data points belong to 9 classes in this data set. The data set was extracted from normalized bitmaps of handwritten digits from a preprinted form from a total of 43 people, 30 individuals contributed to the training set and another 13 individuals contributed to the test set (See [20] for more details). The third data set is the waveform data set generated by Wadsworth International Group. The waveform data set has 5000 waves; each belongs to one of three classes. There are 40 attributes, all of which include noise (see [20] for more details).

6.2 BENCHMARK SETTING AND RESULTS

We use the three data sets mentioned in Section 6.1 to train SVM without any reduction techniques. However, we divide the training set into two separate sets, namely, training set (40%) and testing set (60%). We train SVM on the training set and calculate the error rate for both the testing and training. In order to be as fair as possible, we ran SVM 10 times, each time using different training and testing sets, then we calculated the average time and accuracy across all the 10 runs. We consider the results of Table 6.1 as the benchmark to measure the performance of our different approaches. Table 6.1 shows the average training time and average training errors we obtained from training SVM without any reduction techniques. We achieved very good accuracy results for the opt. digits and Yeast data sets, 98% and 86% respectively. We achieved 78% accuracy on the waveform data set. The testing accuracy is calculated over the 60% testing set that we remove (60% for testing, 40% for training).

Table 6.1 training results of SVM without reduction techniques

	Yeast	Opt. Digits	Wave
Training Set size	594	1530	2000
Testing set size	890	2293	3000
Features	9	64	40
Average SVM time (training, testing) in seconds	(18.7,1.8)	(35.1,4.5)	(375.1,47.4)
Average Testing Accuracy (%)	86	98	78

For the next experiments and to measure the performance based on our benchmark results, we use the following definitions: We define the error rate (ER) of the data set X as in Equation 6.1 and 6.2, where N is the number of data points in the set and $x_i \in X$ and ER is the number of mistakes the classifier makes divided by the size of the testing set.

$$ER = \frac{1}{N}\sum_{i}^{N}\delta(x_i) \quad \textbf{(6.1)}$$

$$\delta(x_i) = \begin{cases} 1 & \text{if } x_i \text{ is mis-classified} \\ 0 & \text{otherwise} \end{cases} \quad \textbf{(6.2)}$$

$$PG = \frac{P_N - P_C}{P_N} \quad \textbf{(6.3)}$$

$$A = 1 - ER \quad \textbf{(6.4)}$$

$$GL = \frac{A_C - A_N}{A_N} \quad \textbf{(6.5)}$$

$$\text{Rocchio Score}(x_i) = \vec{x}_i \cdot (\vec{c}_1 - \vec{c}_2) \quad \textbf{(6.6)}$$

$$l(\vec{x}) = \text{sign}(\text{Rocchio Score}(\vec{x}) - b) \quad \textbf{(6.7)}$$

$$D_i' = s_j(P_i) \quad \textbf{(6.8)}$$

$$s_j(D) = \frac{1}{n}\sum_{i=1}^{n} d_{ij} \quad \textbf{(6.9)}$$

$$\text{Rocchio Score}(d_i) = \vec{d}_i \cdot (\vec{c} - \vec{c}') \quad \textbf{(6.10)}$$

We consider the testing set in calculating the testing error rate while in computing the training error rate we consider the training set. However, we only consider the testing error rate to compute the performance gain of a specific reduction technique. We define the performance gain (PG) as in Equation 6.3, where P_C is the training time of SVM using reduction technique and P_N is the training time of SVM without using any reduction technique. Positive PG for a reduction technique T means that there is an improvement/decrease in the training time and negative PG means the opposite. We also define the accuracy A and Generalization Loss, GL, as in Equation 6.5, where ER is the testing error rate, A is the generalization accuracy, A_C is the generalization accuracy of SVM using reduction technique, and A_N is the accuracy of SVM without using any reduction technique. GL is the amount of generalization accuracy gained of using a given reduction technique with SVM. Negative GL for a reduction technique T means that there is a degrading of the generalization accuracy while positive GL indicates an increase in the accuracy.

For the sake of consistency and to be as fair as possible, we only report the results for training sets of size 4%, 12%, and 20% of the size of each data set. In the case of our techniques, we let the hierarchical trees grow, then we train SVM when the size is approximately 4%, 12%, and 12%. For Rocchio Bundling algorithm we reduce the data set using the Rocchio Bundling algorithm to these percentages. We compare our results with the Rocchio Bundling technique and the sample selection technique. We use our own implementation of the Rocchio Bundling technique [15] for data reduction. The Rocchio bundling technique tries to maintain user-chosen statistics of the data (see [15] for more details). Since the Rocchio Bundling algorithm uses the Rocchio Classification algorithm, before discussing how the Rocchio bundling algorithm works, we present the idea of the Rocchio classification algorithm.

6.3 ROCCHIO CLASSIFICATION ALGORITHM

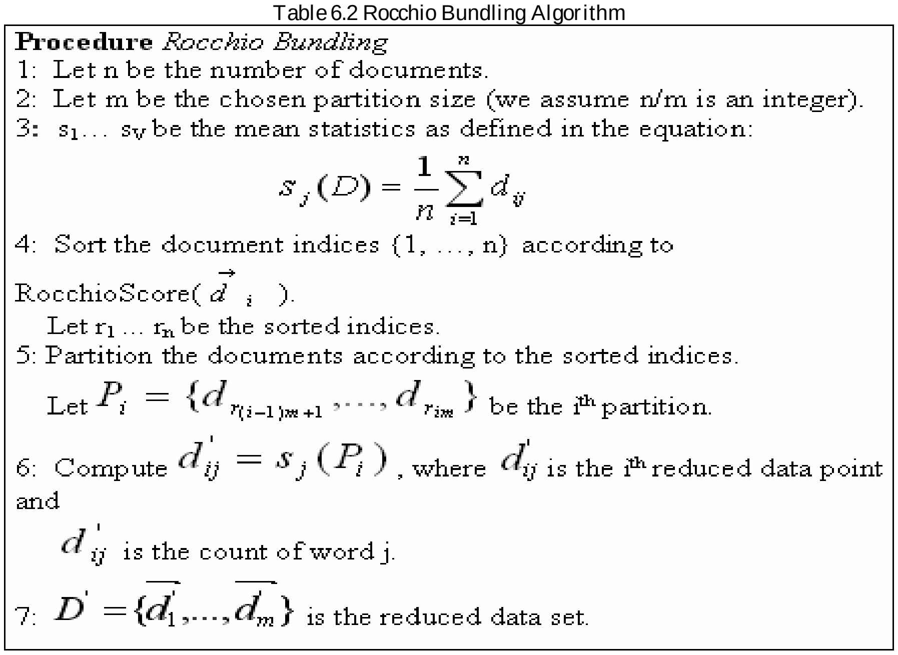
The Rocchio classification algorithm is used in the Rocchio Bundling Reduction technique. For completeness, we briefly review this algorithm in this section.

Consider a binary classification problem. Simply put, Rocchio selects a decision boundary (plane) that is perpendicular to a vector connecting two class centroids. Let $\{\overrightarrow{x_{11}},..., \overrightarrow{x_{1n}}\}$ and $\{\overrightarrow{x_{21}},..., \overrightarrow{x_{2m}}\}$ be sets of training data for positive and negative classes, respectively. Let $\vec{c_1} = \frac{1}{n}\sum_i \overrightarrow{x_{1i}}$ and $\vec{c_2} = \frac{1}{m}\sum_i \overrightarrow{x_{2i}}$ be the centroids for the positive and negative classes, respectively. Then we define the Rocchio score of an example x as in Equation 6.6. One selects a threshold value, *b*, which may be used to make the decision boundary closer to the positive or negative class centroid. Then an example is labeled according to the sign of the score minus the threshold value as in Equation 6.7. Figure 6.1 explains the idea of the Rocchio Classification boundary in the binary classification case. Positive and negative points are presented in oval and square shapes respectively. The Points C_1 and C_2 represent the centroids of positive and negative classes, respectively. The bold line is the Rocchio classifier, and it is clearly the perpendicular line to the line connecting the two class centroids. The Rocchio score represents the distance of a point from the classifier (bold line), for example, Point p_1 has Rocchio score equals to the distance represented by the line connecting it to the Rocchio classifier (bold line).

6.4 ROCCHIO BUNDLING TECHNIQUE

The Rocchio Bundling algorithm is a data reduction technique. Like every reduction techniques, the Rocchio Bundling retains some information and remove others. Bundling preserves a set of k user-chosen statistics, $\vec{s} = (s_1,\ldots s_k)$, where s_i is a function that maps a set of data to a single value. In this implementation, the mean statistic is only used because it has proved to have good performance for Rocchio and Multinomial Naïve Bayes classification.

Table 6.2 Rocchio Bundling Algorithm



```
Procedure Rocchio Bundling
1: Let n be the number of documents.
2: Let m be the chosen partition size (we assume n/m is an integer).
3: $s_1 \ldots s_V$ be the mean statistics as defined in the equation:
       $s_j(D) = \frac{1}{n}\sum_{i=1}^{n} d_{ij}$
4: Sort the document indices {1, ..., n} according to
RocchioScore( $\vec{d}_i$ ).
   Let $r_1 \ldots r_n$ be the sorted indices.
5: Partition the documents according to the sorted indices.
   Let $P_i = \{d_{r(i-1)m+1}, \ldots, d_{r_{im}}\}$ be the i^th partition.
6: Compute $d'_{ij} = s_j(P_i)$ , where $d'_{ij}$ is the i^th reduced data point
and
    $d'_{ij}$ is the count of word j.
7: $D' = \{\vec{d'_1}, \ldots, \vec{d_m}\}$ is the reduced data set.
```

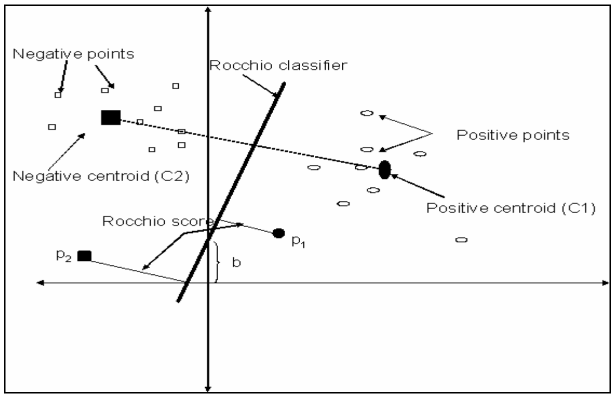


Figure 6.1: Rocchio Classifier

Rocchio Bundling works as follows: We assume that there is a set of training examples for each class. Bundling is applied separately to each class, so we will only discuss what needs to be done for a single class. We select a statistic to preserve and in our experiments we choose the mean statistic for each feature to preserve. Then we partition the full data set D into m equal-sized partitions $P_{1,\ldots,}P_{m.}$ Each partition becomes a single data point in the bundled data set $D^{©} = \{ D_1^{©},\ldots, D_m^{©}\}$.

Each element $D_i^{©}$ is a vector of the mean statistics of the data in partition $P_{i.}$ We calculate the elements of $D_i^{©}$ as in

Equations 6.8 and 6.9, where d_{ij} is the value of the feature *j* of data point *i*. We then feed D' to a classifier of our choosing. Rocchio bundling projects points onto a vector and then partitions points that are near one another in the projected space. For binary classification, that vector is the one that connects the class centroids. Let $\vec{c}$ be the centroid of one class and $\vec{c}\,'$ be the other centroid. Let d_i be the data point. The Rocchio score is the projection of d_i on to $\vec{c} - \vec{c}\,'$ as in Equation 6.10. By sorting data points by their score, the algorithm concentrates the data set into bundles, chooses the first *K* data set; thus we reduce the original data set into a smaller data set (=*K*) that will be used for SVM training.

Table 6.2 present the pseudo code of the Rocchio Bundling algorithm. Step 1 and 2 are the initialization steps for the size of the data set (*n*), and size of each partition (*m*). In step 3, the algorithm computes the mean of each feature as defined in Equation 6.9. For example, s_1 is the mean of the first feature (first value in the vector representing the data points) of all data points. In step 4, the Rocchio scores for each data point are computed as in Equation 6.8. The Rocchio score reflects the distance of each point from the Rocchio decision boundary created as described in Section 6.2. After computing the Rocchio score for each data point in the class, step 5 sort the data points according to their scores. This means that points with close distance to the Rocchio decision boundary will be placed close to each others in the list. Step 6 partitions this sorted list to *n/m* partitions. Step 6 also computes the mean of each partition as a representative of that partition. Finally, in step 7, the reduced data set will be the list of all representatives of all partitions.

6.5 ROCCHIO BUNDLING RESULTS

As we mentioned before, we reduced the data sets to 4%, 12%, and 20% of the original data sets sizes. We report the time consumed in reducing the data sets, and we also add this time to the training time of SVM when computing the performance gain (PG). Table 6.3 shows the results of testing the Rocchio bundling technique on the three data sets. The columns for each part represents the 4%, 12%, and 20% reduced data sets, respectively. Rows represent the training set size, testing set size, reduction time, SVM training time, SVM testing time, performance gain (PG), and generalization loss (GL). The computation of PG and GL is computed using Equations 6.3 and 6.5.

As expected, the performance gain is very good because the training set size is small. However, the generalization loss is high, i.e., accuracy is lost as a result of reducing the training set size. For example, the average accuracy loss over all the training set sizes for the yeast, waveform, and opt. digit data sets are 40%, 17%, and 30%, respectively.

The Rocchio Bundling algorithm condenses the data points close to the Rocchio classifier and represents them by their centroid; hence, data points closer to the margin will be represented by only a few data points. However, the SVM technique depends on the quality of the data points, i.e., those are closer to the margin, and not on the quantity. Because the Rocchio Bundling reduces the number of points closer to the margins, training SVM on the reduced data set did not work well.

Table 6.3 Rocchio Bundling Results

A: Rocchio Bundling Results of Wave data set

	20%	12%	4%
Training set size	1001	626	201
Testing set size	5000	5000	5000
Rocchio Bundling Time (in seconds)	0.7	.07	0.7
SVM time (training, testing) In seconds	(65.5,18.5)	(28.9,11.4)	(3.4,3.8)
Testing Accuracy (%)	55	55	54
PG (%)	82	92	99
AL (%)	-29	-29	-30

B: Rocchio Bundling Results of Opt. Digits data set

	20%	12%	4%
Training set size	765	478	153
Testing set size	3823	3823	3823
Rocchio Bundling Time (in seconds)	0.6	0.6	0.6
SVM time (training, testing) In seconds	(9.9,2)	(4,1.2)	(1.1,1.1)
Testing Accuracy (%)	81	81	81
PG (%)	71	88	96
AL (%)	-17	-17	-17

C: Rocchio Bundling Results of Yeast data set

	20%	12%	4%
Training set size	297	186	60
Testing set size	1484	1484	1484
Rocchio Bundling Time (in seconds)	0.1	0.1	0.1
SVM time (training, testing) In seconds	(12.6,2.8)	(5.2,1.9)	(0.6,0.7)
Testing Accuracy (%)	53	47	46
PG (%)	32	72	96
AL (%)	-38	-45	-46

Table 6.4 Random Selection Results

Part A: Random Selection Results of Opt. Digits dataset

	20%	12%	4%
Training set size	764	456	152
Testing set size	3059	3365	3671
SVM time (training, testing) In seconds	(10.7,2.8)	(5.0,2.1)	(0.9,1.2)
Testing Accuracy (%)	98	98	96
PG (%)	69	85	97
GL (%)	0	0	-2

Part B: Random Selection Results for Yeast Dataset

	20%	12%	4%
Training set size	296	177	58
Testing set size	1188	1307	1425
SVM time (training, testing) In seconds	(5.7,1.06)	(2.17,0.7)	(0.25,2.31)
Testing Accuracy (%)	84	79	80
PG	68	85	98
GL	-2	-8	-6

Part C: Random Selection Results of Wave dataset

	20%	12%	4%
Training set size	999	599	199
Testing set size	4000	4401	4801
SVM time (training, testing) In seconds	(82,20.7)	(32,13.5)	(3.55,4.0)
Testing Accuracy (%)	80	83	80
PG (%)	78	91	99
GL (%)	+2	+6	+2

6.6 RANDOM SELECTION RESULTS

Random selection has been used in many applications to reduce the data set size. In this section, we reduced the three data sets to 4%, 12%, and 20%. In order to achieve better results, we did not select for example 4% from the overall data set. Instead, we randomly selected 4% from the positive data points and 4% from the negative data points. Random selection proves to be a success here. As we can see in Table 6.4, the performance gain (PG) is very high as expected and it is somewhat close to the PG achieved in Rocchio Bundling. For example, the average PG for yeast, opt. digits, and waveform data sets are 83%, 83%, and 89%, respectively. Interestingly enough, the accuracy loss is very low. For all data sets, the reduction of data sets achieved a loss of less than 8%. The largest loss was for the yeast data set using 12% reduction percentage (AL is -8%). Also, in some cases the accuracy has increased. For example, Part C shows that using sample selection has increased the accuracy for the waveform data set, namely, 2%, 6%, and 2% accuracy gain for 4%, 12%, and 20% reduced data set, respectively. The reason that random selection works well here might be because on the one hand we include the whole data set for testing, hence the training data used for training will add to the accuracy. On the other hand, we selected a specific percentage of the positive data sets and the same percentage from the negative data set. This might contribute to having a training set spread over the whole space of the original data set. In contrast, we did not do that during our benchmark results, i.e., we choose 40% of the whole data set (positive and negative) for training and the remaining for testing, not 40% from the positive data points and 40% from the negative data points.

However, Yu et al. [4] showed that random sampling could hurt the training process of SVM, especially when the probability distribution of training and testing data were different.

5.7 CLSUTERING ANALYSIS TECHNIQUES

In this section, we present the setting and the results we obtain for our three techniques, namely, TCT-SVM, TCTD-SVM, and OTC-SVM.

Table 6.5 TCT-SVM technique results on 3 data sets

Part A: TCT_SVM Results of Opt. Digits data set

	20%	12%	4%
Training set size	977	295	125
Testing set size	3823	3823	3823
DGSOT Time (in seconds)	6.0	8.1	5.1
SVM time (training, testing) In seconds	(8.4,2.7)	(2.1,1.9)	(0.7,1.4)
Testing Accuracy (%)	98	98	97
PG (%)	60	71	82
AL (%)	0	0	-1

Part B: TCT_SVM Results of Yeast data set

	20%	12%	4%
Training set size	378	130	51
Testing set size	1484	1484	1484
DGSOT Time (in seconds)	0.7	0.9	0.4
SVM time (training, testing) In seconds	(1.5,0.3)	(0.4,0.2)	(0.2,0.1)
Testing Accuracy (%)	89	91.7	83
PG (%)	87	92	98
AL (%)	+3	+5	-3

Part C: TCT_SVM Results of Wave data set

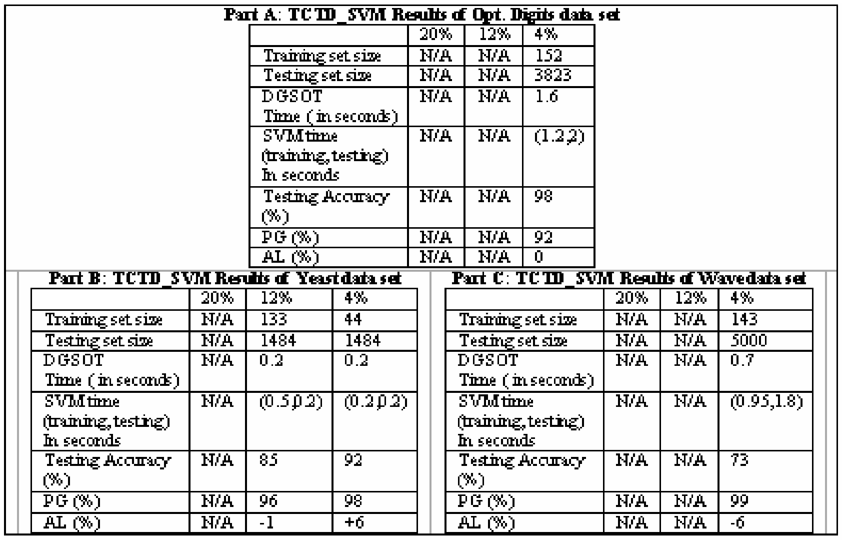
	20%	12%	4%
Training set size	1269	382	126
Testing set size	5000	5000	5000
DGSOT Time (in seconds)	2.7	3.2	2
SVM time (training, testing) In seconds	(10.5,2.8)	2.5,1.8)	(0.6,1.1)
Testing Accuracy (%)	71	73	74
PG (%)	96	98	99
AL (%)	-8	-6	-5

One critical issue is that many researchers avoid using clustering analysis in improving SVM because of the time consumed by the process of clustering itself. In our case, we include SVM training within the process of clustering, such that when we reach a specific tree size (for clustering) or specific accuracy (for SVM), we could stop. The DGSOT algorithm constructs a tree and grows it horizontally and vertically during phases called epochs. We use the nodes generated for each epoch during clustering to train SVM. Specifically, after each epoch during clustering, we use the nodes of the tree as a training set for SVM, we use the original data set for testing, and we report the accuracy loss and performance gain (adding to the performance gain the clustering time for that epoch). Further, after each time we train SVM, we stop the expansion of those nodes that are not support vectors, while allowing the support vector nodes to expand. In doing so, we restrict the DGSOT from growing its hierarchical tree. In the following subsection we present the results of our techniques. We present the results for TCT-SVM, TCTD-SVM, and OTC-SVM in Subsections 6.7.1, 6.7.2, and 6.7.3 respectively.

6.7.1 TCT-SVM RESULTS

The TCT-SVM trains SVM on the nodes of the tree after each epoch. In order to compare our results with other techniques, we only report results when the size of the tree is 4%, 12%, and 20% of the original data set sizes. Table 6.5 shows the results for the three data sets. As we can see, the performance gain is pretty high as expected. The average performance gains over all percentages are: 92%, 97%, and 71% for the yeast, waveform, and opt. digits respectively. Interestingly enough, the accuracy loss is very low and some times in case of yeast data set we actually improved the accuracy. For example, in case of opt. digits data set, we only lost 1% accuracy when training on 4% training set size, while the accuracy stayed intact when training SVM using 12% and 20% training set sizes. In case of waveform data set, we lost atmost 8% of the accuracy, which is acceptable.

Table 6.6 TCTD-SVM Results for 3 data sets.



Part A: TCTD_SVM Results of Opt. Digits data set

	20%	12%	4%
Training set size	N/A	N/A	152
Testing set size	N/A	N/A	3823
DGSOT Time (in seconds)	N/A	N/A	1.6
SVM time (training, testing) In seconds	N/A	N/A	(1.2,2)
Testing Accuracy (%)	N/A	N/A	98
PG (%)	N/A	N/A	92
AL (%)	N/A	N/A	0

Part B: TCTD_SVM Results of Yeast data set

	20%	12%	4%
Training set size	N/A	133	44
Testing set size	N/A	1484	1484
DGSOT Time (in seconds)	N/A	0.2	0.2
SVM time (training, testing) In seconds	N/A	(0.5,0.2)	(0.2,0.2)
Testing Accuracy (%)	N/A	85	92
PG (%)	N/A	96	98
AL (%)	N/A	-1	+6

Part C: TCTD_SVM Results of Wave data set

	20%	12%	4%
Training set size	N/A	N/A	143
Testing set size	N/A	N/A	5000
DGSOT Time (in seconds)	N/A	N/A	0.7
SVM time (training, testing) In seconds	N/A	N/A	(0.95,1.8)
Testing Accuracy (%)	N/A	N/A	73
PG (%)	N/A	N/A	99
AL (%)	N/A	N/A	-6

6.7.2 TCTD-SVM RESULTS

The TCTD-SVM approach is an extension of the TCT-SVM. After each epoch of the DGSOT algorithm, we optimize the nodes of the hierarchical tree by choosing those positive nodes closest to the negative nodes, and those negative nodes closest to the positive nodes. Because of this restriction the hierarchical tree could not grow enough to have many nodes, and sometimes we could not obtain results for 12% and 20% sizes. Table 6.6 shows the results using the TCTD-SVM approach. As mentioned before, some of the hierarchical trees did not grow enough during clustering because of the extra step of distance measurement. The N/A in the Table 6.4 refers to the fact that the tree could not grow to have specific portion of the original data set size. Overall, the results are very promising. For example, as expected the performance gain is very high, the least performance gain is 92%, which is quite good. On the other hand, the accuracy loss (AL) is very similar to the loss in CTC-SVM results. This is because, obviously, this approach is an extension of the CTC-SVM approach. The extra step of distance measurement has benefited the training of SVM

Table 6.7: OTC-SVM Results on 3 data sets.

Part A: OTC_SVM Results of Opt. Digits data set

	20%	12%	4%
Training set size	N/A	N/A	155
Testing set size	N/A	N/A	3823
DGSOT Time (in seconds)	N/A	N/A	6.6
SVM time (training, testing) In seconds	N/A	N/A	(2.4,2.8)
Testing Accuracy (%)	N/A	N/A	83
PG (%)	N/A	N/A	74
AL (%)	N/A	N/A	-24

Part B: OTC_SVM Results of Yeast data set

	20%	12%	4%
Training set size	300	175	60
Testing set size	1484	1484	1484
DGSOT Time (in seconds)	0.81	0.5	0.6
SVM time (training, testing) In seconds	(8.6,2.3)	(2.2,0.8)	(0.4,0.3)
Testing Accuracy (%)	44	59	71
PG (%)	47	85	94
AL (%)	-48	-31	-17

Part C: OTC_SVM Results of Wave data set

	20%	12%	4%
Training set size	N/A	790	262
Testing set size	N/A	5000	5000
DGSOT Time (in seconds)	N/A	8.7	6.3
SVM time (training, testing) In seconds	N/A	(3.9,17.8)	(4.8,5.6)
Testing Accuracy (%)	N/A	82	82
PG (%)	N/A	96	97
AL (%)	N/A	+5	+5

6.7.3 OTC-SVM RESULTS

The OTC-SVM uses only one tree for both negative and positive data points, and bases the support vectors using the heterogeneity idea, i.e., if the cluster has positive and negative data points assigned to it, then it is heterogeneous and might be a good candidate to be a support vector.

As mentioned before, the algorithm starts by training SVM on the heterogeneous nodes. Further, we stop the expansion of the non-heterogeneous nodes because expanding them may not improve the quality of the training set for SVM. As a result, the hierarchical trees sometimes did not grow enough to have 12% or 20% of the original data set size.

Table 6.7 shows the results of applying OTC-SVM on our three data sets. The N/A notation refers to the fact that there were no enough nodes to cover that portion for training SVM.

As expected, the performance gains (PG) are fairly good (notice that we added the DGOST clustering time for that epoch to the training time of SVM). The worse gain was reported for the yeast data set using 20% of the original size. But, in general, the performance gain (PG) is less than other approaches mainly because of the clustering time consumed for checking and setting heterogeneity. For example, after the assignment of all data points during each epoch, we had to go over every node in the tree and compute its heterogeneity. Furthermore, we had to compute whether each node belongs to the negative or positive class using majority vote.

The accuracy loss (AL), however, is also reasonably good. For example, in the waveform data set, AL is -5%. On the other hand, in the yeast data set the AL is very high: -48%, -31%, -17% for 20%, 12%, and 4%, respectively. Accuracy loss reported for opt. digits is also high -24%.

The general conclusion of this approach is that heterogeneity and majority vote techniques are not quite affective to approximate support vector machines. Furthermore, they might add extra overhead to compute them.

7 CONCLUSION AND FUTURE WORK

In this paper, we applied reduction techniques using clustering analysis to approximate support vectors in order to speed up the training process of SVM. The TCT-SVM approach proves to outperform all other techniques in terms of accuracy, but it takes more time. However, the TCTD-SVM did not work well compared to the TCT-SVM because we discarded some of the points that might be important in the general distribution of the original data set. In the case of Rocchio Bundling, the partitions produced were not good representatives as support vectors of the whole data set; hence, it gave fair accurate results. Therefore, we observe that our approaches outperform the Rocchio bundling algorithm. Except for TCT-SVM technique, we observe error rates are high at the beginning of the training process because the size of training set is very small. As the training sets get bigger, during de-clustering, the error rates go down. However, the error rates settle on specific level even if the training set size increase.

There are several issues that can be further investigated in separate research directions. First, an incremental adaptation of the SVM can be proposed to handle incremental sources of data. Second, the choice of kernel function in mapping data points to feature space can be further investigated to be used in both clustering analysis and SVM. Finally, other techniques, such as Randomized algorithms, can be used in approximating support vectors.

8 REFERNCES

[1] The Nature of Statistical Learning Theory, Vapnik, V., Springer-Verlag; 1999, ISBN: 0387987800.

[2] Cortes, C. (1995) Prediction of Generalization Ability in Learning Machines., Ph.D. Thesis, Department of Computer Science, University of Rochester.

[3] LeCun, Y., Jackel, L. D., Bottou, L., Brunot, A., Cortes, C., Deker, J.S., Drucker, H., Guyon, I., Muller, U.A., Sackinger, E., Siard, P. and Vapnik, V., (1995)Comparison of Learning algorithms for handwritten recognition , International Conference on Artificial Neural Networks, Fogelman, F. and Gallinari, P. (Ed.), pp. 53-60.

[4] Hwanjo Yu, Jiong Yang, and Jiawei Han, "Classifying Large Data sets Using SVM with Hierarchical Clusters". SIGKDD '03, August 24-27, 2003, Washington, DC, USA.

[5] A. Ben-Hur, D. Horn, H.T. Siegelmann and V. Vapnik: Support Vector Clustering. Journal of Machine Learning Research 2:125-137, 2001

[6] Ying Zhao and George Karypis., Prediction of Contact Maps Using Support Vector Machines. Third IEEE Symposium on Bioinformatics and Bio-Engineering (BIBE'03) March 10 - 12, 2003 Bethesda, Maryland

[7] Michael P. S. Brownz,William Noble Grundyz, David Lin, Nello Cristianini, Charles Sugnet, Manuel Ares Jr., David Hausslerz, Support Vector Machine Classification of Microarray Gene Expression Data, technical report UCSC-CRL-99-09, University of California, Santa Cruz.

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

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

[10] Ma, Q., Wang, J. T. L. and C. H. Wu. (2000). Application of neural networks to biological data mining: A case study in DNA sequence classification. Proceedings of International Conference on Software Engineering and Knowledge Engineering SEKE-2000, 23-30.

[11] Introduction to Support Vector Machines, N. Cristianini and J. Shawe-Taylor,, Cambridge University Press 2000 ISBN: 0 521 78019 5

[12] F. Luo, L. Khan , F. Bastani, I-Ling Yen and J. Zhou, A Dynamical Growing Self-Organizing Tree (DGSOT) for Hierarchical Clustering Gene Expression Profiles, The Bioinformatics Journal, Oxford University Press, UK.

[13] Breiman, L. (1996)., Bias, variance and arcing classifiers, Technical report 460, Statistics Department, University of California.

[14] T. Zhang, R. Ramakrishnan, M. Livny, BIRCH: An Efficient Data Clustering Method for Very Large Databases, SIGMOD Conference 1996, pp. 103-114.

[15] L. Shih, Y D. M. Rennie, Y. Chang, D. R. Karger, Text Bundling : Statistics-based Data Reduction, Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

[16] J. L. Balcazar, Y. Dai, O. Watanabe, Provably Fast Support Vector Regression Using Random Sampling, 2001 IEEE International Conference on Data Mining, IEEE Computer Society 2001, pp. 43-50.

[17] D. Tufis, C. Popescu, R. Rosu,, Automatic Classification of Documents by Random Sampling, Proceedings of the Romanian Academy Series. Volume 1, Number 2/2000, pp. 117-127.

[18] G. Feng, O.L. Mangasarian, Semi-Supervised Support Vector Machines for Unlabeled Data Classification, Optimization Methods and Software. 1-14(2001) Kluwer Academic Publishers, Boston.

[19] Thilo Friess, Nello Cristianini, Colin Campbell. The Kernel-Adatron: a Fast and Simple Learning Procedure for Support Vector Machines, Proceeding of the Fifteenth International Conference on Machine Learning (*ICML*), 1998, pg. 188-196

[20] http://www.ics.uci.edu/~mlearn/MLSummary.html